Linux com a sistema operatiu de temps real

Treball de final de carrera

Enginyeria Informàtica

Gener del 2013

Consultor: Francesc Guim Bernat

Alumne: Carles Marsal Castillero

Departament d'Arquitectura de Computadors i Sistemes Operatius.

Universitat Oberta de Catalunya

Índex de continguts

Índex de figures

Índex de taules

1 Introducció

1.1 Motivació

En l'última dècada hem pogut observar com el sistema operatiu Linux ha estès el seu domini dels servidors fins als ordinadors de sobretaula i els sistemes encastats. En aquest camp hem vist com múltiples companyies deixaven d'utilitzar sistemes operatius propietaris i començaven a incorporar Linux en els seus dispositius electrònics, des de petits i senzills equips portàtils fins a potents nodes de commutació en xarxes de proveïdors de serveis.

Algunes de les raons que han animat a aquestes companyies a realitzar un canvi d'aquesta envergadura són: la possibilitat de disposar d'una plataforma de desenvolupament robusta, escalable, fàcilment adaptable, estandarditzada i lliure de la dependència del codi tancat i propietari; el suport d'una gran quantitat d'aplicacions i maquinari, i un ampli recolzament per part dels fabricants d'unitats de processament.

Fins arribar a aquest punt ha estat necessari, però, superar alguns reptes que els sistemes encastats imposaven i pels que Linux no havia estat inicialment dissenyat: el temps d'arrancada, la possibilitat de perdre l'alimentació en qualsevol moment o certes restriccions de maquinari com l'espai reduït de memòria de programa (*footprint*), la limitació del nombre d'escriptures dels dispositius d'emmagatzemament d'estat solid, la mancança en alguns casos d'una unitat de gestió de memòria (MMU) i d'altres.

En particular, és especialment rellevant l'esforç realitzat per dotar a Linux, i més concretament el seu nucli, de comportament en temps real, propietat imprescindible per a molts dels sistemes encastats desplegats en sectors com l'industrial, el mèdic o l'automoció. En aquest aspecte el nucli ofereix a data d'avui diverses configuracions alternatives que, segons alguns dels seus autors[1] i diverses proves realitzades[2], satisfan requeriments de temps molt exigents.

Davant d'aquest escenari, les companyies de desenvolupament de sistemes encastats es troben amb la possibilitat de considerar Linux com una alternativa en el disseny dels seus equips i una de les primeres preguntes que han de resoldre és si aquest serà capaç de complir amb les restriccions de temps requerides pel sistema a desenvolupar.

En aquesta línia, aquest projecte pretén tant donar a conèixer quines són les possibles optimitzacions encarades a l'assoliment de propietats de temps real com proposar un marc metodològic per a l'assaig d'una plataforma de maquinari donada. Això ens ha de permetre quantificar quina és la magnitud de cadascuna d'aquestes millores i quin és el seu impacte en el rendiment de l'equip per finalment poder avaluar de forma objectiva fins a quin punt és viable la utilització de Linux en un escenari amb unes requeriments temporals determinats.

---

1 Segons Robert Love, contribuïdor del nucli de Linux, "the Linux kernel 2.6 is capable of meeting stringent timing requirements." ([LOV10]:64)

2 En aquest aspecte, la Open Source Automation Development Lab (OSADL), una associació formada per fabricants de components electrònics dedicada a la promoció i suport de l'ús de programari lliure en la industria de l'automatització industrial, realitza sistemàticament tests de càrrega i rendiment sobre cada versió de nucli produïda amb diferents configuracions de maquinari i distribucions de Linux. [OSA13]

1.2 Objectius

El propòsit d'aquest projecte és el de determinar i avaluar les configuracions alternatives del nucli del sistema operatiu Linux enfocades a la millora del comportament en temps real. Amb aquest propòsit es plantegen els següents objectius:

- Entendre de manera precisa que és el tems real, quines imposicions planteja i en quines aplicacions és pot trobar.
- Conèixer el funcionament intern del sistema operatiu Linux pel que fa als aspectes crucials en l'assoliment del temps real.
- Avaluar les optimitzacions disponibles a dia d'avui per millorar el comportament en temps real (reactivitat i determinisme) de nucli de Linux.
- Disposar d'una plataforma de maquinari i programari preparada per la mesura d'aquestes millores.
- Establir unes mètriques, condicions i mètode per a l'assaig repetit de la plataforma sota configuracions diferents.
- Extreure coneixement del procés realitzat i dels resultats obtinguts que permeti avaluar de manera objectiva la viabilitat de Linux com a sistema operatiu de temps real.

1.3 Abast

Determinat per l'objectiu final del projecte així com per la limitació en la durada i cost del mateix es restringeix l'abast a l'estudi i avaluació de les capacitats de temps real de la línia principal del nucli de Linux (vainilla) de la sèrie 2.6[3] amb l'addició del pedaç RT.[4]

A més, per tal d'evitar la complexitat analítica afegida en el balanceig de càrrega i assignació d'afinitat en sistemes multiprocessador (SMP), és restringeix el projecte a un sistema uniprocessador (UP), d'un sol nucli i que no utilitzi tècniques *hyperthreading*[5].

No es considera l'avaluació d'alternatives de nucli dual utilitzades, també, per la obtenció de propietats de temps real en sistemes Linux mitjançant la introducció d'una capa de micro-nucli per sota del propi nucli de Linux que gestiona els esdeveniments de temps real de manera prioritària. [BER10]

1.3.1 Plataforma d'avaluació

L'avaluació de les capacitats de temps real s'efectuarà sobre el següent maquinari:

- Netbook DELL Inspiron 910 (Intel ATOM N270 a 1,6Ghz[INT08])
- 1GB de SDRAM DDR2 a 553Mhz
- Unitat d'estat solid de 32GB.

Creiem que aquesta plataforma posseeix unes característiques força interessant pel que fa als experiments que volem realitzar: reflecteix prou bé el que podria ser un sistema en mòdul (SOM) com els que s'utilitzen en alguns sistemes encastats (processador de baix consum, *fanless*, potència de comput mitjana, ús d'una unitat d'estat sòlid...), utilitza una arquitectura prou estesa i suportada

3 Que inclou les versions de nucli numerades com 3.x.y. [TOR11]
4 Pedaç destinat a millorar el comportament en temps real de la línia principal de nucli de Linux. [KER13B]
5 Tècnica de maquinari consistent en la creació de diversos nuclis virtuals a partir d'un nucli físic mitjançant la utilització de diferents components de la unitat de processament no utilitzats en un fil d'execució determinat, degut al bloqueig produït per certes dependències de maquinari creades per aquest propi fil, per a l'execució d'un fil alternatiu.

(IA-32 de la família x86) i té unes facilitats properes a les d'un ordinador de sobre taula (dispositius d'interfície d'usuari, fins a 32GB de capacitat d'emmagatzemament...)

1.4 Anàlisi de riscos

Es valoren els riscos associats al projecte i se'ls associa una acció destinada a evitar-los o, en cas de no assolir-ho, pal·liar-los.

Taula 1: Anàlisi de riscos

Risc	Probabilitat	Impacte	Acció
Problemes en arrancar les imatges de nucli creades.	Mitjana	Moderat	Planificar amb temps per reaccionar.
Complicacions amb l'entorn de proves o eines de benchmarking al utilitzar nuclis poc testats.	Mitjana	Moderat	Planificar de manera folgada per tenir temps per corregir.
Pèrdua de feina en proves llargues (>12h) necessàries per trobar un worst case representatiu.	Mitjana	Alt	Començar amb un joc de proves de curta durada (<4h) i finalitzar amb una única prova llarga (>12h) per a cada configuració del nucli. Fer impressions temporals dels tests. Col·locar la implementació d'una aplicació d'exemple com a última activat ja que es tracta d'un objectiu secundari.

1.5 Planificació

Es realitza la planificació inicial del projecte en les següents parts: desglòs de les activitats, estimació de l'esforç i cost, i elaboració d'un calendari en diagrama de Gantt.

1.5.1 Desglòs d'activitats

- Cerca d'informació: cerca d'informació inicial per conèixer les possibilitats així com els mecanismes utilitzats per assolir temps real a Linux.
- Exposició teòrica: selecció, síntesi i exposició dels conceptes fonamentals del temps real així com els mecanismes adoptats pel sistema operatiu Linux per proveir-ne.
- Compilació i validació d'imatges de nucli: selecció d'una versió vainilla del nucli amb suport al pedaç CONFIG_PREEMPT_RT. Aplicació del pedaç i compilació segons opcions desitjades.
- Marc metodològic: Establiment de les mètriques, estadístiques i condicions per l'assaig i avaluació de les diferents opcions de nucli.
- Preparació de l'entorn de proves: selecció i prova d'eines de *benchmarking*. Disseny i prova d'un entorn d'estrès.
- Execució de les proves: execució de les proves en els diferents nuclis compilats amb optimitzacions distintes.
- Implementació i prova d'exemple RT: implementació d'una senzilla aplicació amb tècniques de temps real. Execució i verificació del seu comportament (finalment no s'ha realitzat).
- Anàlisi de resultats: anàlisi i exposició de conclusions sobre els resultats extrets.
- Documentació: redacció de la memòria i elaboració de la presentació.

1.5.2 Estimació d'esforços

Considerant una persona i una jornada de 2,5 hores, i tenint en compte les limitacions temporals del projecte,[6] es realitzar la següent estimació i distribució d'esforços.

Taula 2: Estimació d'esforços

Tasca	Dies	Hores / dia	Hores
Cerca d'informació	14	2	28
Exposició teòrica	14	2	28
Compilació i proves dels nuclis	10	2	20
Preparació del entorn de proves	10	2	20
Execució de les proves	10	2	20
Implementació d'un exemple	5	2	10
Anàlisi dels resultats	5	2	10
Documentació	68	0,5	34
	27	2,5	67,5
		TOTAL	**237,5**

1.5.3 Estimació de costos

Es distingeixen dos tipus de costos associats al projecte: la realització de les activitats i l'adquisició del maquinari.

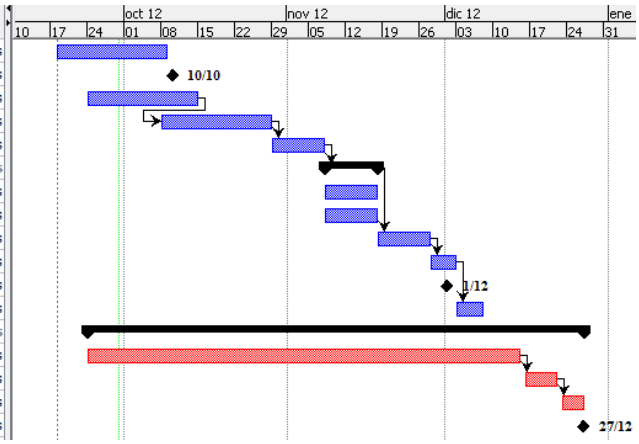
- Realització d'activitats: si tenim en compte un salari brut de 30.000 €/any per un conveni de 1.750 hores/any i 237,5 hores per executar el projecte el seu cost seria de 4.071 €.
- Maquinari: el model de *netbook* emprat, tot i esta fora de catàleg, té un cost d'adquisició en el mercat de segon mà que oscil·la entre els 200 i 325 $[7].

1.5.4 Diagrama de Gantt

Es realitza el calendari del projecte per a una sola persona que realitzarà una jornada de 2'5 hores al dia i tenint en compte les dependències entre les diferents activitats.

Figura 1: Diagrama de Gantt

	Nombre	Inicio	Terminado	Duracion
1	Definició formal	18/09/12 8:00	9/10/12 8:00	21 days
2	PAC1	10/10/12 8:00	10/10/12 8:00	0 days
3	Cerca d'informació	24/09/12 8:00	15/10/12 8:00	21 days
4	Exposició teòrica	8/10/12 8:00	29/10/12 8:00	21 days
5	Compilació i prova d'imatges de kernel-RT	29/10/12 8:00	8/11/12 8:00	10 days
6	**Preparació d'entorn de proves**	**8/11/12 8:00**	**18/11/12 8:00**	10 days
7	Selecció i prova d'eines de benchmarking	8/11/12 8:00	18/11/12 8:00	10 days
8	Disseny i prova d'entorn d'estrés	8/11/12 8:00	18/11/12 8:00	10 days
9	Execució de les proves	18/11/12 8:00	28/11/12 8:00	10 days
10	Implementació i prova d'exemple RT	28/11/12 8:00	3/12/12 8:00	5 days
11	PAC2	1/12/12 9:00	1/12/12 9:00	0 days
12	Anàlisi de resultats	3/12/12 8:00	8/12/12 8:00	5 days
13	**Documentació**	**24/09/12 7:00**	**27/12/12 9:30**	95 days
14	Elaboració memòria	24/09/12 7:00	15/12/12 9:30	83 days
15	Elaboració presentació	16/12/12 7:00	22/12/12 9:30	7 days
16	Revisió	23/12/12 7:00	27/12/12 9:30	5 days
17	ENTREGA FINAL	27/12/12 7:00	27/12/12 7:00	0 days



6 Segons el pla d'estudis vigent (2003) entre 225 i 270 hores.
7 Aquests preus s'han pogut obtenir de diversos venedors de segona mà registrats a la tenda virtual Amazon a data de 09/10/2012.

1.6 Organització del document

El cos del present document està organitzat en el següents capítols:

- El temps real: on es defineix què és temps real.
- El sistema operatiu Linux: on s'exposen els principis i mecanismes claus dels sistemes operatius multitasca pel que fa a la gestió de processos així com la implementació que en fa Linux d'aquestos.
- Linux com a sistema operatiu de temps real: on s'analitzen els factors involucrats en l'assoliment del temps real així com els elements introduïts en la serie 2.6 i el pedaç RT que hi tenen una aportació significativa.
- Avaluació del temps real a Linux: on s'exposa el marc metodològic emprat per l'avaluació i mesura de la capacitat de resposta en temps real de diverses configuracions de nucli triades, i es presenten i analitzen els seus resultats.
- Conclusions, aportacions i línies obertes: on s'exposen les conclusions extretes de l'experiència i els resultats del projecte, s'analitza el grau d'assoliment i s'enumeren possibles extensions del mateix.

Finalment aquesta memòria inclou, també, un capítol bibliogràfic així com un annex on es poden trobar el conjunt complet de mesures realitzades, els passos seguits en l'aplicació del pedaç PREEMPT_RT i la configuració de les diferents opcions del nucli, diversos *scripts* creats per la generació de càrregues i mesura i altres afegits que poden ser d'interès per aquell lector que conèixer amb més detall el treball realitzat en aquest projecte.

2 El temps real

2.1 Conceptes bàsics

Abans de poder establir que s'entén per temps real i quins tipus de temps real es poden trobar en el domini dels sistemes computacionals és necessari definir una sèrie conceptes. En aquest capítol és presenten aquests conceptes i és dona una definició de què és el temps real formulada a través d'aquestos. Finalment, és presenta una de les classificacions més habituals pel que fa a aquests sistemes.

2.1.1 Latència

S'anomena latència al temps transcorregut entre l'instant t en què hauria d'ocórrer un succés i l'instant t' en el que aquest realment succeeix $L = t - t'$. En el domini del temps real fa referència al temps que transcórrer entre un esdeveniment i l'atenció del mateix .

2.1.2 Jitter

Es defineix *jitter* -en anglès tremolor- com les variacions de magnitud donades en la repetició d'un fenomen determinat. En el domini del temps real fa referència a la variació de la latència, es sol mesurar respecte a la mitjana aritmètica o un valor ideal i es presenta en termes estadístics com la desviació típica o la variància

2.1.3 Worst Case

El *worst case* (pitjor cas) és el resultat més desfavorable per a un objectiu donat. En el domini del temps real s'utilitza en referència tant al valor màxim de latència en una resposta com el de temps d'execució (WCET) de la mateixa.

2.1.4 Determinisme

Determinisme és el fet de saber amb anticipació quan de temps durarà una operació i el fet que aquest temps es compleixi. En el temps real fa referència a la capacitat del sistema operatiu i les tasques que executa de comportar-se de manera predictible de manera que es pugui determinar, en el pitjor cas, quina serà la latència i temps d'execució màxims experimentats.

2.1.5 Termini

Es defineix termini com l'instant en el temps en el que ha d'haver ocorregut una acció determinada. En el domini del temps real fa referència al temps màxim per finalitzar l'execució d'una tasca lligada a un esdeveniment des de que aquest s'ha produït.

2.2 Sistema de temps real

Un cop establerts els conceptes fonamentals, es diu que un sistema és de temps real si te la capacitat de donar resposta als esdeveniments succeïts (senyals externes, temporitzadors...) dins uns terminis determinats, normalment associats a processos externs al sistema, del mon real.

Factors que influeixen en l'assoliment d'aquest objectiu són la latència i el *jitter*, pel que fa a l'atenció de l'esdeveniment, així com el temps d'execució, pel que fa a la finalització de la resposta.

Per poder garantir el compliment d'un termini es necessari que el sistema gaudeixi d'un cert determinisme. En tot cas, de manera empírica o formal s'ha de poder establir el pitjor cas, la latència i el *jitter* que seràn els paràmetres que estabiliran el grau d'assoliment del temps real del sistema.

Segons el grau d'utilitat i l'impacte de l'incompliment dels terminis el temps real es pot classificar en dur (*hard*), estricte (*firm*) o tou (*soft*). En la pràctica, en un mateix sistema, solen coexistir, però, tasques amb requeriments de temps real diferents, tot i que un sistema sol agafar habitualment el nom del requisit més exigent que presenta.

2.2.1 Tasques *hard real time*

Una tasca de temps real dur es caracteritza pel fet que l'ocurrència d'una resposta fora de termini invalida el sistema. A més, la fallada del sistema pot tenir greus conseqüències com la pèrdua de bens o persones. Alguns exemples de sistemes de temps real dur són controls de navegació, aparells mèdics o sistemes de control industrial crítics.

2.2.2 Tasques *firm real time*

Una tasca de temps real estricte es caracteritza pel fet que l'ocurrència d'alguna resposta fora de termini no invalida el sistema però tampoc li aporta valor. No hi han conseqüències fatals associades. Alguns exemples de sistemes de temps real estrictes són aplicacions de predicció o sistemes de control o regulació no crítics.

2.2.3 Tasques *soft real time*

Una tasca de temps real tou es caracteritza pel fet que l'ocurrència de respostes fora de termini no invaliden el sistema però li resten valor. La reducció de qualitat és acceptable o recuperable, tot i que no és desitjable. Alguns exemples de sistemes de temps real tou són aplicacions d'àudio i vídeo, telecomunicacions o jocs.

3 El sistema operatiu Linux

3.1 Introducció

Linux és un sistema operatiu multitasca de propòsit general de tipus Unix per a sistemes sistemes uniprocessador (UP) o multiprocessador (SMP). Permet l'execució simultània de diverses tasques mitjançant compartició de temps via preempció, és a dir, la suspensió involuntària d'una tasca.

Està composat per: un nucli monolític[8] -programa únic, executant-se com un procés únic, en un espai d'adreçament únic-, encarregat de gestionar els recursos; els controladors, encarregats la interacció directa amb el maquinari, i les llibreries, utilitats i aplicacions d'usuari, utilitzades per a l'ús i administració del sistema.

En aquest capítol -elaborat a partir de la literatura docent existent, [LOV10], [LOV07] i [MAR04] de manera significativa- es presenten els aspectes més rellevants dels sistemes operatius, i més concretament la implementació que en fa Linux d'ells, pel que fa a l'assoliment del temps real, tema que s'abordarà més particularment en el següent capítol. Paral·lelament s'introdueixen breument alguns conceptes teòrics al voltant de la planificació així com alguns elements arquitectònics de maquinari necessaris per donar una imatge completa de les necessitats i dificultats de la implementació d'un sistemes de temps real.

3.2 Arquitectura del sistema

A Linux, com a la majoria de sistemes operatius moderns, el nucli s'executa en un mode elevat amb un espai de memòria protegit i accés complet al maquinari. A aquest espai i mode se'l designa com *espai de nucli*. Alternativament, les aplicacions d'usuari, s'executen en el denominat *espai d'usuari* on només poden realitzar un subconjunt restringit d'operacions.

Les aplicacions d'usuari, per tant, quan necessiten d'interactuar amb el sistema, s'hi comuniquen mitjançant les *crides de sistema* que, alhora, provoquen una interrupció de programari que situa el processador en mode elevat i cedeix el control al nucli perquè en pugui realitzar les operacions restringides en benefici de l'aplicació.

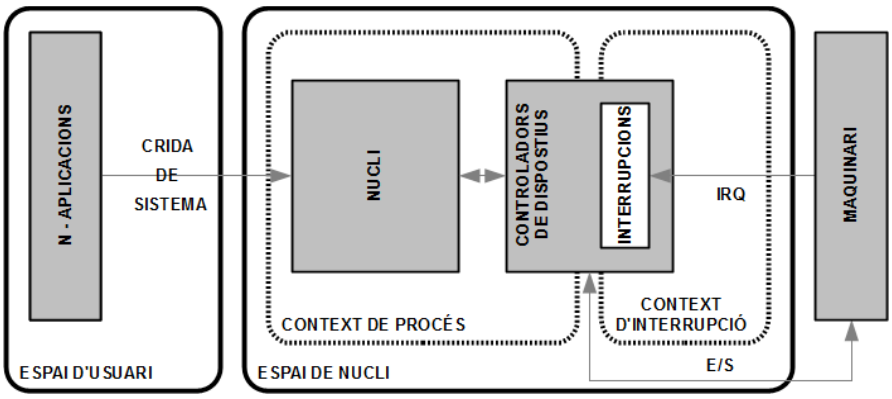
El nucli, a més, en les seves funcions de gestió del maquinari, implementa els mecanismes d'atenció a les interrupcions, senyals asíncrones produïdes pel maquinari per la notificació de la finalització d'alguna operació o l'aparició d'algun esdeveniment donat.

En resum, es diu que que l'execució de codi pot estar realitzant alguna de les tres següents activitats:

- En espai d'usuari, executant codi d'usuari d'un procés determinat.
- En espai de nucli i en context de procés, en benefici d'un procés determinat.
- En espai de nucli i en context d'interrupció, atenent una interrupció.

8 En contrast amb els *micronuclis* que implementen únicament serveis de planificador, gestor de memòria i interrupcions, i comunicació entre-processos; i deixen la resta de serveis (sistemes d'arxiu, controladors de dispositius...) a l'espai d'usuari.

Figura 2: Arquitectura simplificada del Sistema Operatiu



3.3 Processos i fils d'execució

Un procés és la instància d'un programa en execució. Inclou el seu codi així com un conjunt de recursos relacionats: estat del procés, espai d'adreçament, variables globals, descriptors de fitxers o senyals pendents d'atenció, entre altres. Els processos s'executen en un context de memòria i processador (CPU) virtuals proveïts pel sistema operatiu.

La virtualització de memòria s'assoleix, per una banda, mitjançant la unitat de gestió de memòria (MMU) un dispositiu de maquinari que realitza la traducció d'una adreça de memòria virtual, expressada en planes, a una de física. La MMU ofereix, a més, funcions de protecció de memòria de manera que un procés no pugui accedir a la memòria d'una altre. La virtualització de processador, per altra banda, s'assoleix mitjançant la compartició de temps de CPU assolida amb el desallotjament voluntari (cessió) o involuntari (preempció) del procés en execució

Un procés en Linux, a més, està composat per un o més fils d'execució, seqüències d'instruccions que conformen la unitat mínima de planificació del sistema i el context dels quals està format per l'estat dels registres de la CPU, una pila pròpia, i senyals i prioritat independents. Els fils d'un mateix procés comparteixen el mateix espai d'adreçament i, per tant, no gaudeixen de virtualització de memòria entre ells. Per contrapartida, el canvis de context -salvat del context del fil a substituir i restauració del context del fil substitut- entre fils d'un mateix procés són menys costos.

3.4 Gestió de processos

La gestió de processos es pot dividir en tres activitats principals: creació, gestió de l'estat i finalització. La majoria de sistemes operatius moderns realitzen aquestes activitats de manera molt similar tot i que existeixen petites diferències segons el model particular de gestió. A continuació es descriu de manera general la implementació realitzada al nucli 2.6 de Linux.

3.4.1 Creació d'un procés

La creació d'un procés a Linux es realitza mitjançant la ramificació del procés creador, amb el que s'estableix una relació de pare-fill, i l'assignació d'un número identificador (PID) i espai d'adreçament únic. El primer procés, anomenat *init* (PID=1), és creat durant la carrega del sistema.

Un cop creat el nou procés es pot realitzar la càrrega d'un nou programa al seu espai d'adreçament. A aquest procediment se'l denomina recobriment i si és executat a continuació de la ramificació evita la copia de recursos del procés pare (optimització *Copy-on-Write*).

3.4.2 Estats d'un procés

El descriptor de l'estat d'un procés en Linux pot prendre diversos valors dels que s'expliquen els més rellevants pel que fa a la gestió de processos:

- TASK_RUNNING: en espera preparat per a ser executat (4 o 6) o en execució (1 o 2).
- TASK_INTERRUMPIBLE: en repòs esperant una condició o senyal (3 o 5).
- TASK_UNINTERRUMPIBLE: en repòs esperant una condició (3 o 5).

A més, depenent de la estructura de dades on aquest estigui referenciat (*current*, cua/arbre en espera, llista de processos *zombi*, en memòria principal o secundaria) ens podem trobar amb els següents estats diferents:

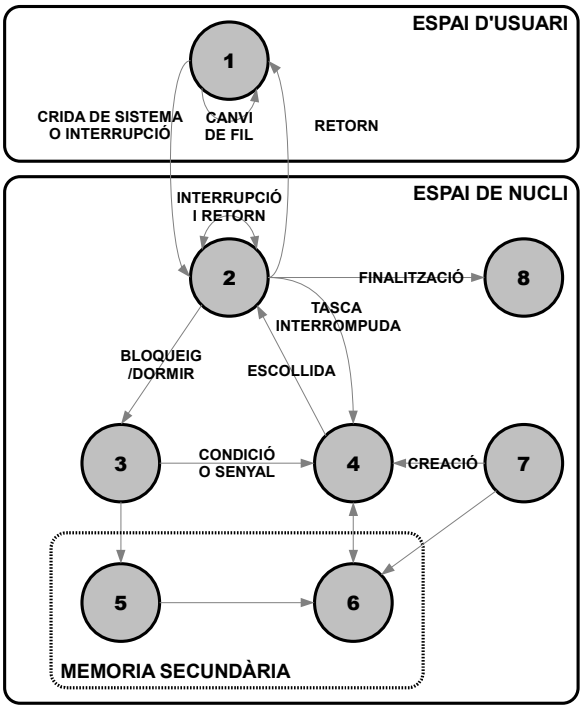
1. El procés s'esta executant en espai d'usuari fins que realitza una crida de sistema o ocórrer una interrupció, moment en el que entra al espai de nucli (TASK_RUNNING).
2. El nucli s'està executant en benefici del procés si aquest ha executat una crida de sistema o està atenent una interrupció (TASK_RUNNING).
3. El procés resta *dormint* en espera d'alguna senyal o condició donada (TASK_INTERRUMPIBLE o TASK_UNINTERRUMPIBLE segons la crida utilitzada).
4. El procés que estava dormint en rebre alguna senyal o una condició donada *desperta* o estava en execució i es interromput per una tasca de major prioritat o per criteri de planificació. En qualsevol cas s'incorpora a l'estructura de dades de tasques llestes per a ser executades (TASK_RUNNING).
5. El sistema reclama espai de memòria i salva el procés en memòria secundària (TASK_INTERRUMPIBLE o TASK_UNINTERRUMPIBLE).
6. Ídem estant la tasca a punt per executar-se (TASK_RUNNING).
7. El procés acaba de ser creat i s'afegeix a l'estructura de dades de tasques llestes en memòria principal o secundària segons l'espai disponible (TASK_RUNNING).
8. Al finalitzar, el procés és queda en estat zombi guardant, entre altres, un codi de sortida que podrà ser consultat per el pare que el va crear.

3.4.3 Finalització d'un procés

La finalització d'un procés es pot produir de manera voluntària, el programa executa una crida de sistema, o bé de manera involuntària, al rebre una senyal o al produir-se una excepció. En qualsevol cas, al concloure la seva execució, s'alliberen pràcticament tots els seus recursos a excepció de l'estructura d'informació del procés que es manté per proporcionar informació al procés pare.

A més, si el procés té fills aquests són reallotjats a un altre pare -fil viu del mateix procés o grup- o, en cas de no trobar-se, al procés inicial *init*. Arribat el moment aquest consultarà, si ho considera necessari, i finalment alliberarà l'estructura d'informació del procés finalitzat.

Figura 3: Diagrama d'estats d'un procés



3.5 Conceptes de la planificació

Linux disposa de dos tipus de polítiques de planificació: les enfocades a assolir una repartició justa i un màxim aprofitament dels processadors (*best-effort*), i les enfocades a obtenir una certa qualitat de servei pel que fa als temps d'atenció(*QoS/RT*). Les tasques de temps reals responen, doncs, als requeriments satisfets per aquest segon grup.

Tanmateix, i ja que un sistema de temps real esta composat per tasques de diferent necessitat pel que fa al compliment dels terminis, és beneficiós entendre ambdues polítiques alhora de dissenyar un sistema que satisfaci els requisits plantejats.

A continuació s'introdueixen, doncs, els conceptes fonamentals de la planificació. Entre aquests es dona la visió original -en Unix- de les prioritats dinàmiques, les llesques de temps i el valor *nice* tot i que, com es veurà més endavant, el seu significat ha canviat significativament en l'algorisme actual de planificació (CFS).

3.5.1 Tasca, procés i fil d'execució

Des del punt de vista del nucli, Linux implementa els processos i fils d'execució amb una única abstracció anomenada tasca. De fet, per aquest sistema operatiu la única diferència entre un fil i un procés es produeix en el moment de la seva creació en el qual s'estableix els recursos que seran compartits, en el que als fils d'un mateix procés se'ls assigna el mateix grup i en el que al procés o fil inicial creat d'un grup -anomenat líder- representa, en certa manera, el concepte de procés utilitzat en la literatura de sistemes operatius.

Tenint en compte aquesta consideració, i d'ara en endavant, s'utilitzarà l'abstracció tasca indistintament per a referir-se a un procés o a un fil d'execució exceptuant aquells casos en els que sigui necessària aquesta distinció.

3.5.2 Preempció

S'anomena preempció a la interrupció involuntària d'una tasca per donar pas a una altre. Es pot donar en l'espai d'usuari en el moment de retornar a aquest al finalitzar l'execució d'una crida de sistema o al retornar d'una interrupció. Els motius de la preempció en l'espai d'usuari poden ser dos: per garantir la repartició justa del processador en benefici d'un altre procés o per donar pas a un procés de major prioritat.

La preempció, a partir de la serie 2.6, es pot donar també a l'espai de nucli en els següents casos: al finalitzar la gestió d'una interrupció, al sortir d'una zona de no preempció, explícitament cridant a la rutina del planificador o al bloquejar el procés a l'espera d'una condició. La preempció a l'espai de nucli és necessària per garantir la ràpida reacció del sistema als esdeveniments associats a les tasques de temps real.

3.5.3 Canvi de context

El canvi de context és la operació utilitzada per guardar els recursos usats pel procés que abandona l'execució i restaurar els del procés que en reprèn la seva. Involucra principalment dos passos: intercanvi de l'espai d'adreçament virtual, i guardat i recuperació la informació de pila, registres així com qualsevol altre informació d'estat del processador rellevant al procés.

Degut al cost d'aquesta operació, entre altres com la pèrdua de les dades en la memòria cau d'un a altre procés, el canvi de tasca produeix un sobrecarrega al sistema que en fa davallar el seu rendiment fet que fa que les polítiques enfocades a assolir un major aprofitament del processador (*best-effort*) estiguin normalment en competició amb les orientades a una millor resposta (*QoS/RT*).

3.5.4 Prioritat fixa i prioritat dinàmica

En la decisió de quina tasca ha d'executar-se en un instant concret amb el propòsit d'assolir un objectiu de planificació determinat s'utilitza el concepte de prioritat, com a unitat d'ordenació en el temps. Així doncs, una tasca amb major prioritat s'executarà abans que una de menor prioritat.

Per assolir un repartiment del processador entre diferents processos el sistema utilitza prioritats dinàmiques que són periòdicament calculades en funció del temps que s'ha executat una tasca en relació a les tasques que estan en espera d'execució.

Per altra banda, per assolir respostes dins un termini determinat, és a dir planificar tasques de temps real, el sistema utilitza prioritats estàtiques. O sigui, s'assigna una prioritat superior a una tasca determinada per privilegiar-lo davant la resta, indistintament de si aquestes han gaudit o no de temps de procés.

3.5.5 Timeslice

La llesca de temps (*timeslice*), concepte utilitzat en sistemes Unix per a la compartició de CPU, defineix quan temps es pot executar una tasca fins que és interrompuda de manera involuntària pel procés de planificació. Per norma, tasques de prioritat més alta rebran llesques de temps majors.

L'elecció de la llesca, però, no es un tasca trivial. Una de massa curta produeix una davallada en el rendiment deguda a la sobrecarrega originada pels continus canvis de context. Una massa llarga, per contra, es tradueix en una resposta lenta del sistema davant els esdeveniments.

3.5.6 Valor *nice*

El valor *nice* (amabilitat) és un concepte utilitzat en els sistemes Unix que serveix en les polítiques de repartiment per privilegiar una tasca per sobre d'una altre. Quan més *amable* sigui una tasca més processador cedirà. Per contra, quan menys en sigui, menys en cedirà. En qualsevol cas, totes les tasques acabaran rebent alguna part del processador, tot i que algunes més que les altres.

Aquest valor s'utilitza, per tant, per ponderar el còmput de la prioritat dinàmica i, en conseqüència, el valor de la *timeslice* assignada. Si la tasca té un valor alt d'amabilitat se la penalitzarà en el comput de prioritat dinàmica. En el cas oposat, rebrà una bonificació.

3.5.7 Tasca I/O bound o processor bound

Cal distingir, també, entre dos tipus contraposats de tasca: les lligades a la entrada i/o sortida ((I/O boudn)) del sistema i les lligades al processador (processor bound). Les primeres es caracteritzen per consumir la major part del seu temps enviant i esperant resposta d'algun dispositiu de maquinari -una interfície d'usuari, per exemple- mentre les segones es distingeixen per dedicar el seu temps a realitzar càlculs -un programa de còmput matemàtic, per exemple.

Una política favorable als primers tendeix a executar amb molta freqüència cada una de les tasques però durant curts terminis de temps, donant una resposta ràpida als distints esdeveniments, mentre que una de favorable als segons tendeix a executar les tasques de manera poc freqüent però per períodes de temps llargs, disminuint el nombre de canvis de context i per tant maximitzant l'aprofitament dels recursos.

Evidentment, aquesta classificació no es mutualment exclusiva essent molt habitual de trobar aplicacions que exhibeixen ambdós tipus de comportament, de manera simultània o alternativament. Així doncs, les polítiques de planificació de repartiment han de satisfer dos objectius enfrontats: temps de resposta ràpids (baixa latència) per les tasques lligades a la E/S i màxim aprofitament dels recursos (alt rendiment) per les tasques lligades al processament.

3.6 Planificació amb repartiment

Anteriorment, Linux feia servir un algorisme de planificació seguint la tradició de Unix d'assignar una *timeslice* proporcional a la prioritat dinàmica del procés i bonificant les tasques lligades a la E/S. Més concretament, potenciava les tasques que bloquejaven abans d'esgotar la seva *timeslice*.

Tanmateix, en la versió 2.6.23, després de reconèixer la pobre interactivitat del sistema en els equips d'escriptori[9], es va canviar profundament l'aproximació al repartiment amb un nou planificador anomenat Completly Fair Scheduling (CFS)[MOL07] per a la classe *SCHED_OTHER*[10]. Aquest es basa en l'aproximació a un model de processador amb *multitasca perfecte* on cada procés rep exactament la seva part proporcional de temps. Aquest paral·lelisme teòric s'assoleix dividint

9 L'antic planificador, entre altres, mostrava una sobrecarrega excessiva en les continues commutacions entre tasques lligades al processador -penalitzades amb una *timeslice* petita-, una relació no lineal entre els increments del valor *nice* i els increments en la *timeslice* assignada, la falta de resolució de la *timeslice* múltiple del *tick* -senyal periòdica del sistema- i la debilitat del heurístic encarregat de mesurar la interactivitat d'una tasca davant dels atacs -algunes aplicacions es bloquejaven instants abans de finalitzar la seva *timeslice* per rebre la bonificació.

10 Nom de la classe de planificador utilitzat per a totes les tasques que no tenen requeriments de temps real. També existeixen les classes SCHED_BATCH i SCHED_IDDLE per treballs de tipus de processament de lots -és marcada com a lligada a la CPU i per tant penalitzada davant les interactives- i tasques de molt baixa prioritat que rebran la més baixa prioritat de tot el sistema.

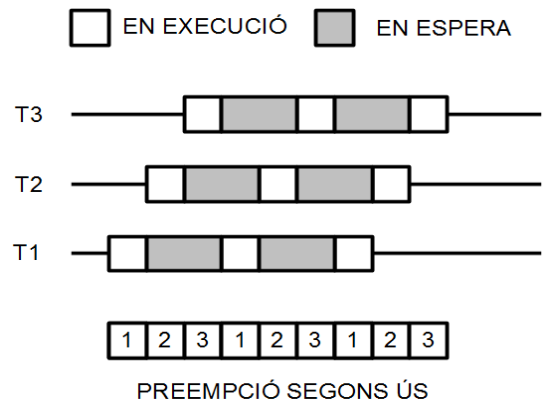
en porcions un temps de latència objectiu -temps màxim que tardarà una tasca en ser planificada- entre el total de tasques executables.

Per una banda, les tasques computen un temps virtual d'execució en nanosegons ponderat amb el valor *nice* -amb increments més grans per a valors d'amabilitat més alta. Per altra banda, es manté un arbre de cerca anomenat *red-black tree*[11] on s'ordenen les tasques de menor a major temps virtual executat. Un cop finalitzada la porció de temps s'escull la tasca amb menor temps virtual.

D'aquesta manera, d'un costat, es potencien automàticament les tasques interactives -mentre romanguin dormides el seu temps virtuals d'execució decreixera relativament i, per tant, seran ateses per davant de les altres- i, d'altre costat, les tasques lligades al processador no es veuen penalitzades a un valor de *timeslice* petit per si, si no a un temps proporcional a la quantitat de tasques executables.

A més, per reduir un possible excés de sobrecàrrega en canvis de context, es defineix un topall mínim de commutació de tasca que es pot ajustar depenent del sacrifici en rendiment que s'estigui disposat a assumir per tant d'assolir un temps de resposta més ràpid. Finalment, CFS suporta també la planificació basada en grups i usuaris per evitar que una aplicació pugui abusar del repartiment *just* de l'algorisme a base de crear múltiples fils d'execució.

Figura 4: Exemple de repartiment de temps en tasques amb el mateix valor *nice* i la mateixa interactivitat



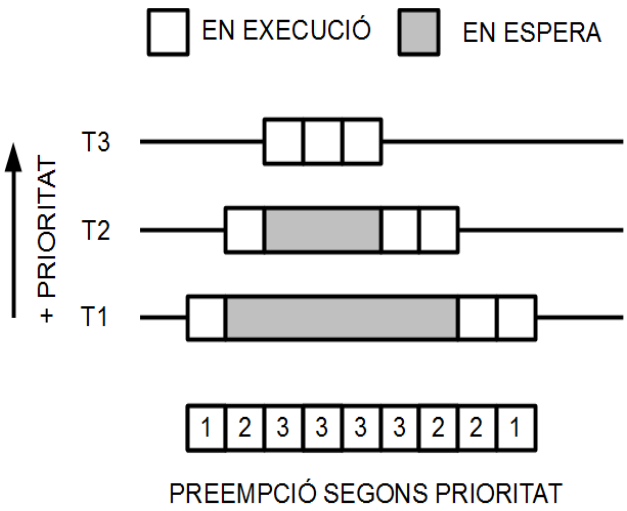
3.7 Planificació per prioritat estàtica

Linux ofereix dos classes de planificadors en temps real, *SCHED_FIFO* i *SCHED_RR*. Ambdós funcionen amb prioritats estàtiques de manera que en el moment en què es té coneixement d'una tasca executable de major prioritat s'interromp la tasca en curs per donar pas a aquesta.

El primer, *SCHED_FIFO*, serveix les tasques en l'ordre d'arribada (*First Input First Output*) i no les interromp fins que finalitzen o una tasca de major prioritat esdevé executable. En el segon, *SCHED_RR*, les tasques executables d'una mateixa prioritat es van alternant segons una *timeslice* fixa (*Round Robin*) i no s'interrompen fins que finalitzen o una tasca de major prioritat esdevé executable.

11 Un tipus d'arbre binari balancejat de cerca. És a dir, un graf acíclic, connectat i dirigit on s'ordenen els nodes de manera que cada node pot apuntar fins a dos nodes més, el de l'esquerra per sota del seu valor i el de la dreta per sobra. Consegüentment, la cerca d'un node determinat te un cost logarítmic O(log N).

Figura 5: Exemple de preempció per prioritat estàtica



En ambdós casos s'utilitza una cua per prioritat i processador augmentant les possibilitats de que la memòria cau estigui calenta[12] i garantint que la latència de planificació esdevé constant, és a dir, d'ordre O(1). Tanmateix, en un entorn multiprocessador, pot ser necessari que el planificador reallotgi alguna tasca per assegurar el balanceig de càrrega entre processadors.

3.7.1 Assignació de prioritats estàtiques

La assignació de prioritat estàtiques requereix d'un anàlisi previ de les característiques temporals de les tasques a planificar que permeti ordenar-les segons la freqüència i/o el termini d'execució de manera que es compleixin uns objectius de temps real donats. A continuació, i com a material complementari, s'exposa una breu introducció a la planificació basada en l'assignació de prioritats a les tasques més freqüents i, també, a la basada en a les més urgents.

Considerant un sistema de temps real composat per un nombre fixat de tasques periòdiques independents executades en un sistema monoprocesador amb preempció per prioritat fixa, es defineixen els següents termes per a cada tasca:

- Condicions conegudes / desitjades
  - Ti: Període d'execució.
  - Fi: Freqüència d'execució = 1/Ti.
  - Ci: Pitjor temps d'execució (WCET) + pitjor latència (inclou canvi de context).
  - Di: Termini màxim de resposta.
  - Ui: Factor d'utilització = *Ci/Ti* .
- Prioritat a assignar
  - Pi: Prioritat (quan major Pi, major prioritat).
- Condició a verificar (Ri ≤ Ti):
  - Ri: Temps de resposta en el pitjor dels casos.

12 És diu que una memòria cau està calenta quan conté dades del procés a executar-se.

3.7.2 *Rate Monotonic Scheduling* (RMS)

La planificació per prioritat al més freqüent és optima per a tasques periòdiques amb terminis iguals al seu període *Di = Ti*. Segons aquest criteri les prioritats han d'assignar-se en ordre directament proporcional a la freqüència *Fi = 1/Ti*, on *Fi >Fj* ⟶ *Pi > Pj* i és optima en el sentit que:

> "Si assignant prioritats amb un altre esquema es garanteix el plaç, amb RMS també es garanteixen . Si amb RMS no es poden garantir, amb qualsevol altre esquema tampoc" (Sha & Lehoczky 1986 [SHA86])

La prova del factor d'utilització total del sistema (Liu & Layland [LIU73]) *Ui = Ci/Ti* verifica que el sistema es pot planificar -és a dir, es compleixen tots els terminis- si es compleix la següent inequació:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} < N\left(2^{1/N} - 1\right)$$

Aquesta prova es suficient però no indispensable. En cas de que no es compleixi es pot recorrer al càlcul del pitjor temps de resposta (Ri) de cada tasca segons (Lehoczky et al. [LEH89]):

$$R_i = C_i + \sum_{j \in pM(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

on:

- *pM(i):* totes les tasques amb prioritat major a la calculada
- *⌈ ⌉*: operador *arrodoniment a l'alça*[13]

En definitiva el que fa la funció és calcular el pitjor temps de resposta sumant al pitjor temps d'execució i pitjor latència *Ci* de la tasca tots els temps que pot ser interrompuda per les tasques de prioritat superior. Com que *Ri* no es pot aïllar de la funció al estar dins de la funció d'arrodoniment[14] s'utilitza el següent càlcul recursiu:

$$w_i^{n+1} = C_i + \sum_{j \in pM(i)} \left\lceil \frac{w_i^n}{T_j} \right\rceil C_j$$

amb valor inicial:

$$w_i^0 = C_i + \sum_{j \in pM(i)} C_j$$

El càlcul finalitza quan $w_i^{n+1} = w_i^n = R_i$, és compleix el termini *Ri ≤ Di*, o bé quan $w_i^{n+1} > D_i$, no es compleix el termini i per tant no es pot planificar.

3.7.3 *Deadline Monotonic Scheduling* (DMS)

El criteri d'assignació de prioritats RMS deixa de ser òptim si alguna de les tasques compleix que *Di < Ti*, cas en el que s'aplica el criteri de prioritat al més urgent (DMS) que assigna la major prioritat a la tasca amb el menor termini de resposta *Di < Dj* ⟶ *Pi > Pj* òptim en aquesta nova situació. Per saber si el sistema es pot planificar, s'aplica també el càlcul de la pitjor resposta *RI*.

---

13 També anomenat sostre, de l'anglès *ceiling*.

14 Les equacions de la forma *x = funció(x)* no són resolubles per aïllament ja que al aplicar la funció inversa a banda i a banda de la equació torna a quedar una expressió de la forma *funció$^{-1}$(x) = x*.

3.8 Interrupcions

Les interrupcions són senyals elèctriques generades pel maquinari, recollides habitualment mitjançant un controlador especialitzat i utilitzades per senyalitzar al processador d'un esdeveniment succeït. Provoquen la preempció del codi en execució codi -sempre que no estiguin emmascarades- estalviant d'aquesta manera el sondeig periòdic d'un succés determina, per exemple, la finalització d'una operació prèviament requerida.

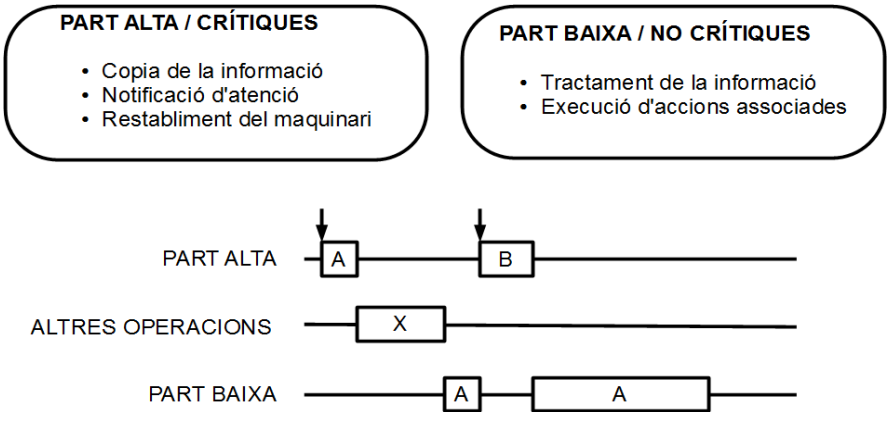
El sistema d'interrupcions està format per diferents línies d'interrupció (IRQ) associades a diferents dispositius amb el propòsit de senyalitzar esdeveniments diversos. Algunes d'aquestes línies d'interrupció estan associades de manera estàtica mentre d'altres s'associen de manera dinàmica. De la mateixa manera mentre algunes línies d'interrupció són exclusives d'un esdeveniment determinat n'hi han que estan compartides per varis.

3.8.1 Gestió de les interrupcions

Cada línia d'interrupció està associada mitjançant un vector de salt -una adreça de programa de l'espai del nucli- al que s'anomena gestor d'interrupció o rutina d'atenció a la interrupció (ISR). Per evitar la degradació del sistema, és de vital importància que el gestor d'interrupció s'executi amb celeritat, retornant el flux d'execució al codi interromput el més aviat possible. Tanmateix, existeixen certs esdeveniments que requereixen d'una quantitat de treball considerable a realitzar.

Amb l'objectiu de satisfer aquests dos objectius contraposats, finalitzar amb celeritat i alhora realitzar el treball requerit davant d'un succés determinat, Linux divideix la gestió de les interrupcions en dos parts ben diferenciades: la meitat alta i la meitat baixa. La primera correspon al propi gestor d'interrupció i realitza les operacions crítiques en el temps com poden ser la copia d'informació, la notificació d'atenció i el restabliment del maquinari. La segona conté les operacions no crítiques en el temps com poden ser el tractament de la informació rebuda i l'execució de les accions associades. Mentre la primera s'executa en el mateix moment en què es rep la interrupció, la segona es ajornada fins que el sistema operatiu ho consideri.

Figura 6: Divisió de les operacions associades a una interrupció



3.8.2 El context d'interrupció

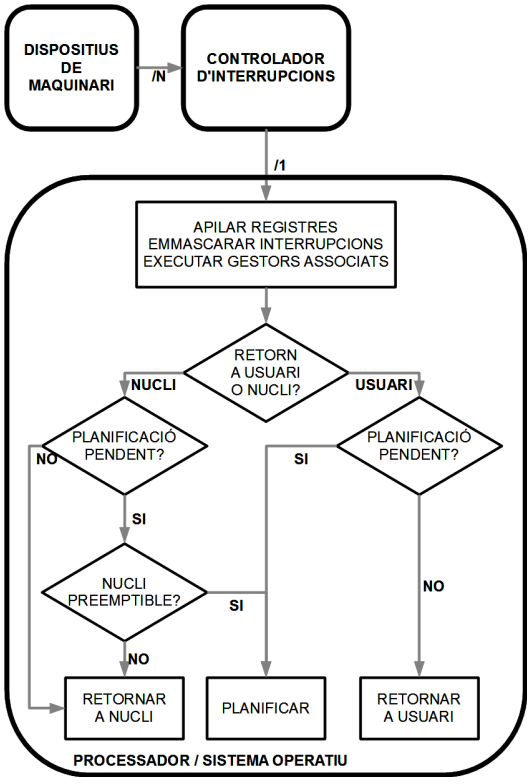
El gestor d'interrupcions s'executa en un context especial anomenat d'interrupció -vist amb anterioritat a l'apartat 3.2. Aquest context té certes característiques especials com el fet de què, en principi, està desproveït de context de procés, pel que no es pot planificar i per tant tampoc es pot ficar a dormir en mig de la seva execució; utilitza el seu propi espai de pila, anomenat *pila d'interrupció,* i, per norma, no és reentrant ja que a l'inici de la seva execució emmascara[15] les interrupcions abans de manera que el mateix gestor no pot ser invocat un segon cop de manera imbricada.

3.8.3 Execució de la part alta

En produir-se una interrupció el flux d'execució es interromput, s'apilen el registres del processador i s'inicia l'execució del codi apuntat pel vector de salt. En aquest punt, el gestor d'interrupció genèric notifica la recepció de la interrupció i emmascara les interrupcions, verifica si hi ha un gestor específic registrat per la línia d'interrupció donada (IRQ) i cedeix el control a una rutina de gestió dels esdeveniments que desemmascara la resta de línies d'interrupció i executa cadascun dels gestor específics pendents de servei associats a la línia .

Finalment, un cop servides totes les interrupció pendents, es verifica si el codi ha de retornar al espai d'usuari, és a dir que s'havia interromput aquest, i si existeix algun procés pendent de planificació. Si és aquest el cas es crida a la rutina de planificació perquè realitzi el canvi de context i cedeixi l'execució a un nou procés. En cas de que el retorn sigui al nucli es verifica a més que aquest estigui en un punt en el que es pugui interrompre.

Figura 7: Flux simplificat de la gestió d'interrupcions



15 S'anomena emmascarar, en referència a una operació de màscara de bit, al fet d'inhibir una línia d'interrupció donada.

3.8.4 Mecanismes per implementar la part baixa

Linux, en resposta a diferents necessitats, posa a la disposició fins a tres mecanismes distints per a la implementació de la part baixa, de més a menys crítics: *softirqs*, *tasklets* i les cues de treball. El primer, *softirqs*, es tracta d'un conjunt de fins a 32 part baixes -en l'actualitat només s'utilitzen 9- de prioritat fixada i definides de manera estàtica en la compilació del nucli. Poden executar-se de manera simultània en qualsevol processador, fins i tot dos del mateix tipus. S'executen, normalment, al propi retorn d'interrupció, al finalitzar l'execució de la part alta, i són utilitzades única i exclusivament en processos crítics de temps o amb una freqüència d'aparició molt alta.

Les segons, *tasklets*, estan dividides alhora en dos prioritats (alta i baixa), és poden crear dinàmicament i s'executen cada una de les dues prioritats en una *softirq*. Diferents *tasklets* poden executar-se de manera concurrent en processadors diferents però mai dos del mateix tipus. Al proporcionar protecció contra la reentrada són més segures i fàcils de programar que les *softirq*. Proporcionen un bon compromís entre temps de resposta i facilitat d'ús.

Finalment, les cues de treball s'executen en un fil del nucli en un context de procés. Com a tals es poden planificar i per tant, de les tres, són les úniques que es poden ficar a dormir. S'utilitzen en la gestió de parts baixes que requereix d'executar operacions de duració considerable que es veuen beneficiades pel fet de poder-se bloquejar, dormir i tornar a ser planificades un cop l'operació a finalitzat.

3.8.5 El dimoni *ksoftirqd*

En situacions on existeix una carrega molt alta de interrupcions és pot donar el cas que el ritme peticions superi al d'execució d'una *softirq* o, en aquest cas el mateix, un *tasklet*. Aquesta situació, juntament amb el fet que hi han *softirqs* que es rellancen a si mateix, pot provocar l'exhauriment del temps del processador deixant els processos d'usuari sense atenció.

Per evitar la situació esmentada, si una *softirq* es reassignada abans de que pugui ser servida s'activa el dimoni[16] anomenat *ksoftirqd* que executarà la resta de peticions en context de procés i amb una prioritat molt baixa (valor *nice* màxim). D'aquesta manera es garanteix que aquestes peticions seran ateses en algun o altre moment sense exhaurir els recursos de les altres tasques.

3.9 Rellotge del sistema

El rellotge del sistema és un dispositiu de maquinari programable que senyalitza una interrupció periòdica a una freqüència prèviament establerta, el *tick*[17] del sistema. El gestor d'interrupció associat a aquest dispositiu, anomenat comunament com interrupció de rellotge, realitza una sèrie de tasques periòdiques vitals per al funcionament del sistema com són: l'actualització de la data del sistema, el balanceig de càrrega en sistemes multiprocessador (SMP), la planificació d'una nova tasca si l'anterior ha esgotat la seva *timeslice*, l'execució dels temporitzadors exhaurits i l'actualització de les estadístiques d'ús dels recursos del sistema.

3.9.1 Ritme del *tick*

La freqüència del sistema, anomenada també com ritme del *tick*, es programada al arrancar el sistema, amb un valor per defecte de 100Hz -un *tick* cada 10ms- per a arquitectures x86. Aquest valor, però, pot esser incrementat en la compilació del nucli provocant que les tasques del rellotge del sistema s'executin de manera més freqüent el que es tradueix en una major resolució i precisió

16 En sistemes Unix s'anomena dimoni (*daemon*) a un procés auxiliar del sistema operatiu que s'executa en segon pla.
17 En referència al soroll produït pel moviment d'una agulla de rellotge mecànic.

dels esdeveniments temporitzats, com poden ser: la planificació per repartiment de temps, els temporitzadors del sistema i la mesura del pas del temps i la utilització dels recursos.

Tanmateix, l'increment del ritme del *tick* implica una major sobrecarrega del sistema ja que el processador és interromput de manera més freqüent per la interrupció de rellotge deixant menys temps per la resta de processos, siguin de nucli o d'usuari, i augmentant la possibilitat de provocar majors fallades de memòria cau.

3.9.2 Temporitzadors

El temporitzadors són un recurs del sistema operatiu per poder diferir l'execució de certes tasques a un moment determinat del futur. Un cop iniciats, la interrupció de rellotge redueix a cada *tick* el seu valor fins arribar a cero, moment en el que s'executarà el seu gestor associat. Aquestos seran executats en la part baixa d'una interrupció, en una *softirq* més concretament.

3.9.3 Sistema sense rellotge

A partir de la versió 2.6.21 del nucli Linux permet esser configurat com un sistema sense *tick* del sistema. En comptes de mantenir un ritme constant el rellotge del sistema es programat amb una temporització diferent cada cop corresponent a l'esdeveniment temporal planificat més proper. D'aquesta manera s'evita la sobrecarrega innecessària del processador, característica especialment útil per a l'estalvi d'energia.

Tanmateix, tot i que la reducció en interrupcions ofereix un millor rendiment global, introdueix una major variància pel que fa a la latència causada pel mateix ja que a cada interrupció és necessari calcular el proper termini i tornar a programar el controlador.

3.10 Sincronització

Linux, com a sistema operatiu multitasca, suporta la concurrència, bé sigui a través de paral·lelisme, mitjançant el suport a arquitectures multiprocessador simètriques (SMP), bé sigui través de compartició de temps, mitjançant la preempció de les tasques en execució. Aquest fet implica que diferents fluxos de codi puguin accedir i manipular la mateixa informació donant la possibilitat de produir el que s'anomena *condicions de carrera*, situacions on l'ordre i intercalació de diferents camins en l'accés i/o manipulació d'una estructura de dades produeix resultats diferents.

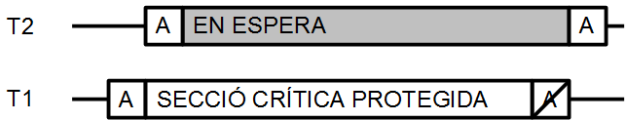
Per evitar les condicions de carrera es necessari coordinar l'execució concurrent dels blocs de codi, denominats *seccions crítiques*, que realitzen l'accés a l'estructura de dades compartida. A aquesta acció se l'anomena *sincronització* i s'implementa de diferents maneres depenent del mode d'accés -llegir i/o escriure, accés per part de dos o més tasques...- i del grau de protecció requerit.

En aquest apartat es s'introdueixen els *locks* utilitzats en el nucli de Linux . L'espai d'usuari disposa de mecanismes similars que, finalment, estàn basats en els descrits a continuació .

3.10.1 Locks

Els panys (*locks*) són el principal mecanisme utilitzat en la sincronització dels diferents camins de codi que volen accedir a una secció crítica. Quan un camí d'execució vol entrar en una secció primer ha d'obtenir -*tancar*- el *lock*. Un cop adquirit la secció de codi protegida no podrà ser accedida per altres fluxos fins que aquest *lock* sigui alliberat -*obert*.

Figura 8:Ús de *locks* per protegir l'accés a una secció crítica



3.10.2 Origen de les condicions de carrera

En l'espai d'usuari, en un sistema de compartició de temps, el fet que una tasca puguin ser interrompuda de manera involuntària enmig de l'execució d'una secció crítica pot provocar condicions de carrera. Paral·lelament, aquest mateix fet és pot produir en sistemes multiprocessador. Alguns exemples de seccions crítiques són l'accés als espai de memòria compartits i els fitxers.

De manera semblant, en el nucli existeixen causes similars de concurrència que poden provocar condicions de carrera: les interrupcions, les *softirq* i els *tasklets*, la preempció del nucli, el bloqueig d'un fil de nucli i la seva sincronització amb l'espai d'usuari i l'accés paral·lel dels sistemes multiprocessador.

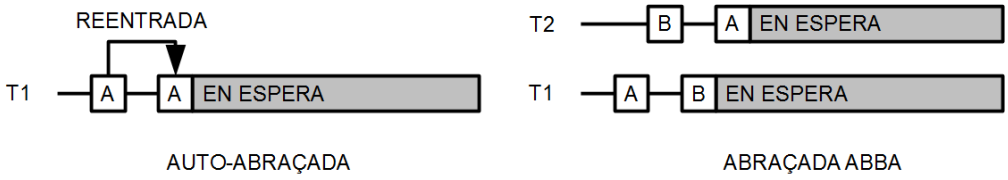
Els errors provocats per condicions de carrera són especialment difícils de trobar ja que només es donen sota una ordenació temporal dels esdeveniments determinada difícil de reproduir o detectar. És per això que es necessari una correcta aplicació dels mecanismes de sincronització per evitar que aquests es produeixin. Sota aquesta perspectiva s'anomena *interrupt-safe*, *SMP-safe* i *preempt-safe* al codi que es segur d'executar a pesar de la seva possible preempció per part de les interrupcions, altres processadors o el propi processador respectivament.

3.10.3 Abraçada mortal

L'abraçada mortal és un possible efecte indesitjat de la sincronització en la que un o més fils d'execució esperen mútuament l'accés a un o més recursos que no poden ser alliberats mentre l'accés als altres no es produeixi. Aquesta situació deixa el sistema en un estat en el que no pot progressar. Existeixen bàsicament dos tipus d'abraçada mortal:

- Auto-abraçada mortal: quan un mateix fil intenta obtenir un *lock* que ell mateix ha obtingut prèviament, per exemple, per efectes de la recursivitat.
- Abraçada mortal mútua: quan diversos fils retenen un *lock* d'algun dels altres i cap d'ells pot progressar esperant la obtenció del *lock* retingut per algun d'aquestos. L'exemple més comú, anomenat abraçada ABBA, es dona quan dos fils d'execució intenten obtenir els dos mateixos *lock* en ordre invers.

Figura 9: Exemples d'abraçades mortals comuns



Tanmateix existeixen algunes regles de disseny bàsiques que es poden seguir per tal d'evitar les abraçades mortals:

- Adquirir els *locks* sempre en el mateix ordre.
- Garantir que les seccions crítiques sempre finalitzen.
- No adquirir dos cops el mateix *lock*.
- Dissenyar esquemes d'obtenció i alliberació de *locks* simples.

3.10.4 Mètodes de sincronització

Linux disposa d'una sèrie de mètodes de sincronització utilitzats tant per la protecció de les seccions crítiques del nucli com per, mitjançant les crides a sistema, la construcció de les rutines disponibles a l'espai d'usuari amb la mateixa finalitat.

3.10.4.1 *Operacions atòmiques*

Les operacions atòmiques són la base per a la creació dels mètodes de sincronització. S'executen de manera indivisible i, per tant, estan implementades de manera totalment dependent a l'arquitectura, bé sigui mitjançant instruccions en assemblador, bé sigui emmascarant les interrupcions. Es disposa de dos tipus d'operacions atòmiques, de tipus sencer i de màscara de bit.

3.10.4.2 *Spinlocks*

Els *spinlocks* són panys que poden ser adquirits per un fil d'execució d'un processador únicament. Qualsevol altre fil, d'un altre processador, que pretengui disputar-lo roman donant voltes (*spinning*) a un bucle que comprova contínuament el seu estat. És a dir, els *spinlocks* proporcionen sincronització a nivell de multiprocessador i, per tant, permeten crear codi *SMP-safe*.

Tanmateix, degut al malbaratament de temps de procés que suposa aquest mecanisme és recomana utilitzar-lo únicament quan la secció crítica a protegir o bé resideix en un context d'interrupció -on no pot ficar-se a dormir- o bé el seu temps d'execució està per sota del necessari per realitzar dos canvis de context -per ficar-se a dormir i per tornar-se a despertar.

Existeixen diferents crides d'obtenció i alliberació d'aquest pany que permeten emmascarar les parts altes i les parts baixes de les interrupcions per evitar condicions de carrera entre: gestors d'interrupcions, parts baixes i gestors d'interrupcions, i context de procés i parts baixes. De la mateixa manera, es troben variacions d'aquest mateix pany per a usos de lectors-escriptor on es relaxen les regles de protecció de manera que varis lectors poden accedir a una mateixa secció crítica però tan sols un escriptor, i cap lector, pot accedir a la mateixa.

Ja que proporcionen sincronització entre processador diferents no són útils en sistemes uniprocessador tot i que són utilitzats per la inhibició de la preempció del nucli ja que determinen l'inici i la finalització d'una secció crítica. En cas de no estar configurada la preempció del nucli els *spinlocks* són completament suprimits en la compilació quedant únicament la part d'emmascarament d'interrupcions o part baixes en cas d'utilitzar les crides que si n'incorporen.

3.10.4.3 *Semàfors*

Mecanismes de pany que, en cas de trobar-se amb el pany ja adquirit, dormen la tasca pretendent en un cua a la espera de l'alliberament del mateix. Degut a aquesta propietat, i tenint en compte que suposen un sobrecost de gestió en comparació amb els *spinlocks*, són utilitzats en seccions crítiques que tarden un temps considerable en executar-se i que pertanyen a context de procés, ja que el context d'interrupció no es pot planificar. A més, al contrari que els *spin-locks*, permet l'adquisició per part de més d'un propietari a la vegada.

Els semàfors tenen associat un comptador inicialitzat al nombre màxim de propietaris que poden tenir que es manipula mitjançant dues operacions: *down* i *up*. La primera, utilitzada per obtenir el pany, decrementa el nombre del comptador, si al fer-ho aquest té un valor positiu el pany s'ha obtingut. En cas contrari, las tasca que realitza la petició s'afegeix a una cua d'espera. La segona, utilitzada per alliberar el pany, incrementa el nombre del comptador i en cas d'esdevenir negatiu desperta la següent tasca de la cua d'espera. De la mateixa manera que en el cas dels *spin-locks* es troben variacions d'aquest mateix pany per a usos de lectors-escriptor

3.10.4.4 *Mutexs*

Els *mutex*[18] o zones d'exclusió mútua, per la seva banda, són semàfors binaris, d'un únic propietari, que compleixen una sèrie de restriccions de manera que són més simples d'utilitzar i més eficients: només una tasca pot adquirir el pany; aquest ha de ser alliberat des del mateix context i espai des d'on es va obtenir, pel que no es pot utilitzar en esquemes complexos de sincronització entre el nucli i l'espai d'usuari; un procés no pot finalitzar mentre n'és propietari d'un, i, finalment, no pot ser adquirit per una interrupció o *softirq*.

3.10.4.5 *The big kernel lock*

Es tracta d'un pany de tipus *spin_lock* de caràcter global -afecta a tot el nucli- utilitzat en la transició a sistemes multiprocessador (SMP). Aquest té unes característiques peculiars: el pany es alliberat en dormir, és recursiu i s'utilitza únicament en context de procés. A dia d'avui el seu ús està vetat tot i que per les seves propietats encara roman en algunes part del codi del nucli.

3.10.4.6 *Locks de seqüència*

Els *locks* de seqüència són un mecanisme lleuger de sincronització per a pocs escriptors i molts lectors. Utilitza un comptador que en els escriptors s'incrementa tant a l'adquisició com a l'alliberament. En els lectors per la seva banda aquest mateix comptador es llegeix abans i després de la lectura donant el mateix nombre si no hi hagut una escriptura pel mig i nombres diferents si n'hi hagut. A més, degut a la natura seqüencial del pany els valors parells significaran que no hi ha una escriptura en curs. Aquest tipus de pany prioritza els escriptors sobre els lectors. Un exemple de la seva utilització és en el comput del temps del sistema -a través de la unitat anomenada *jiffie*[19]- realitzat en la interrupció de rellotge.

3.10.4.7 *La inhibició de la preempció*

Tot i que s'utilitzen els *spin-locks* per a inhibir i habilitar la preempció del nucli, mitjançant el comptador de preempció que opera de manera similar a un semàfor, existeixen seccions crítiques de codi que utilitzen dades lligades a la CPU on s'executa la tasca i que, per tant, no estan escortades per aquest tipus de *locks*. Per poder protegir aquestes seccions existeix també la possibilitat de flanquejar-les amb instruccions específiques de inhibició i habilitació de la preempció.

3.10.4.8 *Variables de finalització*

Les variables de finalització són una manera senzilla de senyalitzar la fi d'una secció o operació a una altra tasca. Les variables definides com a tal permeten tres tipus d'operacions: inicialització, espera i senyalització de la finalització.

18 Contracció del terme en anglès *mutual exclusion*.
19 Terme col·loquial per descriure "un instant curt". En ciencies de la computació defineix el temps entre dos cicles de rellotge.

3.10.4.9 *Barreres de memòria*

Les barreres de memòria són un tipus de pany que col·locat en un punt donat del programa garanteix la ordenació, a nivell de compilació i de processador, en la execució d'una seqüència d'instruccions. És a dir, les instruccions col·locades abans de la barrera de memòria hauran de ser executades previ ament a les col·locades després de la mateixa.

3.10.5 Mecanismes RCU

El mecanisme RCU (Read Copy Update) permet la compartició de dades sense bloqueig (*lock*) dels lectors proveint un mètode de sincronització més lleuger, pel que fa a les lectures, dels vistos fins ara. El seu principi de funcionament es basa en la utilització de copies per part dels processos escriptors en la modificació de la estructura de dades sincronitzada i en la posterior actualització de les referències a aquesta estructura.

Per salvar les possibles condicions de carrera produïdes en la actualització de la estructura de dades, aquest procés es divideix en dos etapes ben diferenciades: substitució de la referència a la antiga copia per una referència a la nova i retard de la eliminació de la antiga copia fins que cap altre tasca lectora en faci referència. Per garantir aquesta última situació cal que, per una banda, les tasques lectores realitzin l'accés de manera atòmica -no podran ser interrompudes- i, per altre banda, en sistemes multiprocessador (SMP), que la tasca escriptora esperi a que tots els processadors hagin executat el planificador com mínim un cop. Aquesta última condició garanteix que les seccions crítiques, i per tant l'accés a l'estructura de dades compartida per part de les tasques lectores, han estat finalitzades.

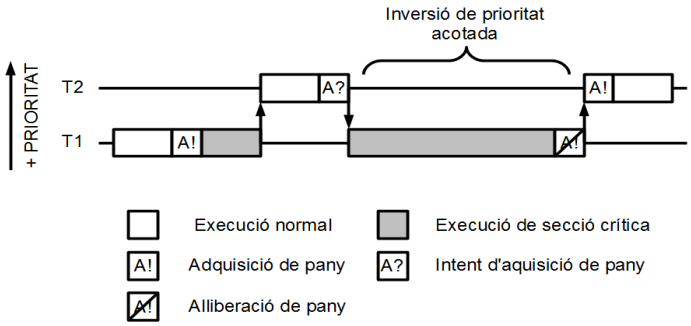
Per contrapartida, com es pot observar, l'ús d'aquest tipus de mecanisme és més costós pel que fa a l'actualització de l'estructura de dades competida. En aquest aspecte la tasca escriptora cal que realitzi les següents operacions: copia de l'antiga estructura de dades a una de nova, modificació de la nova copia, actualització de la referència a la nova estructura, espera (bloqueig) fins la finalització de les tasques de la resta de processadors, alliberació de la memòria de l'antiga estructura de dades.

3.11 Inversió de prioritats

La inversió de prioritats es un fenomen que ocorre quan una tasca de major prioritat que vol accedir a un recurs compartit es veu obligada a cedir la CPU a una de menor prioritat que té aquest mateix recurs reservat, en espera de la seva alliberació. Existeixen dos tipus de inversió de prioritats: acotada i desacotada.

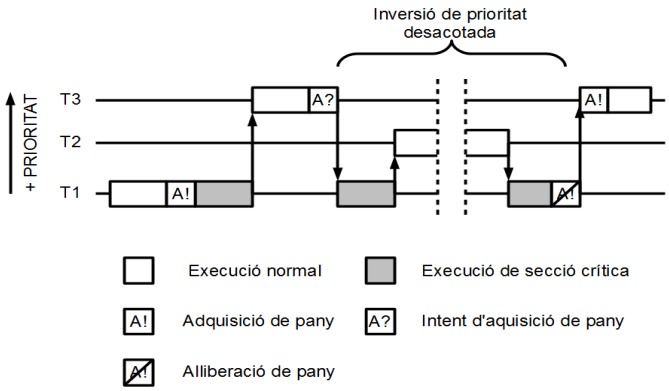
La primera, acotada, és el cas més simple i esdevé quan una tasca de major prioritat intenta accedir a un recurs compartit l'accés al qual està reservat per una tasca de menor prioritat. En aquest cas, fins que la tasca de menor prioritat no allibera la de major prioritat no pot continuar la seva execució. El temps durant el qual es produeix la inversió de prioritat, per tant, està acotat a la longitud de la secció crítica de la tasca de baixa prioritat.

Figura 10: Inversió de prioritat acotada



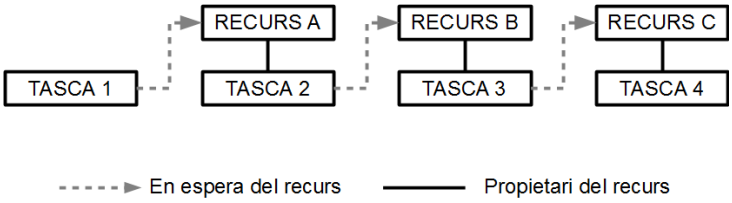
La segona, desacotada, és produeix quan en una inversió acotada la tasca de baixa prioritat és veu interrompuda per una de mitja prioritat que no permet finalitzar la primera i que, per tant, allarga el temps de resum de la tasca d'alta prioritat. En aquest cas, l'extensió incontrolada de la durada de la secció critica produïda per la tasca de mitjana prioritat provoca una inversió desacotada. El temps durant el qual es produeix la inversió de prioritats esdevé, en conseqüència, la suma de la secció crítica de la tasca de baixa prioritat més el pitjor temps d'execució de la tasca de mitjana prioritat. Si per qualsevol motiu la tasca de mitjana prioritat s'executés indefinidament la tasca d'alta prioritat no tindria oportunitat de fer-ho.

Figura 11: Inversió de prioritat desacotada



La inversió de prioritat, a més, pot tenir efectes encara més notables quan es produeix una *cadena de reservació de recursos.* És a dir, quan diverses tasques romanen bloquejades de manera successiva una per l'intent d'accés als recursos compartits i reservats de l'altre. En aquest cas la inversió, si es acotada, esdevé la suma de les seccions crítiques de les tasques involucrades en la cadena de reservació. Si és desacotada, més encara, pot allargar-se fins la suma dels pitjors temps d'execució de la cadena de tasques. Una cadena de reservació de recursos incrementa, a més a més, la probabilitat de que es produeixin inversions de prioritat desacotades.

Figura 12: Cadena de reservació de recursos



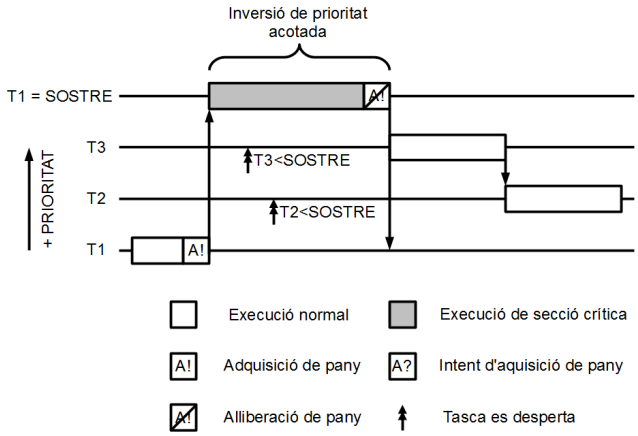
Per mirar de minimitzar els efectes de la inversió de prioritats s'apliquen principalment algun d'aquests dos protocols: el sostre de prioritats o la herència de prioritats; o alguna variant dels mateixos. En qualsevol cas, cap d'aquestes dues tècniques és infal·lible i, en última instància, només el correcte disseny d'una aplicació pot garantir l'assoliment dels terminis.

3.11.1 Sostre de prioritats

Aquest protocol assigna a cada recurs compartit un sostre de prioritat per sobre de la prioritat més alta de les tasques que poden accedir al recurs. Al adquirir un recurs compartit la tasca que realitza l'adquisició veu la seva prioritat elevada fins al sostre del recurs. Evitant la contenció del recurs, ja que la tasca que l'adquireix no podrà ser interrompuda fins que finalitzi l'accés, s'aconsegueix acotar el pitjor temps d'inversió de prioritat a la longitud de la secció crítica més llarga de totes les tasques de menor prioritat que accedeixen al recurs.

L'aplicació d'aquest protocol, però, requereix d'un complex i, a vegades, inviable anàlisi estàtic de l'aplicació per establir el sostre de prioritats de cadascun dels recursos compartits fet pel que no es aplicable com a mecanisme autònom ofert per un sistema operatiu. A més, presenta un pobre rendiment provocat per la sobrecarrega que suposa fer créixer i decréixer la prioritat d'una tasca cada cop que adquireix o allibera l'accés a un recurs compartit. Finalment, com que la política és la de augmentar la prioritat alhora d'adquirir el recurs, el protocol arriba a evitar innecessàriament l'execució de tasques de prioritat mitjana, entre la tasca elevada i el sostre de prioritats del recurs.

Figura 13: Protocol de sostre de prioritats



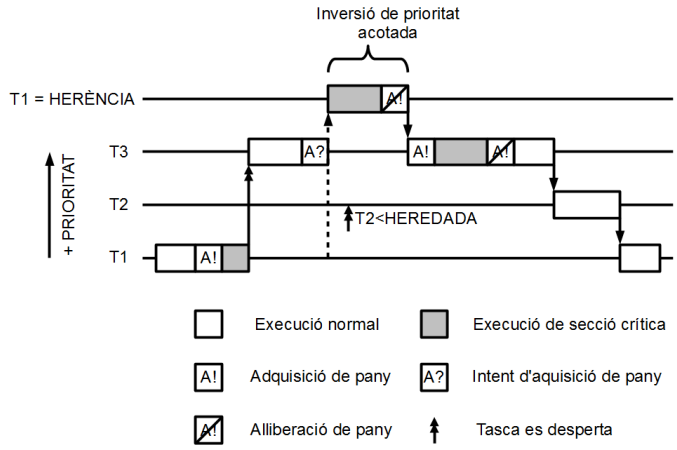
3.11.2 Herència de prioritats

El protocol d'herència de prioritats, al contrari que el de sostre de prioritats, utilitza un mecanisme d'ajust dinàmic pel que pot ser implementat per un sistema operatiu. El funcionament és basa en

l'elevació de la prioritat d'una tasca de menor prioritat quan essent propietària d'un recurs compartit aquest és reclamat per una de major prioritat. D'aquesta manera s'evita que una tasca de prioritat mitjana pugi arribar a interrompre a una de baixa estenen la seva secció crítica per un període indefinit i bloquejant l'execució d'una tasca de major prioritat.

Com que estadísticament la majoria de recursos compartits en una aplicació de temps real no s'arriben a competir en realitat aquest protocol ofereix un bon rendiment mitjà, evitant l'elevació i retorn de la prioritat que el protocol de sostre imposa sota qualsevol circumstància. Per altra banda manifesta un pitjor cas degut a la possibilitat d'imbricació produïda per una cadena de reservació de recursos. En aquest cas el pitjor temps és la suma de totes les seccions crítiques imbricades més la gestió que implica la elevació i degradació de les prioritats cada cop que el o els recursos són adquirits o alliberats.

Figura 14: Protocol d'herència de prioritats



3.12 La jerarquia de memòries

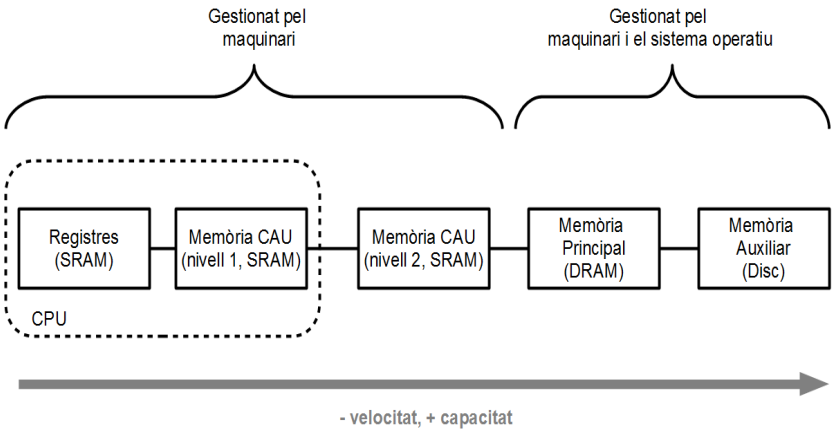
A causa de la relació entre la velocitat, capacitat i cost dels dispositius de memòria -a major velocitat menor capacitat i major cost- les arquitectures de computadors introdueixen el concepte de jerarquia de memòries. Aquest consisteix en estructurar la memòria de manera jeràrquica col·locant els dispositius més ràpids i de menys capacitat més propers al processador i els menys ràpids però de major capacitat més llunyans.

D'aquesta manera, i gràcies a les propietats estadístiques de localitat temporal i espacial[20] de les aplicacions, s'aconsegueix reduir notablement els temps d'accés als dispositius de memòria sense sacrificar la capacitat d'emmagatzemament. Tanmateix, i degut a la pròpia natura estadística d'aquestes propietats, la jerarquia introdueix un considerable grau d'indeterminisme pel que fa als temps d'accés a memòria.

La jerarquia de memòries esta composada bàsicament per la memòria cau, la memòria principal i la memòria auxiliar o disc. La jerarquització entre la CPU i la memòria cau, per un costat, està normalment gestionada pel pròpia maquinari mentre que la existent entre memòria principal i la memòria auxiliar requereix de la cooperació del sistema operatiu.

20 La localitat temporal fa referència a la probabilitat que l'accés a una posició de memòria es torni a repetir en un futur pròxim. La localitat espacial fa referència a la probabilitat que l'accés a una posició de memòria estigui seguit per l'accés a una posició propera de memòria en un futur pròxim.

Figura 15: Jerarquia de memòries



3.12.1 Paginació

Com s'avançava en l'apartat 3.3 un dels serveis que proporciona el sistema operatiu, de manera cooperativa amb la unitat de gestió de memòria (MMU), és la virtualització de la memòria. Aquesta ens ofereix dos propietats bàsiques: protecció i traducció de memòria. La primera serveix perquè un procés no pugui accedir la memòria d'un altre. La segona proporciona la possibilitat d'adreçar linealment un espai de memòria fragmentat i, a més, adreçar més quantitat de la que realment existeix.

Més concretament, la traducció d'adreces de memòria s'assoleix mitjançant un procediment anomenat paginació pel qual una adreça de pàgina virtual, unitat de treball de la MMU, és traduïda a una adreça de pàgina de memòria real, principal o auxiliar (disc). En particular, quan una petició de pàgina virtual arriba a la MMU aquesta es cercada en la taula de traducció (TLB)[21] i en cas de trobar-s'hi es retorna el valor a la pàgina real corresponent. En cas contrari, es senyalitza una interrupció de fallada de pàgina que el sistema operatiu s'encarrega de gestionar de manera diferent segons aquesta pàgina es trobi en memòria principal o auxiliar. Mentre que en el primer cas es recrearà simplement una entrada a la TLB (*minor page fault*) en el segon cas, a més, caldrà realitzar prèviament la lectura de la mateixa de la memòria auxiliar (*major page fault*).

Finalment, el procés d'emmagatzemament i recuperació de pàgines a memòria auxiliar s'anomena *swapping* (intercanvi) i depèn d'un algorisme que en té en conté les necessitats de memòria principal que ha de gestionar el sistema operatiu.

3.13 Usos de la memòria

El sistema operatiu Linux utilitza la memòria principal per a tres objectius diferents: satisfer les necessitats de memòria dels processos, bé siguin d'usuari o de nucli; proveir un magatzem intermedi en l'accés als dispositius de tipus bloc, i millorar l'eficiència de l'accés a disc mitjançant tècniques de memòria cau.

21 Translation Lookaside Buffer: memòria cau utilitzada per la unitat de gestió de memòria (MMU) per a millorar la eficiència en la traducció d'adreces virtuals a adreces físiques.

3.13.1 La memòria de processos

Pel que fa a la memòria de processos es distingeixen en les següents seccions: el codi d'un programa (*text section*), les dades inicialitzades d'un programa (*data section*), les dades no inicialitzades d'un programa (*bss section*[22]), l'espai de pila (*stack*) i la memòria dinàmica (*heap*).

3.13.1.1 La capa slab

Dintre la memòria utilitzada pels processos cal destacar la capa *slab* utilitzada per a la reservació i alliberació de la memòria utilitzada en la representació de les estructures internes del nucli. Aquesta capa organitza la representació d'aquestes estructures en blocs de memòria (*slabs*) que permeten representar una certa quantitat d'elements de tal manera que quan el nucli requereix d'obtenir-ne o alliberar-ne li realitza una petició perquè li proveeixi un dels elements no utilitzats. Paral·lelament, aquesta capa realitza reservació de memòria per avançat i l'alliberació de manera diferida, agrupant les operacions per millorar-ne la eficiència i reduir-ne la fragmentació.

3.13.2 *Buffers* d'accés a dispositius de bloc

Linux, segons el tipus d'accés, classifica els dispositius en tipus caràcter o tipus bloc. En el primer cas les dades del dispositiu es reben en forma de fil, una darrera l'altre, i no és possible accedir aleatòriament a una posició donada. Un exemple d'aquest tipus de dispositiu serien el teclat o un port sèrie. Per altra banda, els dispositius de tipus bloc es distingeixen per la possibilitat d'accedir a un nombre fixat de dades de manera aleatòria, per exemple els discos d'emmagatzemament de dades.

Tenint en compte que el rendiment del sistema depèn en gran mesura de la gestió dels dispositius de bloc, Linux realitza la lectura i escriptura dels mateixos a través de *buffers*, fet que permet la planificació eficient del accés a disc. Per això, associa un *buffer* per a cada bloc del dispositiu (unitat mínima del sistema operatiu) a llegir o escriure, composat per un o més sectors (unitat mínima del dispositiu).

Així doncs, allotja temporalment les dades llegides o a escriure de manera que pot planificar un accés més eficient al dispositiu. Bé sigui minimitzant el moviment d'un capçal de disc -com es veurà més endavant a 3.14-, bé sigui realitzant lectures per endavant (*read-ahead*) o diferint les escriptures de manera que es puguin agrupar -en sistemes d'arxius asíncrons[23]-, entre altres.

3.13.3 Memòria cau de disc

Linux, a més, utilitza la memòria lliure no utilitzada pels processos i *buffers* com a memòria cau dels blocs llegits i escrits. D'aquesta manera, baix el principi de localitat temporal, cada cop que es realitza una lectura, una còpia de les pàgines dels blocs llegits es guarda en la memòria principal de manera que si cal tornar-hi a accedir, bé sigui per realitzar una lectura o una escriptura, es redueix significativament el seu temps d'accés.

Pel que fa a les escriptures, Linux utilitza una política *write-back*[24] en què l'escriptura d'un bloc de disc resident en la memòria cau es realitza en aquesta mateixa -quedant marcada com a *dirty*- i no és més endavant, de manera diferida, quan és actualitzada a disc. D'aquesta manera és possible agrupar diverses escriptures en una de major millorant la eficiència de l'accés a disc.

22 *Block Started by Symbol* és un acrònim amb origen històric utilitzat a día d'avui per indicar una secció de variables sense un valor inicial donat que, seguint les convenció d'inicialització del llenguatge C, s'inicialitzen a cero.[ISO07]

23 Aquells que no realitzen les escriptures en el mateix moment en que són requerides si no que ho fan de manera periòdica en una tasca de manteniment on s'agrupen les escriptures pendents optimitzant l'accés al dispositiu. Per defecte els sistemes d'arxius moderns -ext2 o ext3 entre altres- són *montats* com asíncrons tot i que existeix la possibilitat de fer-ho com a síncrons.

24 També anomenada *copy-back* o *write-behind*.

Finalment, quan la memòria utilitzada per la memòria cau de disc es reclamada per aquest mateix mecanisme o per a altres propòsits (memòria de processos o *buffers* de lectura/escriptura), Linux decideix quina serà la pàgina desallotjada utilitzant una estratègia anomenada *two-list strategy*.

Aquesta estratègia consisteix en una versió modificada de la política *Least Recently Used*[25] en la que és mantenen dues llistes anomenades pàgines actives i pàgines inactives on s'incorporen, d'un costat, per ordre temporal, aquelles que, estan ja en memòria cau, són reclamades un altre cop i, per altre costat, aquelles que o bé no han estat mai reclamades o, provinents de la llista d'actives, ho han estat menys recentment (LRU). D'aquesta manera, s'intenta evitar que pàgines utilitzades varis cops (calentes), tot i que no recentment, puguin ser eliminades de la memòria cau abans que altres que han estat utilitzades recentment però un o pocs cops i que, possiblement, degut a la seva natura no es tornaran a reclamar mai més.

3.14 Planificadors E/S

El planificador E/S realitza principalment dues tasques sobre les operacions a disc: la unió de dos o més peticions adjacents i la ordenació de peticions de manera que es minimitzin els moviments dels capçals de disc. Permet millorar, doncs, tot i no tenint un impacte directe en la reactivitat del sistema, l'assoliment dels terminis en aplicacions amb un ús intensiu de disc. En aquest aspecte el nucli de Linux 2.6 ofereix fins a quatre alternatives diferents: *deadline I/O*, *anticipatory*, *completly fair queuing* i *noop*.

3.14.1 *Deadline scheduler*

El planificador *deadline* manté tres cues de peticions: una de peticions ordenades segons el sector d'accés, una de peticions de lectura i una altre de peticions d'escriptura. Les cues de lectura i escriptura estan ordenades segons ordre d'arribada (FIFO) i, a més, tenen associat un temps d'expiració de 500ms i 5s respectivament.

Per norma, s'envien les peticions de la cua ordenada per sectors de manera que es minimitzen els moviments de capçals oferint un major rendiment. Tanmateix, quan els elements més vells d'alguna de les cues FIFO expiren el temps associat a la mateixa es comencen a servir aquestos. D'aquesta manera es prioritza el rendiment sempre que es respectin els terminis marcats.

Finalment, la diferència de temps d'expiració de la cua de lectures respecte a la d'escriptures -500ms en vers 5s- redueix el que és coneix com el problema dels *writes-starving-reads* en el que les lectures pel seu caràcter síncron -l'execució se sol bloquejar fins a satisfer la petició- es veuen desbordades per les escriptures, de índole asíncrona.

3.14.2 Anticipatory scheduler[26]

Basat en els mateixos principis que el *deadline* -una cua amb inserció segons sector i dues cues FIFO de terminis de lectura i escriptura- afegeix un temps d'espera de fins a 6ms durant els quals, després de una lectura, el planificador roman a l'espera de la següent petició. Aquest algorisme es basa, doncs, en la assumpció que les lectures, per norma, es realitzen de manera secuencial i que, normalment, després d'una lectura arribarà una altre de consecutiva. D'aquesta manera s'evita que la planificació, i en conseqüència el capçal, basculi entre la cua de lectures i la ordenada per sectors cada cop que expira el termini de la primera.

25 La estratègia LRU utiliza estampes de temps per realitzar un seguiment de les pàgines menys utilitzades recentment.

26 Degut a una resposta irregular del planificador amb rendiments de fins a 71% en servidors web o només fins un 15% en aplicacions de bases de dades aquest planificador ha estat eliminat del nucli en la versió 2.6.33. [LAY10]

Per altra banda, el planificador utilitza uns heurístics que basats en la recol·lecció de dades estadístiques intenten predir de manera més eficient quins processos realitzaran lectures seqüencials. D'aquesta manera s'assoleix que en una majoria de casos el temps d'espera sigui realment aprofitat per una altre lectura compensant la minoria en què el planificador estarà aturat sense servir peticions.

3.14.2.1 Planificador *completly fair queuing (CFQ)*

El planificador CFQ, per altra banda, utilitza una algorisme completament diferent que els dos anteriors. En aquest cas assigna una cua de peticions per a cada procés en execució. Per a cada cua les peticions adjacents són fusionades i ordenades segons el sector de disc. Les cues són servides segons la política de repartició *round robin* de quatre en quatre -per defecte. D'aquesta manera s'assoleix el repartiment just a nivell de procés.

3.14.3 *Noop scheduler*

Finalment, el planificador *noop*[27] únicament realitza la unió de les peticions adjacents i no realitza cap tipus de reordenació. És a dir, manté pràcticament una política FIFO on les peticions es van servint segons ordre d'arribada, a excepció de la fusió d'adjacents.

Aquest planificador està especialment dissenyat per treballar amb dispositius amb temps d'accés aleatori uniforme, com és el cas dels dispositius de memòria RAM o *flash*, o en casos on el maquinari controlador de l'accés sigui prou eficient organitzant les peticions. En aquests casos l'estalvi dels càlculs realitzats per la resta de planificadors permeten oferir un rendiment superior.

3.15 Dificultats per al temps real

En aquest capítol s'han vist els aspectes més rellevants del sistema operatiu Linux pel que fa a l'assoliment de propietats del temps real. S'ha exposat la seva arquitectura, basada en la separació dels espais de nucli (privilegiat) i usuari (no privilegiat), i la virtualització del processador i memòria en aquest segon. S'han plantejat les diferents polítiques de planificació (temps real i repartiment just), la gestió d'interrupcions, els problemes associats a la compartició de dades, com la sincronització pot resoldre'ls i com pot aparèixer el problema de la inversió de prioritats. Finalment, s'ha introduït el rellotge, element que determina el ritme del sistema, els mecanismes de gestió de memòria i la planificació E/S.

En aquest punt, doncs, tenint en compte que l'assoliment del temps real només pot venir donat pel determinisme (*jitter* conegut i acotat) i la velocitat de resposta (baixa), i tenint en consideració els diversos serveis i mecanismes del sistema operatiu exposats, ja es comencen a intuir quines són les dificultats a les que aquest s'enfronta per obtenir-lo.

En primer lloc, la multitasca i la protecció de l'accés al maquinari -garantides gràcies a la preempció i a la divisió d'espais respectivament- es tradueix en una intercalació d'accions i competició de recursos que el sistema ha de resoldre tenint en compte els diferents conflictes en la compartició de dades, tant d'usuari com d'estructures internes del nucli. La resolució s'assoleix gràcies a la preempció tot i que aquesta introdueix les seccions crítiques, camins de codi que no poden ser interromputs, i provoca la aparició del fenònem anomenat inversió de prioritats, ambdos efectes de impacte negatiu en el temps de resposta.

En segon lloc, el sistema d'interrupcions i els controladors de maquinari, necessaris per comunicar aquests dispositius amb el processador, introdueixen de forma asíncrona trencaments en el flux

27 Mnemonic construït a partir del terme anglosaxò *no operation*.

d'execució del codi i apropiació del processador. En aquesta situació, també, el sistema es veu sotmès, per una banda, als efectes secundaris de la sincronització i, per altra banda, a la impossibilitat de planificar una nova tasca.

Finalment existeixen altre factors, com la resolució del rellotge del sistema -en el rang dels milisegons-, la paginació de la memòria o l'accés als dispositius E/S que dificulten la precisió (determinisme) i velocitat de resposta (latència) en la que el sistema operatiu farà efectiva l'atenció dels dels esdeveniments produïts.

4 Linux com a sistema operatiu de temps real

4.1 Introducció

En la última dècada, Linux ha patit una sèrie de transformacions pel que fa al determinisme del sistema operatiu amb l'objectiu de dotar-lo de millor comportament en temps real. Aquests canvis, centrats principalment en la reducció i acotament de la latència, han estat impulsats per un sector de la comunitat interessada en aquest aspecte i, en un gruix considerable, han anat trobant el seu camí cap a la línia principal de codi del nucli.

En aquest capítol, doncs, prèvia formalització dels requeriments del temps real, és fa un repàs de l'evolució del nucli, s'enumeren les diferents solucions de disseny adoptades i, finalment, s'esmenten les condicions necessàries pel que fa al maquinari i al desenvolupament de codi per fer possible l'assoliment del temps real.

Aquest repàs, posteriorment, ha de servir per dissenyar, entendre i poder interpretar una sèrie de proves i mesures que reflecteixin el nivell de determinisme assolit pel sistema i indiquin el grau de consecució del mateix pel que fa als objectius de temps reals.

4.2 Requeriments del temps real

L'assoliment del temps real passa per un únic requeriment: garantir que el temps de resposta en els pitjors dels casos (R), sigui igual o estigui per sota del termini màxim de resposta (D). Per altra banda, el temps de resposta és pot dividir en: latència d'atenció (L) i pitjor temps d'execució (RWCET).

Aquest segon, alhora, per a una tasca de temps real de màxima prioritat, es pot dividir en: el pitjor temps d'execució de la tasca (WCET), el pitjor temps perdut en tasques de major prioritat (PRIO) o inversió de prioritats (INV) i el pitjor temps en apropiació de CPU per part de les interrupcions (INT) ocorregudes durant l'execució de la tasca.

En resum, per una tasca *i* tenim que:

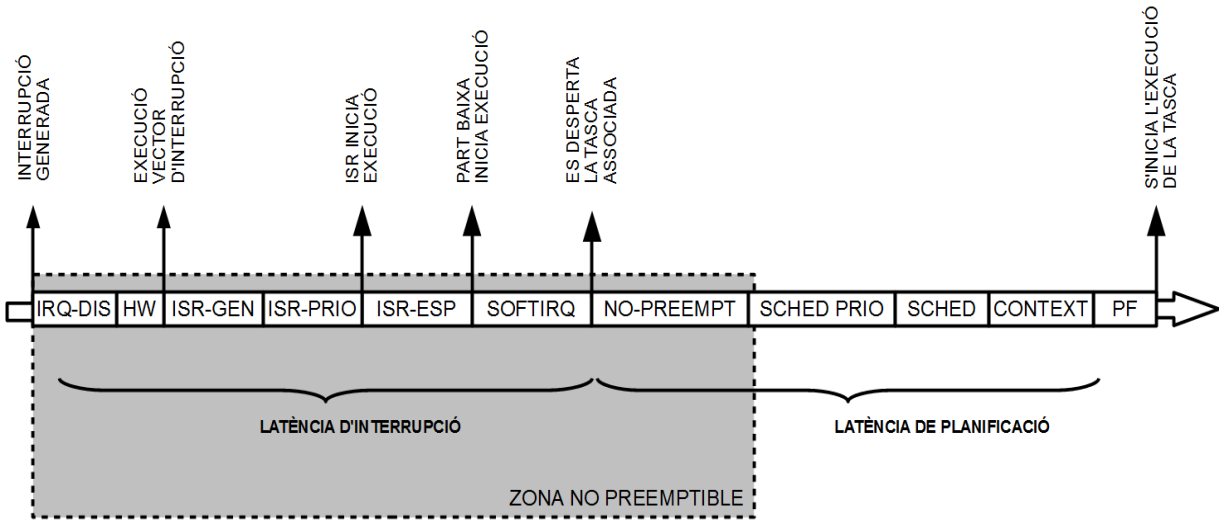
- $R_i \leq L_i + (WCET_i + INV_i + PRIO_i + INT_i)$

D'aquests, mentre que $WCET_i$ és responsabilitat principal del disseny de l'aplicació, de realitzar una implementació adient, L_i i INT_i ho són del disseny del sistema operatiu, de facilitar una reactivitat acotada. $PRIO_i$ i INV_i, per altra banda, són responsabilitat en part del disseny de l'aplicació, d'assignar la prioritat adequada, i del sistema operatiu, de proveir els mecanismes per garantir-la.

4.3 Orígens de la latència

La latència que es produeix des de l'aparició d'un esdeveniment fins l'execució de la tasca associada es pot dividir principalment en dos etapes ben diferenciades: temps en què el nucli està executant la interrupció generada per l'esdeveniment a atendre i d'altres (latència d'interrupció) i temps des de què una tasca és marcada com executable fins que aquesta inicia la seva execució (latència de planificació).

Figura 16: Retards inclosos en la execució d'una tasca associada a un esdeveniment



Aquestes dues etapes, alhora, es poden descomposar en els següents components:

- IRQ-DIS: Temps en que romanen les interrupcions desactivades per evitar condicions de carrera. Depèn del sistema operatiu.
- HW: Temps des de què es generada la interrupció fins que s'inicia l'execució de la primera instrucció apuntada pel vector d'interrupció. Depèn del maquinari: estat del processador, *pipeline*, bus o jerarquia de memòria, entre altres.
- ISR-GEN: Temps en executar el gestor d'interrupcions genèric. Inclou operacions de salvat de context, determinació de l'origen de la interrupció i invocació del gestor específic del IRQ senyalitzat. Depèn del sistema operatiu tot i que és una activitat prou curta i totalment determinista.
- ISR-PRIO: Temps en executar els gestors d'interrupcions de major prioritat, en cas de què s'hagin generat dos o més interrupcions simultàniament. Pel que fa a la priorització, depèn del maquinar, del sistema operatiu i de l'administració del mateix.[28] Pel que fa al temps d'execució depèn de la implementació dels controladors de dispositius.
- ISR-ESP: Temps en executar el gestor d'interrupcions de l'esdeveniment donat. Depèn de la implementació del controlador de dispositiu.
- SOFTIRQ: Temps en executar la part baixa del servei d'atenció a la interrupció. Depèn de la implementació del controlador de dispositiu.
- NO-PREEMPT: Temps en què el nucli roman en una secció de codi que no es pot interrompre. Depèn del sistema operatiu.

28 En el cas de les arquitectures PC està dictada per maquinari i segueix la priorització existent en els primers equips dissenyats per IBM que disposaven de dos controladors en cascada Intel 8259 de buit entrades d'interrupció cadascun amb la següent ordenació, de major a menor prioritat: IRQ0 *tick* del sistema, IRQ1 teclat IRQ8 rellotge de temps, IRQ9, IRQ10, IRQ11, IRQ12 ratolí, IRQ14 controladora de disc dur 1 i IRQ15 controladora de disc dur 2 -de la 8 a la 15 en cascada a l'entrada IRQ2-, IRQ3 port sèrie 2, IRQ4 ports sèrie 1, IRQ5 port paral·lel 2, IRQ6 controladora de disc flexible, IRQ7 port paral·lel 1 o tarja de só. [MSF06]

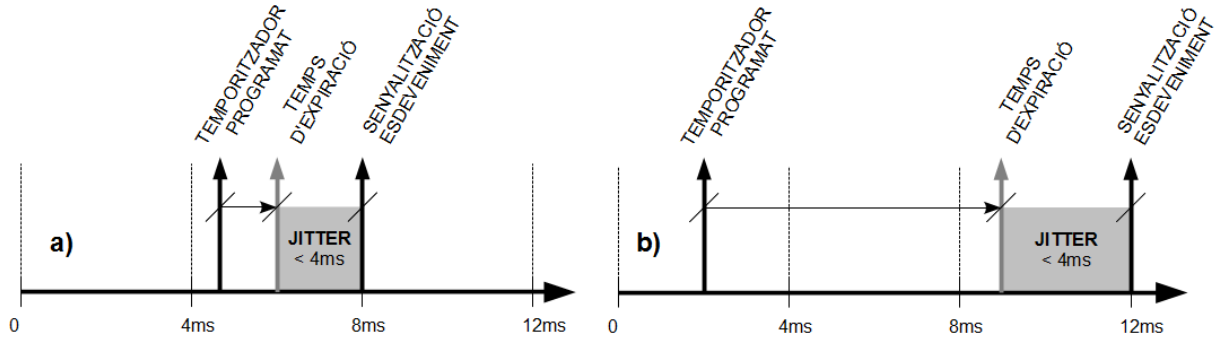
- SCHED-PRIO: Temps durant el qual s'executen tasques de major prioritat que la de la tasca associada a l'esdeveniment produït. Depèn del sistema operatiu i de les aplicacions d'usuari.
- SCHED-O(1): Temps en que es tarda en seleccionar la tasca de totes les que estan apunt per execució. En l'actualitat, aquest temps té un ordre de O(log N) per a les tasques normals -planificador CFS- i de O(1), és a dir constant, per a les tasques de temps real.
- CONTEXT: Temps per salvar el context anterior i restaurar el nou. Depèn del maquinari i del sistema operatiu.
- CACHE-MISS/PF: Temps que degut a no trobar-se el codi a executar en un cert nivell de la jerarquia de memòries cal reclamar-lo d'un nivell superior: memòria cau, principal o secundaria (disc). Aquest retard es pot donar en qualsevol canvi de context i depèn tant del sistema operatiu com dels controladors de dispositius i les aplicacions en espai d'usuari.

D'aquests components, alguns són de maquinari (HW i ISR-PRIO), altres depenen del disseny i implementació del sistema operatiu (IRQ-DIS, ISR-GEN, ISR-PRIO[29], NO-PREEMPT, SCHED, CONTEXT i CACHE-MISS/PF), altres depenen de la implementació dels controladors de dispositius (ISR-ESP i SOFT-IRQ) i, finalment, altres depenen del disseny de l'aplicació (SCHED-PRIO i CACHE-MISS/PF[30]).

4.3.1 Esdeveniments temporitzats

Pels esdeveniments temporitzats, a més, cal tenir en compte la latència que es dóna entre la expiració del temps programat i la senyalització de l'expiració. Aquest temps és degut a la baixa resolució del rellotge del sistema[31], del ordre dels milisegons, i es pot donar per dos motius: el temps programat és menor a la resolució del rellotge, i per tant es veurà arrodonit a aquesta, o la programació del temporitzador no és síncrona amb el propi rellotge. En ambdós casos es pot arribar a experimentar un *jitter* de la magnitud del propi *tick* del sistema.

Figura 17: Exemple de sistema amb rellotge de 250Hz (4ms) de temporització a) de menor a resolució i b) desincronitzada.



4.3.2 Pitjor latència

La pitjor latència (*worst-case latency*) està composada, doncs, per la suma de les pitjors latències d'interrupció imbricades no determinada més la suma de les pitjors latències de planificació que inclouen la imbricació no determinada de seccions crítiques no preemptibles. La millora del temps

29 Depenent del maquinari es possible o no assignar les prioritats de les línies d'interrupció (IRQ).
30 Tot i que el sistema operatiu és responsable de l'administració de la memòria virtual i la paginació, les aplicacions tenen un paper important en l'aparició de fallades de pàgina i memòria cau (CACHE-MISS/PF) segons el seu patró d'accés a la memòria que utilitzin.
31 En el apartat 3.9.1 es descriuen els motius d'aquesta baixa resolució.

de resposta pel que fa a la velocitat i -sobretot- la uniformitat passa, per tant, per la priorització i preempció d'interrupcions i la reducció de les seccions no preemptibles del nucli.

4.4 Evolució del nucli

Els primers esforços per millorar la latència i el *jitter* del sistema operatiu Linux daten del 2000. En aquest mateix any van aparèixer dos pedaços per al nucli anomenats *low latency patches* i *preemption patch*[32] desenvolupats per Ingo Molnar i Andrew Morton per a Red Hat [MOL00], i Nigel Gamble i Robert Love per a MontaVista [GAM00] respectivament. Ambdós projectes es van centrar a assolir un major grau de preempció del nucli, en aquell moment no preemptible, permetent d'interrompre'l en certs punts per a donar servei a tasques de major prioritat.

4.4.1 Low latency patches

La implementació dels *low latency patches* es fonamenta en la introducció explicita de punts de preempció en diversos llocs del nucli amb l'objectiu de fragmentar els camins d'execució llargs, sobretot pel que fa a la iteració d'estructures de dades (*lock-breaking*). Per al descobriment dels camins llargs del nucli, a més, realitza modificacions en el controlador del rellotge de temps real[33] que detecten latències majors a un llindar donat i les registren en un dietari de sistema.

En la figura següent es pot veure un exemple en el que s'itera una llista d'elements codi que podria esdevenir un camí d'execució llarg depenent de la longitud de la mateixa. Tanmateix, mitjançant la tècnica de *lock-breaking*, es comptabilitzen les iteracions i donat un cert llindar es permet la planificació d'una tasca diferent, reduint, per tant, la latència de planificació

Figura 18: Pseudocodi exemple preempció *lock-breaking*

```
Adquireix(A)
Mentre Quedin Elements
      # INICI LOCK-BREAKING
      Iteracions = Iteracions + 1
      Si Iteracions > 10 llavors
            Iteracions = 0
            Si Necessita Planificar llavors
                  Allibera(A)
                  # LOCK-BREAKING
                  Planifica i Executa Següent Tasca
                  Adquireix(A)
            FiSi
      FiSi
      # FI LOCK-BREAKING
      Operació amb Element
      Següent Element
FiMentre
Allibera(A)
```

32 Ambdós pedaços estan en desús i sense suport a data d'avui. Tanmateix consultant referències de la època per arquitectures x86 se'n extreu que "mentre que ambdós pedaços redueixen la latència del nucli, els *low-latency patches* són els guanyadors" (Williams 2002:18). [WIL02]

33 L'utilitzat per mantenir la data i hora del sistema, que no pas el rellotge del sistema en si tal com s'ha vist al capítol anterior.

4.4.2 Preemption patches

Els *preemption patches*, en canvi, aprofiten la labor realitzada en la fragmentació i reducció de seccions crítiques (*fine-grained lock*) realitzada en la sèrie 2.2 del nucli -dissenyada per a donar suport multiprocessador (SMP)- i redefineixen les macros[34] d'adquisició i alliberació de *spinlocks* i el codi de retorn d'interrupció per introduir punts de preempció. Aquest pedaç, a part d'aprofitar els *spinlocks* com a sentinelles de seccions crítiques, introdueix, també, instruccions a l'entrada i sortida de les seccions crítiques locals que manipulen estructures de dades exclusives a una CPU donada i que, per tant, no estan flaquejades pels primers.

La redefinició de les macros consisteix en la utilització d'un comptador de preempció (*preempt_count*) a mode de semàfor de manera que al incrementar-se en l'adquisició i decrementar-se en l'alliberació indica, quan el seu valor es diferent de zero, la presència del nucli en una secció crítica. Per altra banda, tant al retornar d'una interrupció com al alliberar un *spinlock*, si aquest comptador té valor cero -el nucli no està en una secció crítica- i hi ha alguna planificació pendent -algun procés ha despertat[35] durant la no preempció- es realitza la planificació i canvi de tasca, si cal.

En la següent figura es pot veure una simplificació en pseudocodi de la implementació actual de les macros esmentades. En aquesta, s'observa la utilització el comptador *preempt_count* incrementant-lo al entrar i decrementant-lo al sortir d'una secció crítica[36] així com la verificació de la existència d'alguna tasca pendent de planificació (*need_resched*) mentre no existeix cap secció crítica en curs, comptador igual a cero, al sortir d'una secció crítica i al tornar d'una interrupció.

Figura 19: Pseudocodi preempció amb *spinlock i ret_from_intr*

```
#Inhibició de la preempció
preempt_disable()
      preempt_count = preempt_count + 1

#Habilitació de la preempció i comprovació planificació
preempt_enable()
      preempt_count = preempt_count - 1
      Si preempt_count = 0 I Necessita Planificar llavors
            Planifica i executa següent tasca

#Bloqueig spinlock amb inhibició de la preempció
spin_lock(A)
      preempt_disable()
      Adquireix(A)

#Bloqueig spinlock amb habilitació de la preempció i comprovació planificació
spin_unlock(A)
      Allibera(A)
      preempt_enable()
```

34 En el llenguatge C, conjunt d'instruccions agrupades sota un únic estament que en ser tractades pel compilador en una etapa anomenada preprocessament s'expandeixen al conjunt que defineix l'estament únic.

35 Aquesta situació es comprova a través d'un *flag* global de nom *need_resched*.

36 S'utilitza *preempt_disable* o *spin_lock,* que també inclou *preempt_disable*, al entrar en una secció crítica o *preempt_enable* o *spin_unlock*, que també inclou *preempt_enable*, al sortir-ne segons si les variables de la secció són únicament accessible pel propi processador (*preempt-safe*) o altres processadors (*SMP-safe*).

```
#Retorn d'interrupció i comprovació de la planificació
ret_from_intr()
      Operacions retorn
      Si preempt_count = 0 I Necessita Planificar llavors
            Planifica i executa següent tasca
```

4.4.3 Real-time preempt patch

L'any 2004, Ingo Molnar, va iniciar la realització d'un pedaç anomenat *real-time preempt patch*, conegut també com PREEMPT_RT. Aquest pedaç, que se segueix en desenvolupament de manera activa en l'actualitat, és el cos principal de treball entorn a la millora del temps real en la línia principal de codi del nucli i esta essent incorporat a aquesta de manera gradual, en forma d'opcions de configuració, a mida que certes parts van guanyant maduresa i rellevància pel que fa al interès general.

D'aquest pedaç cal destacar algunes aportacions molt significatives[37] que han permès des de la preempció total del nucli[38] fins a l'ús de fonts de temps de precisió o la reducció del cost de planificació, entre altres. Aquestes es tracten, més concretament, de: l'eliminació o reducció dels camins de codi amb inhibició d'interrupcions, la conversió de *spinlocks* a *mutex* amb herència de prioritats, la adaptació del mecanisme RCU a un amb possibilitat de bloqueig (RCUS), la execució de gestors d'interrupció i *softirqs* en context de procés o el suport a temporitzadors d'alta resolució (HRT), que es desenvoluparan a continuació; així com la creació del planificador utilitzat en la actualitat[39], el CFS de O(logN) per al repartiment just i el sistema de dues cues per prioritat i CPU de O(1) per a tasques RT, que s'han pogut veure amb anterioritat.

4.4.4 Línia principal

La següent taula resumeix les fites més significatives pel que fa a l'evolució de la línia principal del nucli pel que fa a preempció i altres funcionalitats encarades a millorar la resposta en temps real del sistema operatiu.

Taula 3 Fites del temps real en la línia principal del nucli

Versió de nucli	Nivell de preempció
1	- No preempció del nucli, ni suport per a multiprocessador.
2	- Suport multiprocessador (SMP) amb protecció de concurrència mitjançant bloqueig total en nucli amb el *Big Kernel Lock* (*BKL*).
2.2	- Disminució de la contenció en multiprocessadors (SMP) mitjançant l'acotament de seccions crítiques a través de *spinlocks*.
2.5	- Introducció de la preempció en el nucli mitjançant tècniques *lock-breaking* i l'ús dels *spinlocks* per a la inhibició i habilitació de la preempció.
2.6	- Introducció del mecanismes RCU amb possibilitat de preempció de lectors (SRCU). - Possibilitat d'interrompre la gestió d'interrupcions mitjançant la seva execució en context de procés. - Introducció dels temporitzadors d'alta resolució. - Introducció del nou planificador O(1).

37 No totes elles han estat incorporades a la línia principal del nucli a data d'avui.
38 A excepció de seccions crítiques molt concretes com, per exemple, el planificador.
39 En comparació amb l'antic planificador -amb una sola cua per totes les prioritats- de O(N).

Versió de nucli	Nivell de preempció
2.6 PREEMPT_RT	- Preempció de seccions crítiques no competides mitjançant la conversió de *spinlocks* a *mutex* amb herència de prioritats. - Preempció dels gestors d'interrupció mitjançant la seva execució en context de procés.

4.5 Modalitats de preempció

A data d'avui, en la sèrie 3.x, conviuen fins a tres modalitats de preempció distintes en la versió principal del nucli: NO_PREEMPT, PREEMPT_VOLUNTARY i PREEMPT. A més, amb la incorporació del pedaç PREEMPT_RT, s'afegeix una quarta anomenada PREEMPT_RT i PREEMPT passa a anomenar-se PREEMPT_LL.

Taula 4: Modalitats de preempció al nucli 3.x

Modalitat	Descripció segons Kconfig[40]
NO_PREEMPT	*This is the traditional Linux preemption model, geared towards throughput. It will still provide good latencies most of the time, but there are no guarantees and occasional longer delays are possible.* *Select this option if you are building a kernel for a server or scientific/computation system, or if you want to maximize the raw processing power of the kernel, irrespective of scheduling latencies.*
PREEMPT_VOLUNTARY	*This option reduces the latency of the kernel by adding more "explicit preemption points" to the kernel code. These new preemption points have been selected to reduce the maximum latency of rescheduling, providing faster application reactions, at the cost of slightly lower throughput.* *This allows reaction to interactive events by allowing a low priority process to voluntarily preempt itself even if it is in kernel mode executing a system call. This allows applications to run more 'smoothly' even when the system is under load.* *Select this if you are building a kernel for a desktop system.*
PREEMPT / PREEMPT_LL	*This option reduces the latency of the kernel by making all kernel code (that is not executing in a critical section) preemptible. This allows reaction to interactive events by permitting a low priority process to be preempted involuntarily even if it is in kernel mode executing a system call and would otherwise not be about to reach a natural preemption point.* *This allows applications to run more 'smoothly' even when the system is under load, at the cost of slightly lower throughput and a slight runtime overhead to kernel code.* *Select this if you are building a kernel for a desktop or embedded system with latency requirements in the milliseconds.*
PREEMPT_RT (Requereix pedaç)	*This option further reduces the scheduling latency of the kernel by replacing almost every spinlock used by then kernel with preemptible mutexes and thus making all but the most critical kernel code involuntarily preemptible. The remaining handful of lowlevel non-preemptible codepaths are short and have a deterministic latency of a couple of tens of microseconds (depending on the hardware).* *This also allows applications to run more 'smothly' even whn the system is under load, at the cost of lower throughput and runtime overhead to kernel code.*

40 Fitxer utilitzat durant la configuració de la compilació del nucli de Linux.

4.5.1 NO_PREEMPT

Aquest és el model tradicional de Linux, heretat de Unix, on el nucli no és preemptiu. Les crides de sistema no es poden interrompre i el canvi de tasca només pot ocórrer al final d'una crida de sistema o quan la tasca es bloquegi, bé sigui voluntàriament o perquè ha esgotat la seva *timeslice*.

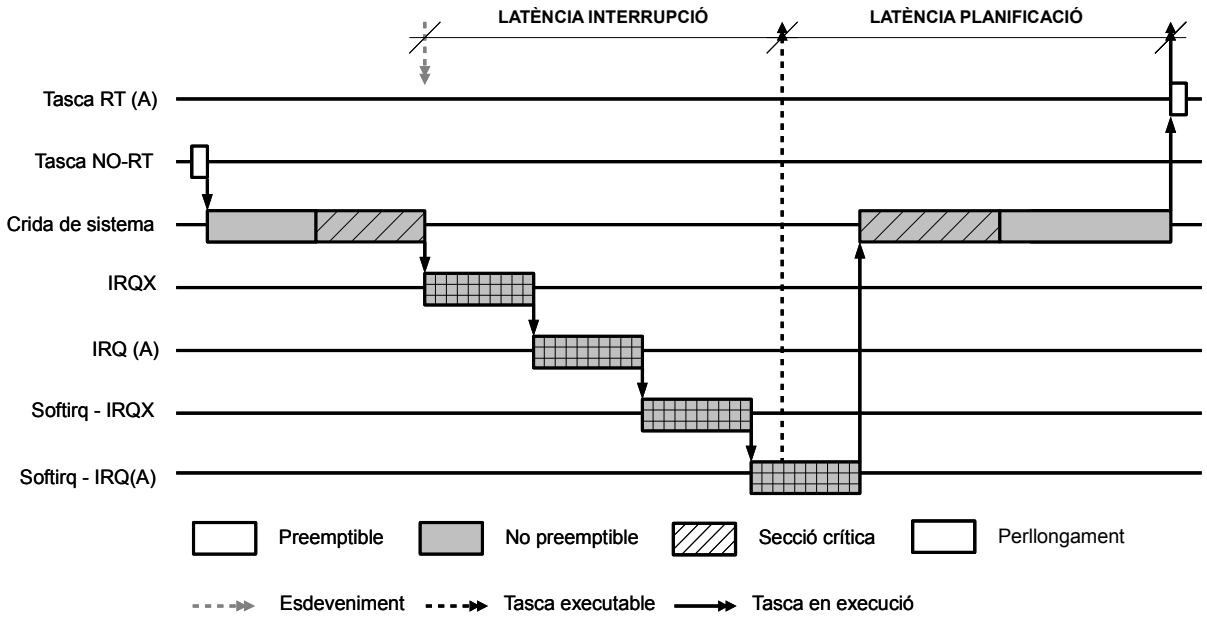
Aquest mode, teòricament, presenta el millor rendiment al minimitzar el nombre de canvis de context i per tant maximitzar la càrrega útil[41] del processador. Per contrapartida la longitud dels camins de les crides de sistema són llargs -poden arribar a dècimes de segon-, no són deterministes -durant l'execució de la crida es poden produir diferents interrupcions- i , com a conseqüència, presenten una gran variància.

Aquesta configuració, per tant, és adequada per a servidors de treball on l'aspecte més rellevant a preservar es la eficiència i la maximització de la utilització del equip (*best-effort*). Tanmateix, al no ser determinista, no és adequada per a sistemes amb requeriments de temps real de cap tipus.

En la següent figura es pot observar un cronograma on es produeixen dues interrupcions durant l'execució d'una crida de sistema d'una tasca qualsevol sense necessitats de temps real. Una corresponent a un esdeveniment de temps real (A) mentre que l'altre correspon a un esdeveniment qualsevol (X) sense necessitats de temps real. Per un costat, la latència d'interrupció augmenta al prioritzar erròniament l'execució del gestor d'un esdeveniment donat -part alta (IRQ) i part baixa (*softirq*)- abans que el de temps real. Per altre costat, la latència de planificació augmenta també quan aquesta no es realitza fins que finalitza la crida de sistema que, a més, és estesa per l'execució dels gestors d'interrupció que es puguin produir durant la mateixa.

En aquestes condicions, doncs, degut a la longitud de la crida de sistema i la incertesa sobre la seva extensió, el sistema operatiu no pot garantir que el temps transcorregut des de l'aparició de l'esdeveniment de temps real (A) fins a la planificació de la tasca que l'ha d'atendre (Tasca RT) està per sota dels límits imposats per les condicions de temps real.

Figura 20: Mode de preempció NO_PREEMPT



41 En aquest cas s'entén càrrega útil com aquella utilitzada en tasques de l'espai d'usuari.

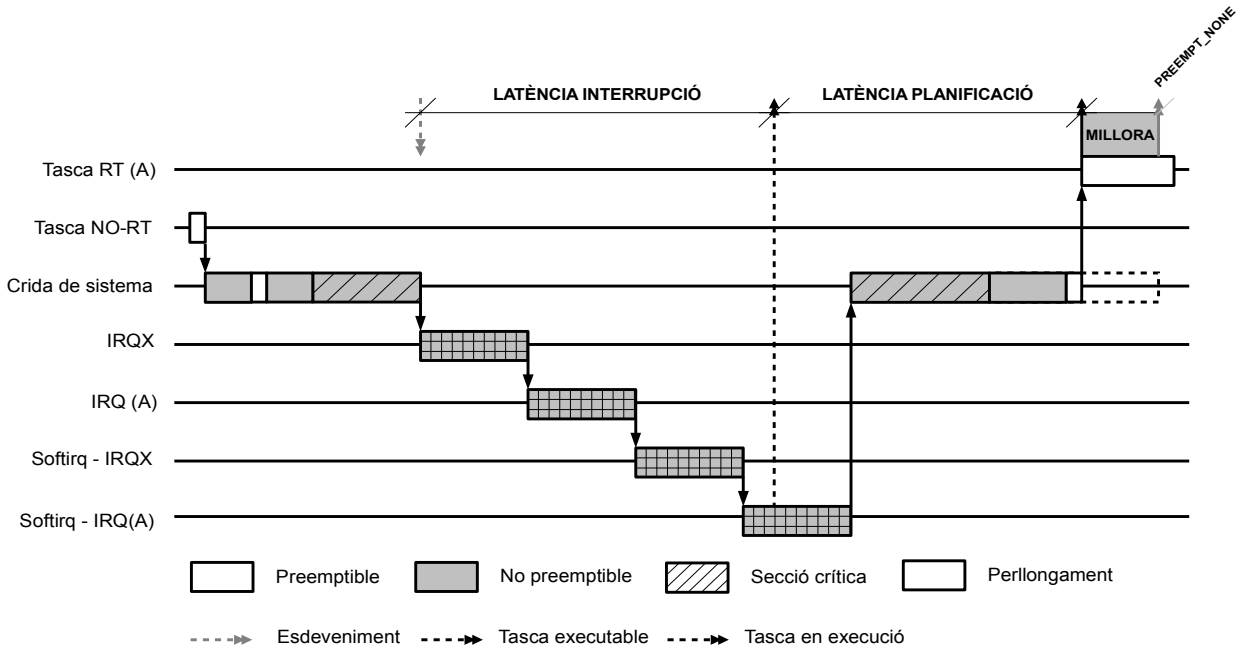
4.5.2 PREEMPT_VOLUNTARY

Un primer pas cap a la reducció de zones no preemptibles del nucli es troba en el mode de preempció voluntària. Aquest utilitza una tècnica similar al *lock-breaking* utilitzat prèviament en els *low-latency patches* de Molnar i Morton introduint crides explícites de preempció en les seccions crítiques més llargues del nucli, principalment la iteració de llistes, permet interrompre una crida de sistema en punts segurs del mateix.

En la següent figura, que reprodueix les condicions descrites en el cronograma de l'apartat anterior, es pot observar com la preemptibilitat del nucli millora la latència de planificació al permetre la mateixa abans de que finalitzi la crida de sistema, tot i que només parcialment al permetre-la només en certs punts. A més, la priorització d'interrupcions segueix sense concordar amb les necessitats del temps real i prioritza l'atenció de la interrupció d'un esdeveniment qualsevol per sobre de la d'un de temps real (A) i, per tant, segueix sotmeten la latència d'interrupció, com en la modalitat anterior *NO_PREEMPT*, a l'indeterminisme intrínsec de l'instant d'aparició dels esdeveniments.

En aquestes condicions, de nou, tot i la millora en la latència de planificació i degut a la longitud de la crida de sistema i la incertesa sobre la seva extensió, el sistema operatiu tampoc pot garantir que el temps transcorregut des de l'aparició de l'esdeveniment de temps real (A) fins a la planificació de la tasca que l'ha d'atendre (Tasca RT) està per sota dels límits imposats.

Figura 21: Mode de preempció PREEMPT_VOLUNTARY



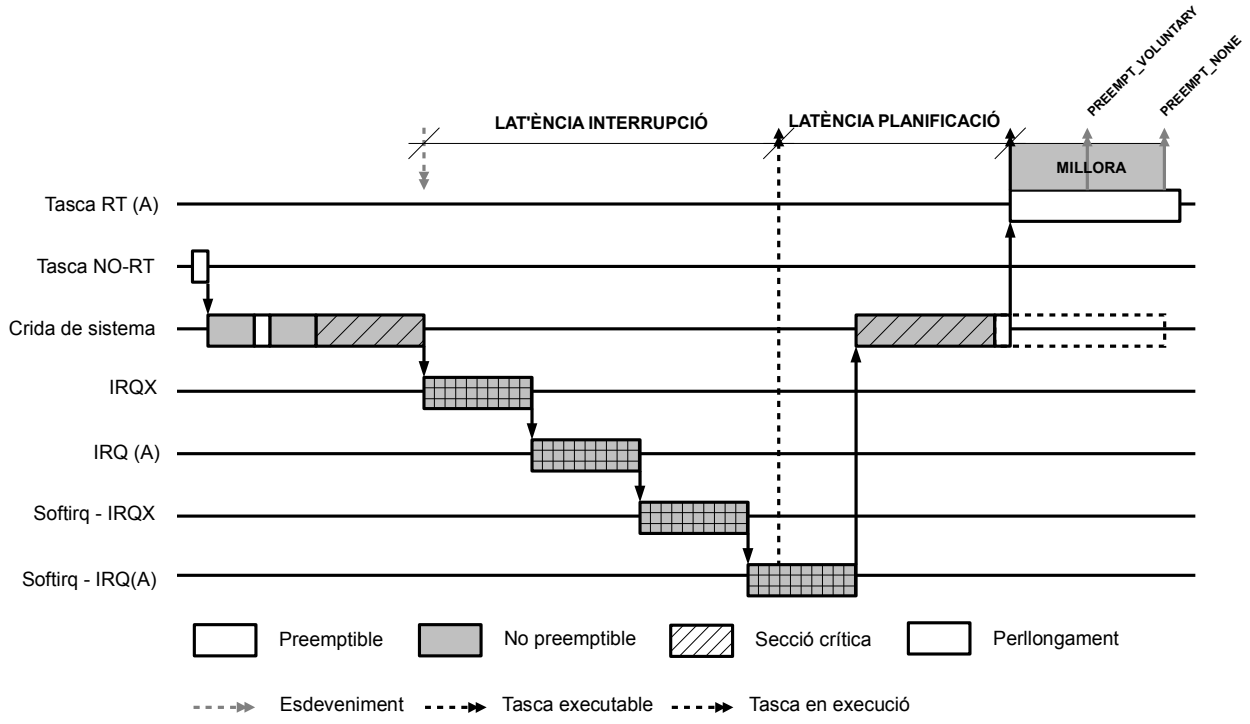
Aquesta configuració, per tant, és adequada quan una resposta més ràpida del sistema és desitjable tot i que no es garantida dintre de certs terminis, per exemple un ordinador de sobretaula de propòsit general on es habitual una major interactivitat i per tant es desitjable una major responsivitat. Per contrapartida, el major nombre de canvis de context produïts per la interrupció i canvi de tasques durant les crides de sistema podria empitjorar, en certa mesura, la eficiència del sistema i, per tant, impactar en el seu rendiment.

4.5.3 PREEMPT / PREEMPT_LL

La segona opció -i última pel que fa a la línia principal de nucli- a més de la introducció explícita de punts de preempció permet, de manera similar als *preemption patches* de Gamble i Love, la preempció del nucli en qualsevol punt a excepció de les interrupcions i les zones crítiques. De manera opcional[42], aquest mode suporta, també, l'execució d'interrupcions en context de procés oferint la possibilitat que, per un costat, es puguin prioritzar i, per altre costat, interrompre en l'arribada d'un esdeveniment de major prioritat.

En la següent figura es pot observar com, comparativament, la possibilitat de preempció del nucli en qualsevol secció no crítica -la tasca RT es planifica al finalitzar la mateixa-, millora la latència de planificació encara més que el model anterior. Tanmateix, al no permetre la interrupció de cap secció crítica, fins i tot en aquelles condicions on no existeix una contenció real de les estructures protegides, segueix sense poder garantir que el temps transcorregut des de l'aparició de l'esdeveniment de temps real (A) fins a la planificació de la tasca que l'ha d'atendre (Tasca RT) està per sota dels límits imposats.

Figura 22: Mode de preempció PREEMPT / PREEMPT_LL



Aquest mode millora sensiblement les capacitats de l'anterior i ofereix una resposta més ràpida i uniforme, amb menor *jitter*, al augmentar l'àrea de preempció del nucli[43]. Al no poder garantir les condicions de resposta en temps real, però, és adequat únicament per a tasques amb requeriments *soft real-time* com poden ser aplicacions multimèdia (àudio i vídeo) amb requeriments de servei de qualitat (*QoS*) o sistemes de control encastats on la fallada esporàdica d'un termini sigui assumible.

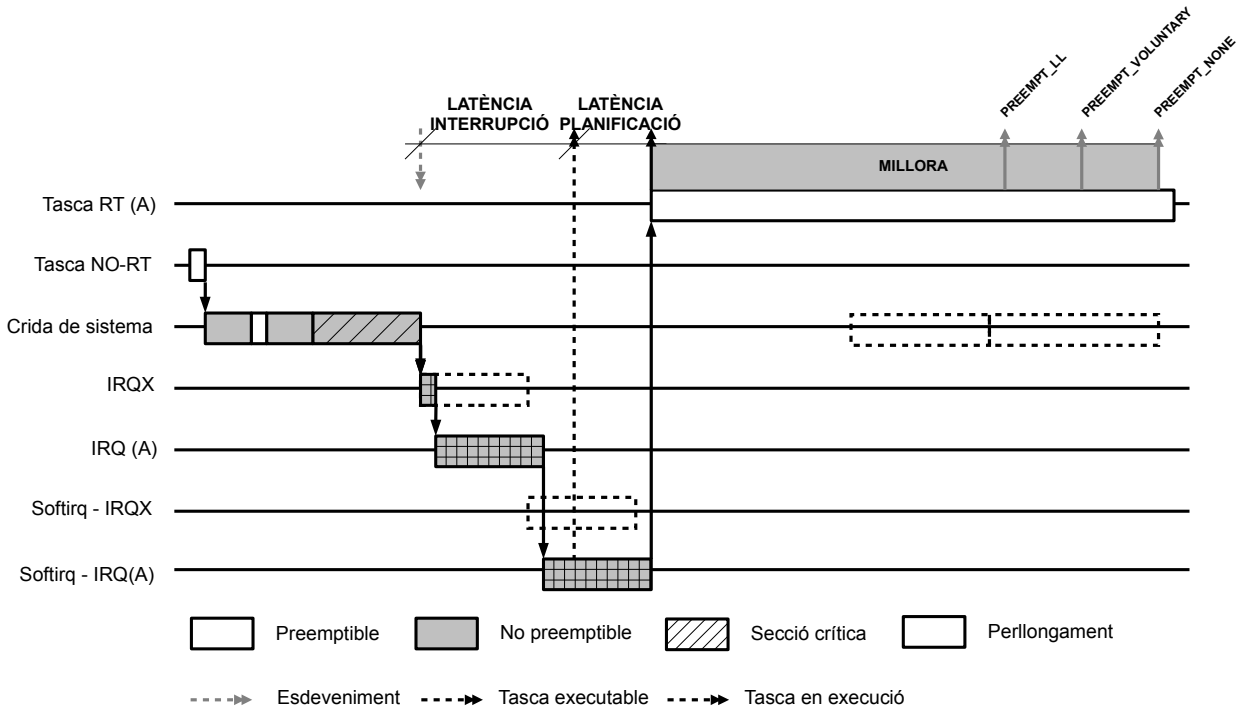
42 Mitjançant l'ús de la línia de comanda del nucli *threadirqs*. La línia de comanda del nucli cal especificar-la en el *bootloader* utilitzat.
43 Només queda exclosa una petita part: interrupcions i seccions crítiques; o seccions crítiques, únicament, si es carrega el nucli amb la línia *threadirqs*.

4.5.4 PREEMPT_RT

L'últim mode de preempció possible, accessible a través del pedaç PREEMPT_RT, aprofita la relativa baixa probabilitat de què en un instant determinat dos tasques concurrents comparteixin la mateixa secció crítica per substituir la majoria de *spinlocks*, que inhibeixen totalment la preempció del nucli, per *mutex* amb herència de prioritats, que només la impossibiliten en cas de competició del *lock*. A més, incorpora l'execució de les interrupcions en context de procés i redueix el nombre i la longitud del codi amb inhibició d'interrupcions limitant la àrea no preemptible del nucli a un conjunt reduït seccions ineludibles.

En la següent figura, que segueix les tres anteriors, es pot observar com, per una banda, en executar les interrupcions en context de procés es possible ajustar la seva prioritització en concordança amb les necessitats del temps real i, per tant, l'esdeveniment de temps real (A) es atès abans que qualsevol altre (X) i com, per altra banda, només finalitzar l'atenció del mateix, i al estendre la preempció del nucli a qualsevol àrea del mateix, incloses les seccions crítiques, es planifica la seva tasca de temps real (Tasca RT).

Figura 23: Mode de preempció PREEMPT_RT



4.5.4.1 Contrapartides en la preempció de seccions crítiques

Cal tenir present, tanmateix, que l'augment de canvis de context no és només producte de canvis de tasca efectius si no, també, de canvis de tasca no útils, seguits per un retorn a la tasca interrompuda per efecte d'una herència de prioritats produïda en la competició d'un *mutex* prèviament adquirit per la primera tasca. Aquest efecte, a més, es pot veure potenciat per un allau d'herències de prioritat que es pot arribar a produir en una cadena de reservació de recursos i que, alhora, empitjora també la pitjor latència.

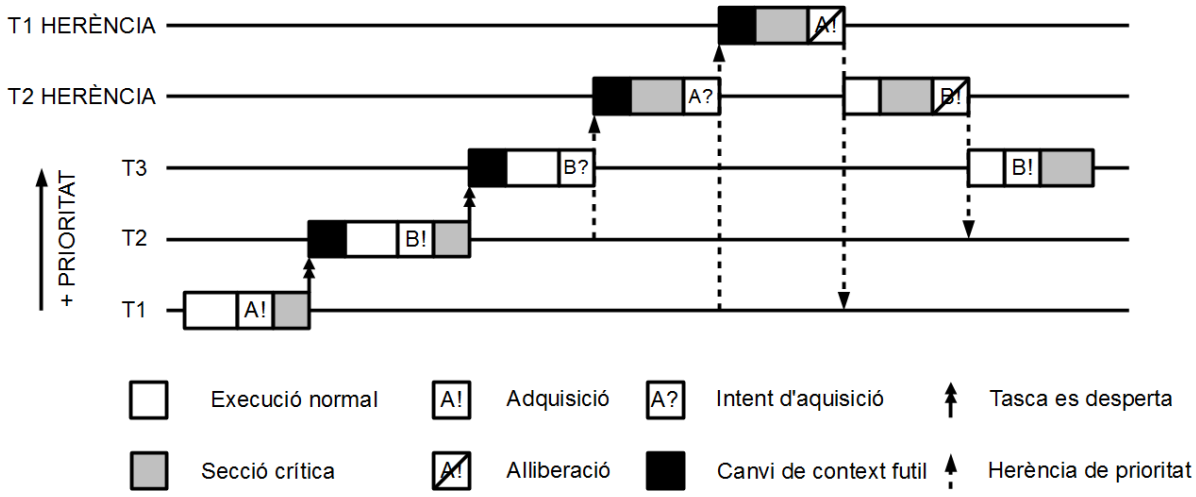
Per tant, tot i que la preempció de seccions crítiques millora notablement la latència d'atenció d'un esdeveniment per la majoria dels casos, en el cas d'una coincidència d'esdeveniments donada pot

arribar a oferir pitjors resultats dels obtinguts en un moment determinat. Aquest és un aspecte que cal tenir en compte alhora de realitzar afirmacions sobre la capacitat de temps real del sistema, sobretot si aquestes han estat elaborades a partir de resultats obtinguts en un període curt de temps o sota condicions que no ofereixin una probabilitat prou alta d'ocurrència de la majoria de casos pel que fa a la competència dels *mutex* entre les diferents tasques del sistema.

En la figura següent s'observa un exemple de canvis de context no útils en una cadena de reservació de dos elements. En un primer instant, la tasca T2 interromp la tasca T1 mentre aquesta posseeix un *lock* sobre A i, a continuació, T3 interromp a T2 mentre aquest posseeix un *lock* sobre B. Més endavant, quan T3 intenta accedir a B, és produeix una herència de prioritats en què T2 és elevada sobre T3 perquè pugui alliberar B i, a continuació, T1 és elevada sobre T2 perquè pugui alliberar A.

Finalment, les tasques s'han acabat executant en l'ordre en què s'havien produït (T1, T2 i T3) i la interrupció de les seccions crítiques de A i B només ha servit per realitzar dos canvis de context (T2 interromp a T1 i T3 interromp a T2) que han estat revertits (T2 interromp a T3 i T1 interromp a T2) al comprovar-se que no podien prosperar. En definitiva, s'ha afegit un sobrecost de quatre canvis de context no útils al temps total d'execució de les operacions i, pitjor encara per al temps real, a la latència d'atenció de les tasques aparegudes (T2 i T3).

Figura 24: Canvi de context no útil



Aquesta configuració, per tant, ofereix la base pel desenvolupament d'aplicacions *firm real-time* o, sota algun procés de certificació, fins i tot, *hard real-time*[44]. Per contrapartida, l'aplicació d'aquest pedaç pot significar una degradació considerable del rendiment del sistema producte del sobrecost de la implementació d'un *mutex* amb herència de prioritats en comparació amb un *spinlock* i l'augment notable de canvis de context, útils i no útils.

44 Les condicions que s'ha de complir per la afirmació d'una certa capacitat del sistema operatiu o qualsevol peça de programari o maquinari, com pot ser el temps real, forma part de la sèrie de criteris establerts en els processos especificats per les organitzacions de certificació que cobreixen diferents graus de criticisme.

4.6 Interrupcions en context de procés

A partir de la sèrie 2.6, segons configuració[45] o mitjançant pedaç PREEMPT_RT, s'introdueix la possibilitat d'executar les interrupcions, part alt i part baixa, en context de procés. Aquest fet possibilita la priorització i la preempció de les interrupcions per part de tasques de major prioritat reduint la àrea de no preempció del nucli i permetent una menor latència.

En el model tradicional al rebre una interrupció s'executen primer tots els gestors (part alta) i, després, les *softirqs* (part baixa) segons la prioritat estàtica definida en temps de compilació. En comparació, en el nou model, la majoria de gestors d'interrupció[46] s'executen en un fil del nucli[47] i les *softirqs* passen a planificar-se directament com a tasca del *daemon softirqd*.

D'aquesta manera, l'execució d'un gestor d'interrupció, així com de les parts baixes, depèn de la prioritat assignada al fil en que s'executa[48] i, gràcies a la possibilitat de planificació, permet la priorització, preempció i resum de la mateixa. L'ordenació, per un costat, elimina la inversió de prioritats directa i la preempció, per l'altre, permet la disminució dels temps de resposta de les tasques de major prioritat.

4.7 Rellotges d'alta resolució

A partir de la sèrie 2.6 el nucli incorpora en la línia principal de codi el subsistema de temporitzadors d'alta resolució (HRT). Aquest subsistema, més precís que el rellotge del sistema, funciona de manera independent i possibilita resolucions, a nivell de microsegon, per sota del mateix.[49] La utilització d'aquest subsistema es basa l'ús de dispositius de maquinari especialitzats com són l'Advanced Programmable Interrupt Controler (APIC) o el High Precission Event Timer (HPET).

Aquests dispositius que permeten la programació simultània de diferents temporitzadors amb una resolució de microsegons. D'aquesta manera el subsistema HRT utilitza directament aquest dispositiu per programar cadascun dels temporitzadors i, per tant, permet la senyalització dels mateixos en l'interval de temps demanat exactament i eviten, d'aquesta manera, la latència de temporització esmentada en l'apartat 4.3.1 .

4.8 PI mutex

Amb el pedaç *PREEMPT_RT* s'inclou la possibilitat d'interrompre el nucli fins i tot quan s'esta executant una secció crítica. Per això es substitueixen els *spinlocks*, que inclouen inhibició de la preempció, per zones d'exclusió mútua[50] implementades amb herència de prioritats (*PI mutex*). D'aquesta manera la secció crítica d'una tasca de menor prioritat només impossibilita l'execució d'una tasca de major si existeix una competició del recurs bloquejat, en comparació amb l'ús de *spinklocks* i inhibició de la preempció que ho fa fins i tot quan no hi ha contenció.

Aquest *mutexes*, per altra banda, presenten les següents particularitats: execució ràpida de l'adquisició i alliberació del *lock* en els casos que no està competit, herència de prioritats transitiva -és a dir, la prioritat heretada d'una tasca és heretada per una segona tasca si aquesta segona

45 Cal compilar el nucli amb la opció PREEMPT, o PREEMPT_LL si s'aplica el pedaç, i, a més, carregar el nucli amb la línia de comanda *threadirqs*.
46 A excepció de gestors crítics pel funcionament del sistema com el rellotge (irq0) que es segueix executant en context d'interrupció.
47 Per defecte amb model de planificació SCHED_FIFO i prioritat 50, la meitat de tot el rang.
48 Administrable en temps d'execució amb la comanda *chrt* (*change real-time attributes of a process*).
49 En el apartat 3.9.1 es descriuen els motius de la baixa resolució del rellotge de sistema.
50 Els *spinlocks* així com les zones d'exclusió mútua o *mutex* estan explicats amb més detall al apartat 3.10.4 .

intenta adquirir un recurs propietat de la primera- i, finalment, limitació a un nivell màxim de profunditat en l'herència per tal d'evitar una degradació significativa en el rendiment al gestionar cadenes de reservació de recursos.

4.9 SRCU

A partir de la sèrie 2.6 del nucli s'incorpora la possibilitat d'utilitzar un nou mecanisme RCU, anomenat *Sleepable RCU* (SRCU), que disminueix la latència produïda per les seccions crítiques de les tasques lectores que utilitzen aquest mecanisme al permetre la seva preempció. Com a conseqüència les tasques escriptores ja no poder comptar amb què la l'execució del planificador en una CPU donada significa implícitament la finalització de qualsevol referència RCU lectora a la estructura de dades modificada com s'havia vist en l'apartat 3.10.5 .

En aquesta nova implementació s'utilitzen dos comptadors de referència per CPU i tres llistes per la comptabilització i senyalització de la fi de la referència a una antiga copia per part d'un codi RCU lector. El parell de comptadors s'utilitzen de manera idèntica, com un semàfor, incrementant-se a l'inici de cada secció de lectura i decrementant-se a la seva fi.

L'ús de dos comptadors permet solapar la gestió en dos lots: incorporació d'antigues copies pendents d'eliminació i control de fi de referència per a la eliminació. Les tres llistes s'utilitzen per: referència a comptadors del primer lot, referències a comptadors del segon lot i, quan tots els comptadors de la segona arriben a cero, referències a copies sense lectures pendents de finalització.

La gestió d'aquest nou mecanisme, tanmateix, és significativament més costosa que el mètode tradicional i, per això, es presenta com una opció de configuració del nucli on es permet triar entre un i altre model segons es prefereixi comprometre latència a rendiment (mecanisme clàssic RCU) o a l'inrevés (mecanisme amb lectors preemptibles RCUS).

4.10 Tècniques de programació

L'assoliment del temps real en un sistema operatiu Linux es basa tant en la possibilitat del nucli d'oferir uns certs requeriments com l'habilitat de l'aplicació de treure'n profit d'aquestos per garantir el compliment dels terminis. En aquest apartat s'examinen breument algunes tècniques de programació normalment utilitzades en la creació d'aplicacions de temps real en llenguatge C recomanades en alguna de la literatura docent [SAL10].

4.10.1 Extensions POSIX

Davant la proliferació de sistemes UNIX i les seves corresponents intefícies de programació (API) l'Institute of Electrical and Electronic Engineers (IEEE), en un esforç d'estandardització, crea les normes Portable Operating System Interface (POSIX)[IEEE01]. D'aquestes, pel que fa al temps real, cal destacar la norma POSIX 1003.1d-1999 Real-time extensions i la norma POSIX 1003.1c-1994 Threads Extensions que inclouen, entre altres, funcionalitat addicional com el suport a temporitzadors d'alta resolució o la planificació amb prioritats estàtiques. En la següent taula es resumeixen les interfícies C utilitzades en aquest apartat.

Taula 5: Resum d'interfícies POSIX 1003.1D-1999 i 1003.1c-1994

Crida	Descripció
Planificació	
sched_setscheduler	Estableix la política de planificació.
sched_setparam	Estableix els paràmetres de planificació, entre ells, prioritat.
sched_get_priority_max	Retorna el valor màxim de prioritat segons política de planificació donada.
sched_get_priority_min	Retorna el valor mínim de prioritat segons política de planificació donada.
sched_yield	Cedeix la CPU.
pthread_setschedparam	Estableix la política de planificació i la prioritat d'un fil.
pthread_mutexattr_init	Creació d'un *fast user-space mutex* (*futex*).
pthread_mutex_setprotocol	Estableix el protocol de gestió de prioritats utilitzat.
pthread_mutex_init	Inicialitza el *mutex* creat.
pthread_attr_setstacksize	Fixa l'espai de pila destinat a un fil d'execució.
Gestió de la memòria	
mlock mlockall	Bloqueja part o tota la memòria virtual usada pel procés a RAM, de tal manera que no estigui disponible per guardar a disc (*swapping*).
shm_open	Obtenció d'un espai de memòria compartida.
shm_unlink	Elimina la referència a un espai de memòria compartida.
Rellotge	
clock_getres	Retorna la resolució d'un temporitzador donat.
clock_gettime	Obté el temps del sistema.
clock_nanosleep	Suspèn l'execució del fil fins un període de temps determinat.

4.10.2 Planificació per prioritats

El primer pas per la implementació d'una aplicació en temps real és la selecció d'una política de planificació en temps real i l'establiment de les prioritats. Això es pot fer a nivell de procés (*sched_setparam*) o a nivell de fil d'execució (*pthread_setsched*). Ambdues crides utilitzen els següents paràmetres:

- Tipus de planificació: SCHED_FIFO o SCHED_RR per a polítiques amb prioritat estàtica.
- Prioritat: de 1 a 99 per a tasques en temps real.

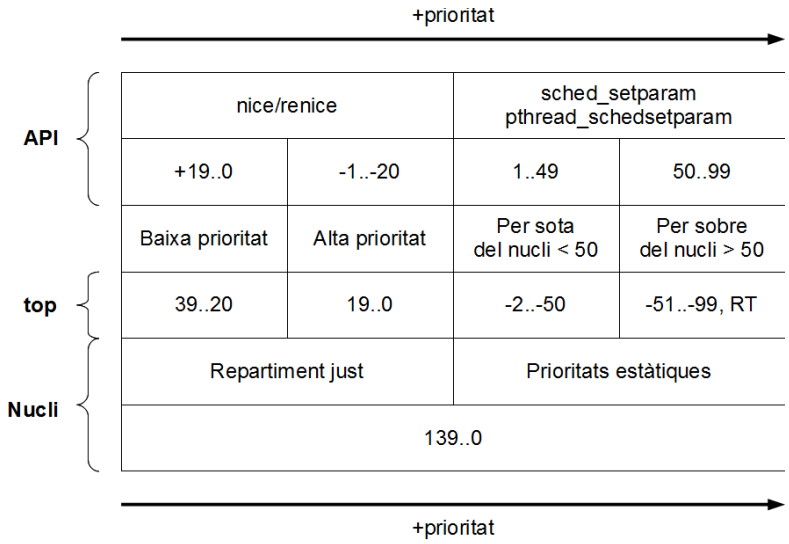
Figura 25: Codi d'exemple de prioritització d'un fil d'execució

```
/* Política de planificació RT SCHED_FIFO amb prioritat 51 */

/* En el nucli del mainline si utilitza execució d'interrupcions */
/* en context procés se li assignen una prioritat 50. Per tant */
/* en aquest cas s'esta prioritzant la tasca per sobre de l'execució */
/* del codi de les interrupcions pel que cal anar amb compte amb el */
/* temps de CPU que es consumeix i de no apropiar-se'l indefinidament */
/* ja que el sistema quedaria sense cap tipus de resposta a estímuls externs */

struct sched_param param;
```

```
param.sched_priority = 41;
pthread_setschedparam(pthread_self(), SCHED_FIFO, &param);
```

En la següent figura es pot veure una taula de les diferents numeracions utilitzades per definir els mateixos nivells de prioritat a diferents contexts.

Figura 26: Mapa de prioritats segons APIs, aplicació top i nucli



4.10.3 Gestió de la memòria

Linux utilitza la unitat de gestió de memòria (MMU) per proporcionar un espai exclusiu per a cada procés i major al total disponible en el sistema, mitjançant la paginació i l'intercanvi de pàgines entre RAM i disc. Un dels efectes secundaris, tanmateix, és la introducció ocasional de temps d'espera notables en la recuperació d'una pàgina de memòria de disc. La duració d'aquest temps, a més a més, és impredictible ja que es veu sotmesa de igual manera a la planificació en l'accés a disc.

Per tal d'evitar que una tasca amb requeriments de temps real es vegi penalitzada per aquest mecanisme cal instruir al sistema operatiu perque reservi la memòria utilitzada de manera exclusiva per a la tasca (*mlock*/*mlockall*). Cal tenir en compte, també, que la quantitat de memòria que es pot reservar per una aplicació esta limitada per la variable RLOCK_MEMLIMIT que es pot establir mitjançant la comanda de línia *ulimit*.

Per altra banda, tenint en compte que al reservar un espai de memòria (*malloc* i *stack*) aquest no és assignat fins al moment de la seva utilització és convenient, també, forzar l'accés al mateix abans d'iniciar les operacions en temps real, de manera que no es produeixi una major latència en el seu primer ús.

Figura 27: Codi d'exemple de modificació del límit de reservació de memòria

```
# Establiment del límit de memòria des de la línia de comandes
sudo ulimit -l <quantitat_màxima>
```

Figura 28: Codi d'exemple de reservació i reclamació de memòria

```
/* Reservació de tot l'espai de memòria utilitzat pel programa: present i futur */
```

```
mlockall(MCL_CURRENT | MCL_FUTURE);
/* Reclamació d'un espai de pila */
stack_prefault();
/* Reclamació d'un espai de memòria dinàmica */
char* buffer = malloc(MAX_ALLOC_MEM);
memset(&buffer, 0, MAX_ALLOC_MEM);
free(buffer);
/* Rutina de reclamació d'espai de pila */
void stack_prefault() {
      unsigned char dummy[MAX_SAFE_STACK];
      memset(&dummy, 0, MAX_SAFE_STACK);
}
```

4.10.4 Sincronització

La complexitat d'una aplicació fa que, de vegades, sigui convenient la utilització de diversos fils o processos d'execució. La comunicació entre fils d'un mateix procés es sol establir mitjançant l'espai de memòria compartit. Pel que fa als processos, per altra banda, aquesta es pot realitzar bé sigui a través de missatges (IPC) o canals de comunicació (sockets), tot i que en aplicacions de temps real l més habitual és utilitzar un segment de memòria compartit, mecanisme amb una menor sobrecarrega i un major determinisme. Tant en un cas com en un altre cal protegir els accessos a les estructures de dades de les condicions de carrera.

En aplicacions de temps reals, més concretament, es recomanable l'ús de *mutexes* amb herència de prioritats. D'aquesta manera s'evita la inversió de prioritats tot i que cal tenir en compte que no s'eviten les abraçades mortals i que un mal disseny dels mateixos pot portar a un empitjorament del comportament en temps real en cadenes de reservació de recursos.

Figura 29: Codi d'exemple de creació de *mutex* amb herència de prioritats

```
/* Creació de mutex amb herència de prioritats */
pthread_mutexattr_t pth_attr;
pthread_mutexattr_init(&pth_attr);
pthread_mutexattr_setprotocol(pth_attr, PTHREAD_PRIO_INHERIT);
pthread_mutex_t pi_mutex;
pthread_mutex_init(&pi_mutex, pth_attr);
```

4.10.5 Creació prèvia de tasques

Degut a que la creació de processos o fils d'execució sol provocar fallades de pàgina i depèn d'operacions relativament costoses és convenient crear totes les tasques necessàries abans d'iniciar l'execució del codi amb necessitats de temps real. Paral·lelament, i tenint en compte que se'n reservarà la memòria de manera exclusiva, pot ser convenient reduir la quantitat de pila utilitzada pels fils creats, per defecte 8Mb.

Figura 30: Codi d'exemple de limitació de l'espai de pila

```
/* Limitació de l'espai de pila del fil d'execució */
```

```
pthread_attr_t small;
pthread_attr_init(&small);
pthread_attr_setstacksize(small, PTHREAD_STACK_MIN + 0x2000);
pthread_create(handler, small, entry, NULL);
```

4.10.6 Lectura i escriptura a disc

Les operacions de lectura i escriptura a disc són per naturalesa de caràcter no determinista. Entre altres, al utilitzar memòria intermèdia poden provocar fallades de pàgina, depenen del estat del dispositiu d'emmagatzemament en el moment de la realització i, a més, estan sotmeses a la ordenació instrumentada pel planificador de E/S.

Per aquesta raó les aplicacions en temps real solen utilitzar processos independents per a la realització d'aquestes operacions de manera que, mitjançant una memòria intermèdia, absorbeixen els possibles pics de degradació de rendiment. De totes maneres cal tenir en compte les dificultats que implica aquesta implementació, principalment: possibilitat de pèrdua d'informació davant d'una fallada del sistema i possibilitat d'afegir latències addicionals o inversió de prioritats en la protecció de l'accés a la memòria intermèdia.

4.10.7 Temporitzadors

Linux permet dos tipus de rellotges com a base de les temporitzacions: CLOCK_MONOTONIC, que denota el temps ocorregut des de un instant arbitrari en el temps i CLOCK_REALTIME, que denota el temps la data del sistema. Per altra banda, suporta dos tipus de temporitzadors: els relatius i els absoluts. De cara al desenvolupament d'aplicacions en temps real és important utilitzar CLOCK_MONOTONIC i temporitzadors absoluts.

Primer, perquè CLOCK_REALTIME pot ser modificat per aplicacions d'usuari i, segon, perquè l'ús de temporitzadors relatius pot produir derives si al utilitzar el temps actual per establir un instant en el futur i abans de calcular el temps necessari a transcorre es produeix la preempció de la tasca. A més, es recomanable utilitzar temporitzadors d'alta resolució per evitar el *jitter* produït per la desincronització de la programació del event i la falta de resolució del rellotge del sistema.

Figura 31: Codi d'exemple de creació d'una temporització absoluta

```
/* Obtenció del temps actual en una base absoluta */
clock_gettime(CLOCK_MONOTONIC, temps);
/* Càlcul de l'instant en el futur sobre una base absoluta */
instant_futur = temps + periode.
/* Programació del temporitzador segons un temps absolut */
clock_nanosleep(CLOCK_MONOTONIC, instant_futur)
```

4.11 Influència del maquinari

Existeixen diferents aspectes en el maquinari, principalment dissenyats per millorar la eficiència o eficàcia del mateix, que poden influir de manera negativa en la latència d'atenció dels esdeveniments. En aquest apartat se'n presenta un recull d'alguns d'aquests aspectes recollits, alguns d'ells, del portal del nucli de Linux dedicat al temps real [BOH09].

4.11.1 DMA

Direct Memory Access (DMA) és un dispositiu de maquinari que es troba en els sistemes de computació moderns utilitzat per evitar que el processador realitzi les taques repetitives de copia de blocs dades d'un dispositiu a un altre. Així doncs, quan un programari requereix de copiar una certa quantitat de dades pot programar aquest dispositiu per realitzar aquesta tasca deixant el processador lliure per altres operacions. Per contrapartida, l'ús del bus de dades, utilitzat per l'intercanvi d'informació entre els diversos dispositius, passa a estar competit entre la CPU i els controladors de DMA introduint una latència inesperada en el seu accés.[51]

4.11.2 Bus locking

Bus locking és una tècnica de maquinari utilitzada en sistemes multiprocessador (SMP) per garantir l'atomicitat de les operacions emprades en la implementació dels *locks*. Aquesta tècnica consisteix en inserir una senyal de control (LOCK) de tal manera que en estar activa cap altre processador pot accedir al bus de dades introduir una latència addicional en la resta de processadors que requereixin d'accedir al bus.

4.11.3 Modalitats d'estalvi d'energia

Els sistemes moderns permeten la operació del sistema en diferents modalitats de consum que involucren canvis en l'estat dels dispositius com poden ser el processador i les memòries principals i auxiliars. Tanmateix, tot i la utilitat d'aquestes modalitats, cal tenir en compte, sobretot pel que fa al temps real, que les transicions d'un estat a un altre comporten un cost temporal. Així doncs, si en produir-se un esdeveniment determinat, el processador és troba en algun estat d'estalvi d'energia, apareix una latència addicional en la realització d'aquesta transició.

En aquest aspecte, Linux contempla dues maneres per evitar que el processador entri en estats d'estalvi d'energia: arrancar el sistema amb les modalitats desactivades,[52] o bé desactivar-les de manera selectiva mitjançant la interfície de control de qualitat de la gestió de energia (PM QOS). En el primer cas s'inhibeix la utilització de modes d'estalvi d'energia en el processador durant tota l'execució de la sessió de sistema operatiu. En el segon cas a través del fitxer */dev/cpu_dma_latency* es pot establir temporalment el temps màxim desitjat de retorn d'un estat d'estalvi d'energia. En cas de què aquest temps valgui cero, el processador no abandonarà mai l'estat d'execució normal.

En la següent figura podem veure un exemple d'utilització de la interfície PM QOS per aturar i tornar a activar les modalitats d'estalvi d'energia del processador[53].

Figura 32: Codi d'exemple de inhibició temporal de les modalitats d'estalvi d'energia del processador.

```
/* descriptor del fitxer d'interfície PM QOS */
static int pm_qos_fd = -1;

/* rutina per inhibir modes d'estalvi */
void start_low_latency(void)
{
 s32_t target = 0;
```

51 Un exemple d'aquest ús és troba en aquells dispositius de maquinari que requereixen de transferir un gran volum d'informació, com poden ser controladors de disc (SATA/PATA/SCSI) o, fins i tot, alguns adaptadors de xarxa. En aquests casos, per tal d'evitar la competició del bus, hi ha la possibilitat de modificar o configurar el controlador del dispositiu perquè faci un ús limitat del DMA, tot i que a costa d'una degradació del seu rendiment.

52 Cal carregar el nucli amb la línia de comanda *processor.max_cstates=1* i *idle=poll*. [KER13]

53 Adaptació d'el codi inclòs en un article divulgatiu publicat al portal de Red Hat [RHT12]

```
 if (pm_qos_fd >= 0)
      return;
 /* obrir la interfície */
 pm_qos_fd = open("/dev/cpu_dma_latency", O_RDWR);
 if (pm_qos_fd < 0) {
      fprintf(stderr, "Failed to open PM QOS file: %s",
      strerror(errno));
      exit(errno);
 }
 /* L'escriptura d'un cero inhibeix els mode d'estalvi dels processadors */
 write(pm_qos_fd, &amp;target, sizeof(target));
 /* El valor es valid fins el tancament del fitxer/interfície PM QOS */
}

/* rutina de la finalització de la inhibició */
void stop_low_latency(void)
{
 /* Al tancar el fitxer/interfície finalitza condició imposada */
 /* per tant, tornen a estar habilitats els modes d'estalvi d'energia */
 if (pm_qos_fd >= 0)
      close(pm_qos_fd);
}
```

4.11.4 Interrupcions SMI

Les interrupcions SMI (System Management Interrupts) són interrupcions generades pel dispositius de de gestió d'energia i supervisió del sistema. Aquestes interrupcions, per un costat, tenen la prioritat més alta i, per altre costat, no són interceptables ja que utilitzen una línia d'interrupció dedicada que provoca l'entrada del processador a un mode especial i el salt a una secció prefixada de codi en la BIOS[54] només accessible en aquest mode. És per això que aquestes interrupcions no són visibles per al sistema i aquest no en pot exercir cap control. Per altra banda, cal anar amb cura amb desactivar-les ja algunes de les funcions que gestionen estan relacionades amb aspectes vitals del sistema com pot ser el control de temperatura del propi processador.

Així doncs, en arribar una interrupció SMI, s'atura el processador en el que esta associada aquesta línia d'interrupció i s'executa el seu gestor introduint una latència addicional la longitud de la qual depèn de la implementació realitzada pel fabricant del maquinari. En aquest aspecte, l'únic que es pot fer es avaluar el maquinari pel que fa a aquesta faceta i demanar al fabricant que millori la implementació dels gestors SMI.[55]

Aquest problema pot afectar a altres processadors d'un sistema multiprocessador (SMP) si en el moment d'arribada d'una interrupció SMI el processador associat posseïx algun *lock* sobre una estructura competida pels altres o simplement és produeix un *bus lock*.

54 S'anomena BIOS (Basic Input/Output System) al *firmware* utilitzat per a les tasques de baix nivell del maquinari com poden ser l'arrancada del sistema, la configuració primera dels dispositius, la supervisió dels elements de maquinari o la gestió dels modes d'estalvi d'energia.

55 Per a la detecció de latències introduïdes pels gestors SMI es pot utilitzar l'aplicació *hwlatdetect* inclosa en la *suite rt-test*. [WIL12]

4.11.5 Consola VGA

D'acord amb algunes referències, [SAL10] i [BOH09], la utilització de la consola VGA, les crides a la qual no permet la seva preempció, pot arribar a causar latències de l'ordre de les centenars de microsegons. En aquest cas, es recomana utilitzar una consola serial, una sessió telnet[56] o SSH,[57] o, alternativament, una interfície gràfica basada en X Window[58] ja que qualsevol d'aquestes aplicacions si permeten la seva preempció. Addicionalment, es suggereix, també, limitar el nombre de missatges del sistema.[59]

4.12 Els controladors de dispositius

Finalment, per assolir uns requeriments de temps real donats no només és necessari que el sistema respongui amb cert grau de determinisme i que l'aplicació estigui ben dissenyada per treure'n profit. A més, cal que la implementació dels controladors de dispositius, normalment realitzada pels fabricants dels mateixos, sigui la adequada per aquestes necessitats.

Així doncs, cal que permeti la seva preempció en totes aquelles zones de la seva àrea de codi on el bloqueig de recursos no sigui totalment imprescindible i que realitzi un ús eficient dels recursos de maquinari. De fet, en aplicacions amb requisits de baixa latència és habitual l'ús de controladors de maquinari especialitzats.[60]

56 Telnet és un protocol de xarxa que permet l'establiment remot d'una consola de comandes.
57 Secure Shell (SSH) és un protocol de xarxa encriptat que permet l'establiment remot d'una consola de comandes.
58 El sistema X Window, també conegut com X11 o X, és un programari de sistema i protocol de xarxa que ofereixen la base per al desenvolupament d'interfícies gràfiques utilitzat en la majoria d'escriptoris de les distribucions de Linux com poden ser Xfce, GNOME o KDE.
59 La reducció de missatges de sistema és pot assolir amb carregant-lo amb la línia de comanda *quiet*. [KER13]
60 A tall d'exemple, les aplicacions de manipulació d'àudio en temps real, com és el cas de Cubase, utilitzen una tecnologia de controladors especialitzada en la millora de la latència anomenada Audio Stream Input/Output (ASIO). [SOS99]

5 Avaluació del temps real a Linux

5.1 Introducció

Un cop analitzades les dificultats per l'assoliment del temps real en el sistema operatiu Linux i exposades les diferents solucions emprades en la millora d'aquest objectiu -principalment major àrea de preempció- i compromisos adquirits en l'aplicació de les mateixes -davallada de rendiment- en aquest capítol es realitza l'avaluació en l'assoliment d'aquest objectiu.

Per això, en primer lloc, es defineix que és el *benchmark*, la tècnica que escollida per realitzar l'avaluació així com les motivacions darrera la seva elecció. Tot seguit, es passa a descriure l'entorn en el que es realitza l'avaluació -el maquinari i sistema operatiu- així com els subjectes sotmesos a prova, el rendiment i la latència de quatre configuracions diferents de nucli que es diferencien principalment per la modalitat de preempció utilitzada.

A continuació, és presenta presenta el marc metodològic emprat on s'expliquen els passos seguits en el disseny i validació de les proves d'avaluació, les mètriques i estadístics escollits per la representació de les propietats avaluades així com la justificació de cadascun d'ells. A més, es descriu el procés utilitzat en l'elaboració de les càrregues a les que s'ha sotmès l'entorn avaluat i es presenten les característiques que se'n busquen.

Finalment, s'exposa un resum dels resultats obtinguts, s'analitzen els trets característics més rellevants i es realitza l'anàlisi dels valors obtinguts en referència a les propietats avaluades.

5.2 Benchmark

Benchmark és la tècnica de mesura de certes propietats del programari mitjançant l'execució d'una una sèrie d'aplicacions que sotmeten el sistema a unes probes determinades i en mesuren la eficiència amb que aquest les resol. La seva finalitat tant és la de realitzar una avaluació preliminar de la capacitat d'un sistema, cas del projecte, com la de verificar que un sistema ja desenvolupat compleix amb uns requisits donats.

Les raons de la utilització d'aquesta tècnica en comptes de l'anàlisi de camins, alternativa formal consistent en el modelatge teòric del cost dels camins de codi, són dos: la dificultat de modelatge del maquinari modern i la complexitat d'un sistema operatiu de propòsit general de les dimensions de Linux.[61]

Per una banda, la dificultat de modelatge del maquinari modern bé donada per les diferents solucions arquitectòniques adoptades per obtenir un major rendiment d'una menor quantitat de recursos com poden ser: el *pipelining*[62], la virtualització en diferents entitats lògiques d'un mateix nucli físic (*hyperthreading*)[63], els diferents nivells i propòsits de memòria cau[64], els components

61 En les seves últimes versions a l'entorn de 15 milions de línies de codi (LOC). [LEE12]

62 Solució arquitectònica consistent en la divisió de l'execució d'una instrucció en diferents etapes de manera que es puguin executar de manera paral·lela diferents instruccions consecutives que no requereixin els mateixos recursos de maquinari.

63 Solució arquitectònica consistent en la execució de diferents fils de codi en una única unitat de processament de manera que per un fil s'aprofitin aquelles etapes del *pipeline* no utilitzades per un altre fil momentàniament aturat en alguna de les altres etapes.

64 Els processadors moderns utilitzen fins a tres nivells de memòria cau i arquitectures *modified harvard* en les que s'empra aquesta memòria per emmagatzemar codi i dades connectats per dos camins de dades diferents al processador permetent la seva lectura

d'accés directe al bus de memòria (DMA)[65] o els sistemes multiprocessador (SMP), entre altres. Totes aquestes solucions obtenen un major rendiment d'una menor quantitat de recursos tot i que a costa de una pèrdua notable de determinisme.

Per altra banda, el sistema operatiu, a mida que avança el seu desenvolupament, Linux incorpora cada cop més arquitectures, propòsits i funcionalitats diferents fent créixer el codi fins un punt en què l'anàlisi de camins no és només notablement complex si no que, a més, té un cost molt elevat.

Per contrapartida, la utilització del *benchmark* per a l'avaluació del temps real a Linux ens ofereix un procediment relativament senzill, de complexitat assumible, que és pot automatitzar i que pot ser aplicat per altres persones amb certa facilitat. Tanmateix, cal senyalar que aquest tipus de tècnica no està normalment acceptada com a mètode de certificació vàlid en sistemes crítics.

Taula 6: Diferències principals entre el benchmarking i l'anàlisi de camins

Benchmark	Anàlisi de camins
Tècnica empírica.	Tècnica analítica.
Relativament senzilla.	Altament complexa.
Poc costosa.	Molt costosa.
És pot automatitzar.	No és pot automatitzar.
És fàcilment repetible o verificable per altres persones.	No és fàcilment repetible ni verificable per altres persones.
Poc reconegut com a sistema de certificació vàlid.	Bastament reconegut com a sistema de certificació vàlid.

5.3 Entorn d'avaluació

5.3.1 Plataforma de maquinari

L'avaluació de les capacitats de temps real s'efectua sobre un *Netbook DELL™ Inspiron™ 910* amb les següents característiques:

- Intel ATOM® N270™ @ 1,6Ghz [INT08]
- Cache L1 32-KB i L2 512-KB (*write-back*).
- 1GB de SDRAM DDR2 @ 553Mhz.
- Unitat d'estat solid de 32GB.

La tria d'aquesta plataforma es basa, per un costat, en el fet que disposa d'unes característiques similars a les d'un sistema encastat: processador de baix consum sense ventilació (*fanless*), potència de còmput mitjana i ús d'una unitat d'estat sòlid, entre altres. Per altra costat, el processador emprat utilitza una arquitectura x86, la més estesa i suportada per Linux i, a més, com a *neetbook*, disposa de les facilitats típiques d'un ordinador de sobretaula (interfície d'usuari, comunicacions, capacitat d'emmagatzemament notable...)

5.3.2 Nuclis de Linux

Partint de la distribució més actual de la branca estable de Debian (6.0 o *Squeeze*), una de les més esteses[66], s'obté la versió de nucli més contemporània suportada pel pedaç RT (3.2.32) i es

simultània, aspecte clau en la eficiència de tècniques com el *pipelining*.

65 Direct Memory Acces (DMA) és un component de maquinari emprat en la transferència de dades entre diferents dispositius connectats al bus de dades sense la intervenció de la unitat central de processament (CPU).

66 Segons DistroWatch, pàgina web que inclou un rànquing de popularitat de les distribucions Linux basat en el nombre de visites rebudes per les seves diferents pàgines web, de les cinc distribucions més populars (Mint, Mageia, Ubuntu, Fedora i Debian) tres són basades en Debian (Mint, Ubuntu i la pròpia Debian).[DIS13]

configuren i compilen quatre nuclis distints (K1, K2, K3 i K4), cadascun amb un nivell de preempció major i l'últim amb el pedaç RT, amb l'objectiu de comprovar-ne l'impacte en la latència i el rendiment de les diferents alternatives de millora del temps real.[67]

5.3.3 Opcions de compilació

Per cadascun dels nuclis es seleccionen una sèrie d'opcions de configuració partint de la estàndard de la distribució utilitzada. La majoria d'aquestes opcions són iguals per a tots els nuclis per, d'aquesta manera, disposar d'un marc comú de codi binari que permeti aïllar l'impacte dels trets diferencials puntuals.

A continuació es presenten les opcions triades per a cada nucli distintament configurat en format de taula i agrupades segons el menú en el que es troben en l'arbre de configuració del nucli.

5.3.3.1 *General setup*

Sufix de la versió de nucli i opcions del mecanisme de sincronització RCU.

Taula 7: Opcions de compilació *General Setup*

Opció de configuració	K1	K2	K3	K4	Justificació
LOCAL_VERSION Afegeix una cadena de text a la versió de nucli (uname -r).	-no preempt	-voluntary	-preempt	-rt	Distinguir els diferents nuclis assajats.
TINY_PREEMPT_RCU Implementació del mecanisme RCU amb preempció.	SI				Permetre la preempció de cicles RCU (Read-copy-update).
RCU_BOOST Elevació de prioritat de les tasques lectores RCU.	SI				Prioritzar la finalització de lectures RCU d'estructures pendents d'actualització per avançar la mateixa.

5.3.3.2 *Processor type and features*

Funcionalitats més rellevants per l'assoliment del temps real, de manera notable el nivell de preempció del nucli i els temporitzadors d'alta resolució.

Taula 8: Opcions de compilació *Processor type and features*

Opció de configuració	K1	K2	K3	K4	Justificació
NO_HZ Sistema sense rellotge	NO				Evitar jitter provocat per càlcul i programació del rellotge.
HIGH_RES_TIMER Temporitzadors d'alta resolució (< jiffie)	SI				Evitar latència provocada per falta de precisió i resolució.[68]
SMP Suport a multiprocessament simètric	NO				Minimitzar la complexitat del sistema analitzat. Aïllar els efectes dels canvis introduïts.
MATOM Processador Intel Atom®	SI				Utilització de les optimitzacions específiques de l'arquitectura.[69]

67 A l'annex s'inclouen els passos realitzats per la descarrega, aplicació del pedaç, configuració i compilació.
68 Per verificar el suport d'aquest subsistema de temporització en el maquinari utilitzat es realitzen les probes indicades a [ELN11]
69 No estaran, però, totes les optimitzacions disponibles ja que la distribució utilitzada incorpora una versió de compilador (4.4) que encara no les suporta (4.5) i, a més, a data de la realització del projecte, no existien *backports* de la mateixa.

Opció de configuració	K1	K2	K3	K4	Justificació
HPET_TIMER Suport de High Precission Event Timer (HPET)[INT04]	SI				Incorporació de suport de dispositius de rellotge de alta precisió.[70]
IRQ_TIME_ACCOUTING Opcions de comptabilitat de temps de servei d'interrupció de precisió.	SI				Millora de la precisió en la comptabilitat dels temps de servei d'interrupcions.[PAL10] Tanmateix, cal tenir en compte que introdueix una certa sobrecarrega.
PREEMPT Nivell de preempció del nucli.	PREEMPT_NONE	PREEMPT_VOLUNTARY	PREEMPT / PREEMPT_LL	PREEMPT_RT_FULL	Selecció d'un nivell de preempció de nucli diferent: enlloc del nucli, en punts voluntaris, arreu excepte seccions crítiques i també a la majoria de seccions crítiques a més d'execució d'interrupcions en context de procés[71].
HZ Freqüència del rellotge del sistema (100, 250, 300 i 1000 Hz)	HZ_300				Augment de la resolució de treball del sistema a través de l'augment de la resolució del rellotge del sistema (300Hz en compte de 250Hz, per defecte).[72]

5.3.3.3 *Block layer / IO Schedulers*

Configuració del planificador d'entrada / sortida.

Taula 9: Opcions de compilació *Block layer / IO Schedulers*

Opció de configuració	K1	K2	K3	K4	Justificació
DEFAULT Permet escollir planificador E/S (noop, deadline o cfq)	DEFAULT_NOOP				Habilitació del planificador NOOP per evitar la planificació E/S en la unitat d'estat sòlid utilitzada.[73]

5.3.3.4 *Kernel hacking*

Opcions de traça per avaluar la latència en les diverses configuracions. Algunes només disponibles en el nucli amb el pedaç RT.

Taula 10: Opcions de compilació *Kernel hacking / Tracers*

Opció de configuració	K1	K2	K3	K4	Justificació
LATENCY_TOP Habilitació d'eines per la monitorització de latències.	SI				Anàlisi interactiu de latències de les tasques en execució.
SCHEDSTATS Recol·lecció d'estadístiques de planificació	SI				Anàlisi d'utilització de processador i espera en cua de les tasques.
FUNCTION_TRACE Traça de les execucions de les funcions del nucli.	SI				Anàlisi de l'origen d'una latència determinada.

70 Es verifica la presència d'aquest controlador mitjançant el pseudo-sistema d'arxius *sysfs*. En Debian 6.0 Squeeze: /sys/devices/system/clocksource/clocksource0/available_clocksource.

71 Aquesta opció també esta present de manera opcional a través de la línia de comandes del nucli, utilitzada en l'arranc, *threadirqs*. [KER13]

72 No s'escull la màxima freqüència per evitar la sobrecarrega excessiva del sistema tenint en compte que no s'utilitzarà un processador d'alta potència.

73 No aporta cap benefici la reordenació de peticions en un dispositiu amb un temps d'accés uniforme a tot l'espai d'adreces.

Opció de configuració	K1	K2	K3	K4	Justificació
IRQSOFF_TRACER Registre de períodes amb interrupcions inhibides.	SI				Anàlisi del període més llarg amb interrupcions inhibides.
INTERRUPT_OFF_HIST Registre d'histogrames de períodes amb interrupcions inhibides.	-			SI	Anàlisi històric dels períodes amb interrupcions inhibides.
PREEMPT_TRACER Registre de períodes amb preempció inhibida.	-		SI		Anàlisi del període més llarg amb preempció inhibida.
PREEMPT_OFF_HIST Registre d'histogrames de períodes amb preempció inhibida.	-			SI	Anàlisi històric dels períodes amb preempció inhibida.
SCHED_TRACER Registre de latència de planificació de la tasca de major prioritat.	SI				Anàlisi de la major latència de planificació de la tasca de major prioritat.
WAKE_UP_LATENCY_HIST Registre d'histogrames de latència de planificació de la tasca de major prioritat.	-			SI	Anàlisi històric de la latència de planificació de la tasca de major prioritat.
MISSED_TIMER_ OFFSETS_HIST Registre d'histogrames de latència d'atenció a un temporitzador.	-			SI	Anàlisi històric conjunt de la latència de planificació i d'atenció del temporitzador associat a la tasca de major prioritat.

5.4 Metodologia

Amb l'objectiu de dissenyar un experiment que permeti avaluar la capacitat de respondre en temps real del sistema operatiu Linux així com la davallada de rendiment que poden suposar les diferents tècniques emprades per assolir aquesta propietat es planifiquen les següents etapes:

1. Anàlisi i selecció de les propietats de l'experiment.
2. Selecció de mètriques i descriptors estadístics a registrar durant l'experiment.
3. Avaluació, selecció i adaptació d'eines per al registre de mètriques i generació de càrregues.
4. Validació i ajust de càrregues.
5. Disseny final de l'experiment.
6. Execució de l'experiment i registre de dades.
7. Tractament, exposició i anàlisis dels resultats obtinguts.

5.5 Propietats de l'experiment

La capacitat d'assoliment dels terminis temporals en l'execució d'una tasca en un sistema operatiu de propòsit general com és Linux, i com s'ha pogut constatar en l'anterior capítol, és producte, pel que fa al mateix, de la successió d'esdeveniments no necessàriament lligats a les tasques de temps real i els camins de codi en el nucli que aquests fan recórrer. És a dir, que depèn en gran mesura de

quines operacions estava realitzant el sistema operatiu en el moment en que és produeix l'esdeveniment el termini del qual cal garantir.

Així doncs, un sistema en repòs, en el que els camins de codi del nucli romanen pràcticament latents, queda clar que no és un bon entorn per mesurar, de manera realista, les capacitats de resposta del mateix i que cal garantir, per un banda, un nivell notable i alhora repartit de càrrega en els diferents subsistemes del nucli i, per una altre banda, l'execució de l'experiment durant un període de temps suficientment llarg per a augmentar la probabilitat de coincidència de les distintes àrees de codi del nucli amb l'aparició d'una sèrie d'esdeveniments donats, entre ells aquells que requereixen d'una resposta en temps real.

Per altra banda, tenint en compte la gran diversitat d'aplicacions i maquinaris diferents en els que el temps real és aplicable i els diferents contexts casuístics d'execució que això pot provocar, cal garantir que l'experiment és prou ampli i alhora genèric així com que és fàcilment reproduïble o comparable .

Amb aquestes característiques com objectiu de disseny és plantegen certes propietats i s'estableixen diferents medis per garantir-les tal com s'exposa en la següent taula a mode de resum.

Taula 11 Propietats del experiment i medis per assolir-les

Propietat	Medis
Sistema estressat de manera notable i repartida.	- Aplicació de càrregues que exercitin els principals subsistemes del nucli i els diferents colls d'ampolla del maquinari.
Freqüència dels esdeveniments suficientment alta i amplada de l'espectre suficientment amplia.	- Establiment d'un rang significatiu de freqüències dels esdeveniments a partir d'una freqüència mínimament alta.
Duració del test suficientment llarga.	- Realització de tests de llarga durada.
Extrapolable a diversos sistemes o aplicacions.	- Disseny de càrregues que contemplin usos habituals. - Caracterització de les càrregues segons ús de les primitives del sistema. - Realització del experiment sense maquinari extern. - Descripció detallada de la configuració utilitzada.

5.5.1 Nivell d'estrès

Per mirar de garantir que el sistema està sotmès nivell d'estrès notable i repartit de tal manera que cap subsistema acapari l'ús del processador desplaçant a la resta és defineixen tres àrees principal de càrrega associades als colls d'ampolla habituals en un sistema: CPU, memòria principal (RAM), memòria secundària (disc) i xarxa.

A partir d'aquestes àrees de càrrega s'estableix la necessitat de seleccionar una sèrie d'aplicacions que produeixin estrès en les àrees esmentades i, a més, de caracteritzar els efectes de les mateixes en terminis de diferents mètriques de càrrega de manera que donin una visió precisa i objectiva del veritable efecte que produeixen sobre el sistema i si aquest es prou notable i balancejat en relació amb la resta de càrregues. La selecció de les aplicacions utilitzades per a la implantació d'una càrrega en el sistema es descriu en les seccions posteriors.

5.5.2 Freqüència dels esdeveniments

Segons Emde de la Open Source Automation Development Lab (OSADL)[74], per al registre de la latència, és recomanable realitzar probes amb una freqüència suficientment alta entorn a la meitat

74 Associació dedicada a la promoció i suport del codi lliure en la industria de la automatització industrial.

del pitjor cas esperat [EMD09]. A més, tal com indica McGuire i Zhou [GUI05], l'elecció d'una freqüència donada pot donar resultats poc realistes pel que fa a altres freqüències. Com exemple, si s'escull una freqüència d'esdeveniments per sobre de les utilitzades en el sistema amb el propòsit d'obtenir un cas pitjor, pot ocórrer que és potencii la localitat de la tasca associada dificultant la seva expulsió de les diverses etapes de la jerarquia de memòria i obtenint, contra pronòstics, uns resultats millors dels que s'obtindran quan s'utilitzen freqüències més baixes.

Per establir una freqüència suficientment alta seguint el criteri de Emde s'utilitza, per un costat, els resultats obtinguts en experiments similars realitzats per la OSADL en arquitectures i revisions de nucli molt semblants[75] en els que s'obtenen una latència màxima de 95us i, per altre costat, unes primeres mesures en el maquinari i configuracions proposades en el que s'assoleix una latència màxima entorn als 200us. A partir d'aquests valors s'estableix una latència màxima de 500us per a la realització de les proves.

Finalment, per establir diferents freqüències d'esdeveniments es seleccionen períodes distints a partir de la freqüència màxima seleccionada de tal manera que aquests no siguin múltiples entre ells (500us, 1100us, 2300us, 5300us i 270000us) per a evitar la possible sincronització d'esdeveniments que podria deixar certa part del codi del nucli sota una baixa probabilitat de coincidència amb els esdeveniments utilitzats per mesurar la latència del sistema.

5.5.3 Duració de l'experiment

Segons experiments realitzats amb anterioritat [EMD09] [MOL05]existeixen diferències notables pel que fa a la latència segons la duració del test pel que és recomana realitzar probes del ordre de 10^9 mostres. Segons s'ha establert en l'anterior apartat si la freqüència màxima d'esdeveniments la latència dels quals es mesurarà és de 500us tenim que:

$$T_{experiment} \geq 10^9\, mostres \times 500 \cdot 10^{-6}\, segons/mostra \geq \frac{500 \cdot 10^3\, segons}{3600\, segons/hora \times 24 \text{hores}/dia} = 5.78\, dies$$

Malauradament, no és disposa del temps necessari per a satisfer aquest requisit ja que si tenim en compte que l'objectiu és repetir l'experiment per a quatre configuracions de nucli diferents requeriria més de tres setmanes de calendari.

Finalment, i d'acord amb les restriccions de calendari en el moment d'iniciar les proves, s'opta per establir una duració de 8 hores que per a una període d'esdeveniments màxim de 500us resulta en un màxim de:

$$T_{experiment} = \frac{8\, hores \times 3600\, segons/hora}{500 \cdot 10^{-6}\, segons/mostra}\, segons/mostra = 5.76 \cdot 10^7\, mostres$$

Cal, doncs, tenir en compte el període de temps durant el que s'han obtingut els resultats i el nombre de mostres recollides alhora de ficar en perspectiva els resultats trobats.

5.5.4 Utilitat de l'experiment

Es defineixen diferents medis pels quals es procura de garantir que l'experiment realitzat es extrapolable o interpretable fins una certa mesura a entorns diferents als utilitzats durant les probes. Sota aquest objectiu s'estableix que el disseny de les càrregues seleccionades haurà de contemplar, sempre que sigui possible, un patró d'utilització dels recursos el més similar possible als utilitzats en les aplicacions habituals i que aquestes càrregues estaran objectivament

75 Proves realitzades en un maquinari amb processador Intel(R) Atom(TM) CPU N270 a 1.60GHz, utilitzant *hyperthreading* -ús dels dos processadors virtual- i una distribució CentOS amb nucli 2.6.33 amb el pedaç RT aplicat.

caracteritzades de manera que es pugui compar el seu efecte amb altres de similars en entorns diferents. Ambdós aspectes s'exploren amb més detall en les properes seccions.

Per altra banda, i amb l'objectiu de disposar d'unes proves que es puguin reproduir amb facilitat en altres entorns, s'estableix, també, que les proves realitzades no requereixin maquinari extern així com que es realitzi una descripció detallada de la configuració utilitzada de maquinari, que es pot trobar a l'annex d'aquesta memòria.

5.6 Mètriques i descriptors estadístics

Durant el disseny de l'experiment és plantegen tres dominis fonamentals que guien la selecció de mètriques i descriptors estadístics a utilitzar: característiques a mesurar, resultats a obtindre i coneixement que se'n pot extreure dels mateixos i que s'exposen en la següent taula a mode de resum.

Taula 12 Resum de característiques mesurades i resultats obtinguts

Característiques a mesurar	Resultats a obtenir	Coneixement que se'n pot extreure
Temps real		
• Latència.	• Histograma. • Desviació típica. • Percentils alts.	• Capacitat del sistema en satisfer sistemes de temps real segons diferents requeriments de qualitat de servei (QoS).
Rendiment		
• Temps d'execució total. • Temps d'espera amb tasca apunt d'executar-se. • Temps d'espera amb tasca no apunt per executar-se.	• Mitja dels temps mesurats. • Proporció entre els temps mesurats per a un mateix nucli. • Proporció dels temps total mesurats entre cadascun dels quatre nuclis.	• Magnitud de la davallada de rendiment. • En comparació amb la capacitat del temps real, compromís que s'assumeix en la seva millora.

5.6.1 Mètriques de temps real

Per a la mesura de les capacitats de resposta en temps real s'utilitza la següent mètrica:

- Latència (microsegons): temps transcorregut des de l'aparició d'un esdeveniment fins l'inici de l'execució de la tasca de temps real associada al mateix. Aquest temps contempla doncs la suma de totes les latències tractades en l'anterior capítol.

5.6.2 Descriptors estadístics de temps real

Per a descriure el grau de capacitat d'aquesta resposta s'utilitzen principalment els següents indicadors o representacions:

- Histogrames (latència/nombre de repeticions): latència observada i el nombre de repeticions projectada en un eix x/y. Ofereix una representació gràfica de la distribució dels resultats obtinguts que permet fer-se una idea aproximada del determinisme (dispersió dels valors), comportament mitjà (àrea d'agrupació major dels valors) o pitjors resultats (valors més allunyats).
- Desviació típica o estàndard (microsegons)[76]: unitat de mesura de la dispersió dels valors -a major desviació, major dispersió- obtinguda de la arrel quadrada de la variància:

76 El valor de la desviació típica presenta per una distribució normal, present en la majoria de fenòmens naturals o variables producte de diversos processos independents, la següent població entorn de la mitja aproximadament: 68,26% en l'interval $[\bar{x}-\sigma, \bar{x}+\sigma]$, 95,44% en l'interval $[\bar{x}-2\sigma, \bar{x}+2\sigma]$ i 99,74% en l'interval $[\bar{x}-3\sigma, \bar{x}+3\sigma]$.

$$\sigma = \sqrt{1/(N-1)\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

On N: nombre de mostres[77], $x_{i:}$ valors de les mostres i $\overline{x:}$ la mitjana aritmètica.

Aquest estadístic s'utilitza per valorar el determinisme del sistema segons la dispersió dels valors obtinguts en l'experiment.

- Percentil (microsegons): indica el valor de latència per sota del qual un cert percentatge d'observacions roman. S'utilitzen més concretament els percentils 95%, 99,999%, 99,999999% i 100% que corresponen respectivament a 5 de cada 100, 1 de cada 10^5 i 10^7 i cap del total de mostres recollides de valors per sobre del valor de percentil donat.

5.6.3 Mètriques de rendiment

Per a la mesura de la davallada de rendiment en cadascun dels nuclis s'utilitzen principalment les següents mètriques:

- Proporció de treball penalitzat (tant per 1): proporció de temps d'execució en l'assoliment del temps real en comparació amb el millor resultat obtingut, calculat com:

  $$Penalized = T_{Execució} / T_{Execució\ Millor\ Resultat}$$

  Amb aquesta mètrica s'analitza la pèrdua total provocats tant pel cost d'implementació de les diferents solucions en l'assoliment del temps real -gestió més complexa- com la ralentització del treball provocada per arbitració dels recursos menys eficient -majors canvis de context, principalment.

- Proporció de temps en espera en cua del planificador (tant per 1): proporció del temps total d'execució que la tasca estant apunt per execució roman a l'espera en la cua del planificador, calculada com:

  $$Slowdown_{Sched} = (T_{Execució} + T_{EnCua}) / T_{Execució}$$

  Amb aquesta mètrica s'analitza per un costat la pèrdua d'eficiència del sistema deguda als possibles continus canvis de tasca produïts per una major reactivitat del sistema en intentar oferir una millor resposta al temps real així com el temps de treball útil en el que la tasca mesurada ha estat desplaçada del planificador per atendre a altres de més prioritaries.

- Proporció de temps en espera d'altres recursos (tant per 1): proporció del temps total d'execució que la tasca no esta apunt per execució perquè està esperant la finalització d'alguna operació, normalment accessos d'entrada / sortida, calculada com:

  $$Slowdown_{NoSched} = (T_{Execució} + T_{EnEspera}) / T_{Execució}$$

  Aquesta mètrica complementa la anterior quantificant quina part de l'espera és producte d'un efecte secundari de la contenció amb els diferents recursos -principalment camins E/S- i no pas per un canvi en la mida de gra de la planificació o la gestió de les prioritats.

5.6.4 Descriptors estadístics de rendiment

Per a descriure la davallada de rendiment produïda en el camí cap a l'assoliment d'una major resposta en temps real i per tal d'establir l'origen de la mateixa s'utilitzen els següents indicadors estadístics principalment:

77 En aquest cas s'utilitza la correció de Bessel -ponderació sobre N-1- emprada en inferències poblacionals

- Mitjana geomètrica (unitat segons mètrica): mitjana calculada amb l'arrel quadrada del multiplicatori dels valors ponderats. Presenta la característica que és menys sensible als valors extrems trobats en una distribució de valors que la mitjana aritmètica. És calcula de la següent manera:

$$Geomean = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

- Coeficient de correlació de Pearson (entre -1 i 1): índex de relació de quan lineament relacionades estan dues variables. Valors positius indiquen relacions directament proporcionals mentre que valors negatius indiquen relacions inversament proporcionals. Quan més proper està el valor absolut de l'index a 1 més nítida és la relació. Per contra quan més proper està de 0 més difusa és. És calcula de la següent manera:

$Pearson = \frac{\sigma_{xy}}{\sigma_x \times \sigma_y}$ on σ_{xy} correspon a la covariància i σ_x a la variància i és calculen com:

$\sigma_{xy} = n\sum_{i=1}^{n} x_i y_i \times \sum_{i=1}^{n} x_i \times \sum_{i=1}^{n} y_i$ i $\sigma_x = \sqrt{1/(N-1)\sum_{i=1}^{n}(x_i - \bar{x})^2}$, respectivament.

El coeficient mostra el grau de relació entre dues variables i tot i que no és condició suficient per garantir una relació causa-efecte en que pot senyalar la seva existència.

5.7 Eines de mesura

Seguint els objectius plantejats en els anteriors apartats pel que fa a propietats desitjades i mètriques seleccionades és descarreguen i avaluen diferents eines de mesura de latència i rendiment i generació de càrrega.

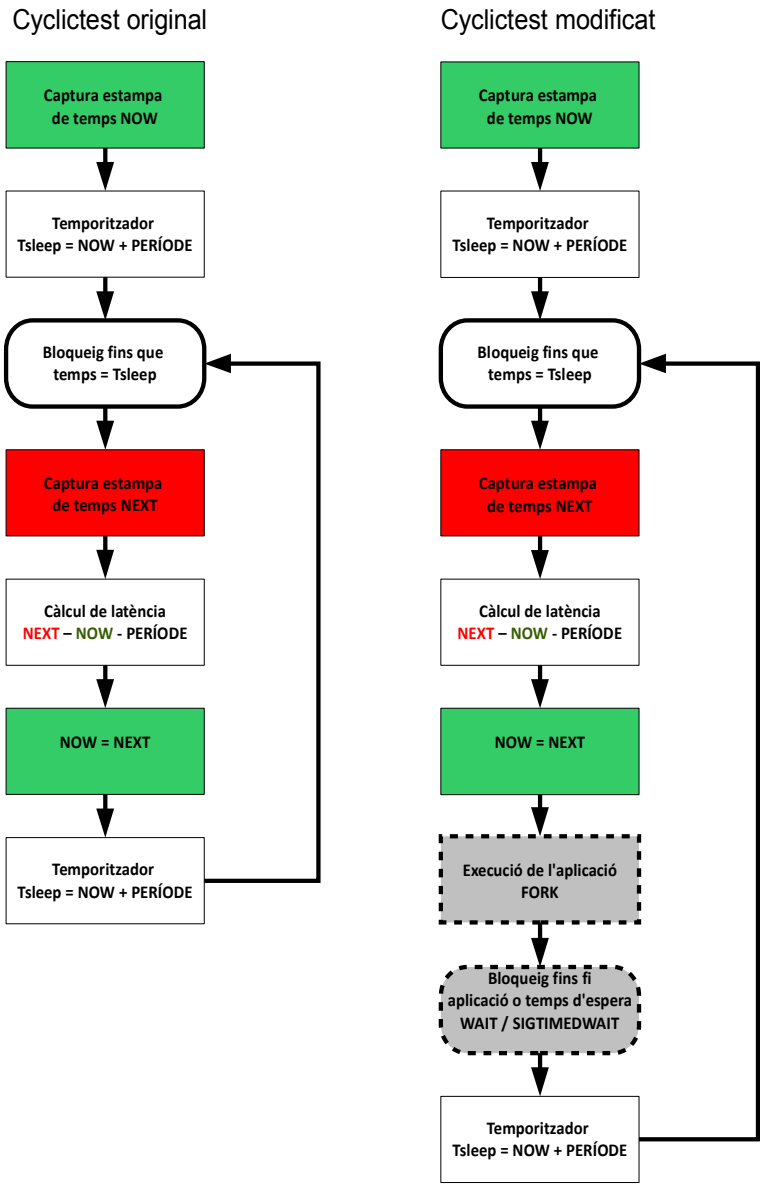
5.7.1 Mesura de la latència

L'eina escollida per a mesurar la capacitat del sistema per respondre en temps real és *Cyclictest*, una aplicació de registre de latència mitjançant la programació de temporitzadors del sistema. El principal criteri de selecció d'aquesta utilitat és el de no requerir maquinari extern especialitzat i, per tant, proporcionar un experiment fàcil de reproduir en altres entorns.

A més a més, per utilitzar aquesta eina per a generar càrregues periòdiques, és realitzen un parell de modificacions que permeten a la l'aplicació executar un programa qualsevol i, de manera opcional, fer-ho durant un temps limitat.

En la figura 33 es representa el principi de funcionament de l'aplicació així com de la modificació realitzada. Els codi complet de la modificació es pot trobar al apartat 8.3 de l'annex.

Figura 33: Principi de funcionament de *cyclictest* original i modificat



Aquesta implementació, però, presenta dues contrapartides:

- El temps de creació (fork) i destrucció dels processos (kill) són indeterminats i, segons diverses proves realitzades, pot arribar fins a l'ordre de les desenes de milisegons.
- Si l'aplicació s'executa amb prioritats de temps real i per tal d'evitar que la mateixa arribi a copsar el 100% dels recursos del sistema cal programar un temps de període inferior al pitjor cas d'execució.

$$Període \leq Pitjor_{Latència} + WCET_{CreacióProcés} + WCET_{Procés} + WECT_{DestrucciónProcés}$$

Com que aquests són valors que no es coneixen a priori se segueixen els següents passos per ajustar-los: per un costat, s'executa l'aplicació externa de manera independent; per altre costat,

s'executa *cyclictest* amb un període suficientment alt; finalment, s'agafen els pitjors temps i se'ls afegeix un marge considerable, entorn al 25% del total.

L'aplicació s'executa des de la línia de comanes i de les múltiples opcions que presenta s'utilitzen les següents en l'avaluació:

- -q : mode silenciós.
- -h999999 : registre d'histograma de fins a 999999 us.
- -m : bloqueig de les pàgines RAM per evitar *swapping*.
- -n : utilització de rellotges d'altra precisió (nanosleep).
- -pXX : execució de l'aplicació amb planificador SCHED_FIFO, prioritat XX.
- -iYYY : interval de programació dels temporitzadors.
- -x cmd : execució d'una aplicació externa (afegit).
- -u useconds : temps màxim d'execució de l'aplicació externa (afegit).

A més l'aplicació *cyclictest*, per defecte, utilitza la interfície de control de qualitat de la gestió de energia (PM QOS) esmentada a l'apartat 4.11.3 per a desactivar els modes d'estalvi d'energia eliminant d'aquesta manera la latència produïda en els canvis de modalitat de consum.

5.7.2 Mesura del rendiment

L'eina escollida per a mesurar el rendiment és la utilitat *time* de GNU[78], que permet executar un programa de manera que al finalitzar se'n reporten diverses mètriques registrades durant la seva execució, principalment aquelles presents en l'estructura *rusage* retornada per la crida de sistema *wait3*. El principal criteri de selecció d'aquesta aplicació és la gran quantitat d'informació que recull, que pot ser d'utilitat alhora de realitzar inferències sobre els orígens dels resultats obtinguts.

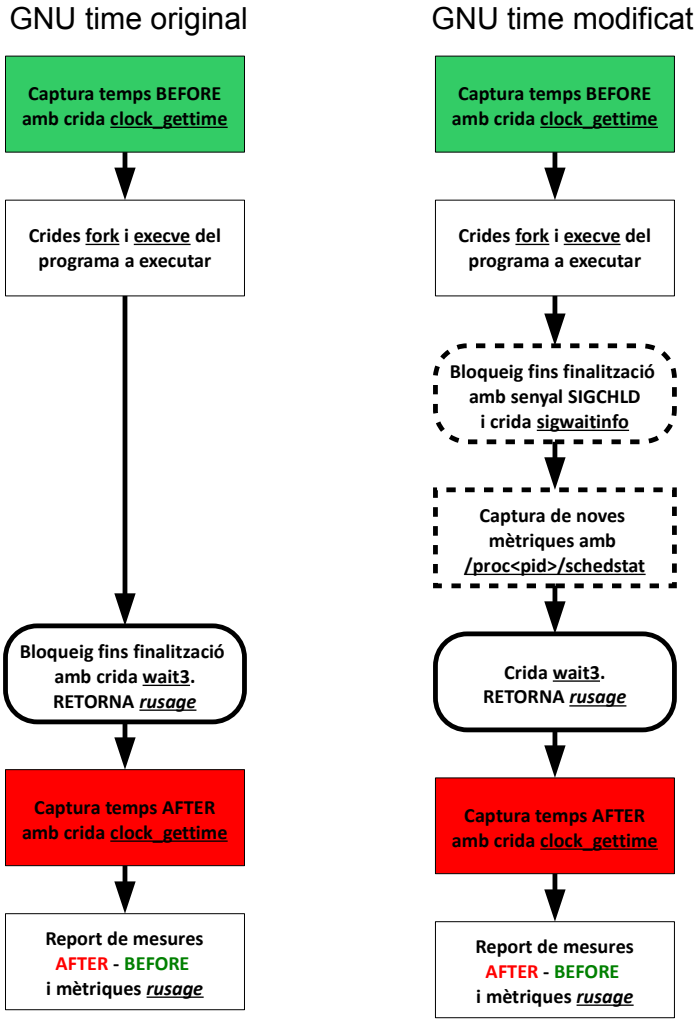
A més a més, es modifica la utilitat per aprofitar la possibilitat de recollir el temps en execució i el temps en espera de planificació registrat mitjançant la opció de configuració SCHEDSTATS. La modificació consisteix, en línies generals, en crear un procés on s'executa la aplicació, esperar la finalització de la mateixa i, finalment, interpretar els resultats mitjançant la interfície /proc/<pid>/schedstats [KER12], on *pid* és el número de procés creat.

Aquesta modificació, tanmateix, només reporta els temps corresponents per al procés executat sense incorporar els temps dels fills pel que no és útil per aplicacions que basen el seu funcionament en la creació i execució d'altres processos com pot ser el cas de un *script*.[79]

En la figura 34 es representa el principi de funcionament de l'aplicació així com de la modificació realitzada. Els codi complet de la modificació es pot trobar al apartat 8.3.2 de l'annex.

78 No s'ha de confondre amb la comanda similar incorporada en *shells* com el *bash*.
79 Una possible modificació de la mateixa consistiria en, un cop finalitzat el procés creat, recórrer la llista de processos fills obtenint les mètriques de cadascun d'ells.

Figura 34: Principi de funcionament de *GNU time* original i modificat



L'aplicació s'executa des de la línia de comanes i de les diverses opcions s'utilitza únicament la utilitzada per a definir la presentació dels resultats (-f) amb les següents opcions:

- %C : comanda executada.
- %P : percentatge CPU utilitzat durant la seva execució (%).
- %e : temps total transcorregut (segons).
- %U : temps transcorregut en l'espai d'usuari (segons).
- %S : temps transcorregut en l'espai de nucli (segons).
- %a : temps en execució (segons, afegit).
- %b : temps en cua d'espera del planificador (segons, afegit).
- %w : nombre de canvis de context voluntaris.[80]
- %c : canvis de context involuntaris.
- %R : fallades de pagina menors
- %F : fallades de pagina majors

80 Per exemple, esperant la finalització d'una petició E/S.

5.7.3 Mesura de càrregues

L'eina escollida durant la caracterització de les diverses càrregues a aplicar al sistema és *vmstat*[81], disponible al paquet *sysstat*, una aplicació de report de l'activitat del sistema pel que fa als processadors, la memòria i el disc. El principals criteris de selecció d'aquesta utilitat són la quantitat i diversitat de mesures que ofereix així com la seva facilitat d'ús. S'utilitza únicament els paràmetres *1*, que indica l'interval de report en segons, i *-n,* que evita la impressió periòdica de capçalera.

5.8 Caracterització de càrregues

S'estableix el propòsit de mantenir la càrrega del sistema alta de manera que s'incrementin els camins de codi recorreguts en el nucli i l'aparició d'artefactes de maquinari.[82] Amb això és pretén augmentar el nivell de contenció del sistema, incloses la competició de seccions crítiques no preemptibles, augmentant d'aquesta manera la probabilitat de detectar les pitjors latències.

Amb aquest propòsit es seleccionen i ajusten una sèrie d'aplicacions de *benchmarking* que s'utilitzen en l'experiment per modelar una sèrie de càrregues. Més concretament, es modelen càrregues de manera que augmenti el nombre d'interrupcions, de manera directa o indirecta, així com l'exercici dels camins del nucli encarregats de la gestió dels principals subsistemes: memòria, disc i xarxa.

Per tal de verificar que, per un banda, aquestes càrregues són realment notables i que, per un altre banda, cap d'elles desplaça a les demés de manera que en comptes d'augmentar en disminueixi la cobertura de codi i els artefactes de maquinari produïts, es verifica l'impacte d'aquestes càrregues mitjançant l'aplicació triada per aquest propòsit (*vmstat*).

5.8.1 Càrregues modelades

En les properes seccions s'analitza la afectació de les càrregues de següent taula. Aquestes han estat analitzades mitjançant l'exploració de les gràfiques obtingudes amb l'aplicació *gnuplot* i gràcies a una sèrie de *scripts* creats que es poden trobar a l'apartat 8.4 i 8.5 de l'annex. D'aquestes gràfiques es presenten únicament la d'aquelles càrregues que s'han acabat utilitzant en l'experiment per ser les més rellevants. El registre de mètriques s'ha realitzat amb el nucli configurat amb l'opció NO PREEMPT.

Taula 13: Càrregues utilitzades durant l'experiment

Nom	Aplicació	Descripció	Objectiu
idle	-	Sistema en repòs.	Adquirir un punt de referència sobre els valors de les mètriques.
smem	stress	Adquisició i alliberació de memòria.	Forçar la expulsió de les pàgines de disc en memòria cau, les memòries intermèdies i forçar el *swapping* esporàdicament.
zipuncached	bunzip2	Càrrega balancejada.	Utilitzar una càrrega balancejada entre ús de processador, memòria i disc per a la mesura de rendiment.

81 En la experimentació amb la modelació de les càrregues també s'han utilitat altres eines com *top, htop, slabtop, iostat, pidstat, sar* o *netstat* que no s'inclouen ja que són de menor rellevància i no són imprescindibles per explicar la caracterització de les mateixes.

82 Es defineixen com artefactes de maquinari aquells esdeveniments del mateix que provoquen un canvi en els camins d'execució de tal manera que afecten al seu determinisme augmentant o disminuint, per exemple, la latència. Un exemple clar d'aquests artefactes el trobem en les fallades de pàgina.

Nom	Aplicació	Descripció	Objectiu
zipu	bunzip2	Càrrega balancejada en USB.	Mateix objectiu que *zipuncached* però sobre una unitat externa connectada al bus USB.
du	du	Recorregut de tots els i-nodes del disc.	Estressar l'accés a disc (lectura), forçar l'adquisició de cau de i-nodes mitjançant el subsistema *slab*.
iozonewrite	iozone -i 0	Escriptura seqüencial.	Estressar l'accés a disc (escriptura), model fitxer.
iozoneread	iozone -i 1	Lectura seqüencial.	Estressar l'accés a disc (lectura), model fitxer.
iozonerand	iozone -i 2	Escriptura i lectura aleatòria.	Estressar l'accés a disc (lectura i escriptura), model BBDD.
pingl	ping	Allau de *ping* local.	Generar una alta càrrega de CPU en espai de nucli i alt nombre de canvis de context.
pinge	ping	Allau de *ping* remot.	Generar una alta càrrega de CPU i alt nivell d'interrupcions.
netperfudp	netperf	Allau de paquets UDP local.	Generar una alta càrrega de CPU en espai de nucli i alt nombre de canvis de context.
netperftcp	netperf	Allau de paquets TCP local.	Generar una alta càrrega de CPU i alt nivell d'interrupcions.
abperf	ab	Múltiples peticions al servidor web *apache* local.	Generar una alta càrrega de CPU i alt nivell d'interrupcions, comunicacions entre processos i
hackbench	hackbench	Creació i comunicació de diversos processos.	Estressar el planificador sistemàticament de manera puntual.

5.8.2 Llegenda

En la següent taula s'enumeren els acrònims utilitzats en les gràfiques i taules de valors.

Gràfica de CPU			Gràfica de memòria i disc		
Eix	Acrònim	Descripció	Eix	Acrònim	Descripció
Y1	%usr	percentatge CPU en espai d'usuari	Y1	swap	memòria virtual usada
Y1	%sys	percentatge CPU en espai de nucli	Y1	mem	memòria lliure
Y1	%iow	percentatge CPU esperant I/O	Y1	buff	memòria utilitzada per operacions intermèdies a disc
Y1	%idl	percentatge CPU en repòs	Y1	cache	memòria utilitzada per a cau de disc
Y2	ints	interrupcions per segon	Y2	s-in	pàgines de memòria llegits de disc per segon
Y2	c-sw	nombre de canvis de context per segon	Y2	s-out	pàgines de memòria guardades a disc per segon
			Y2	b-inp/s	blocs llegits de disc per segon
			Y2	b-out/s	blocs guardats a disc per segon

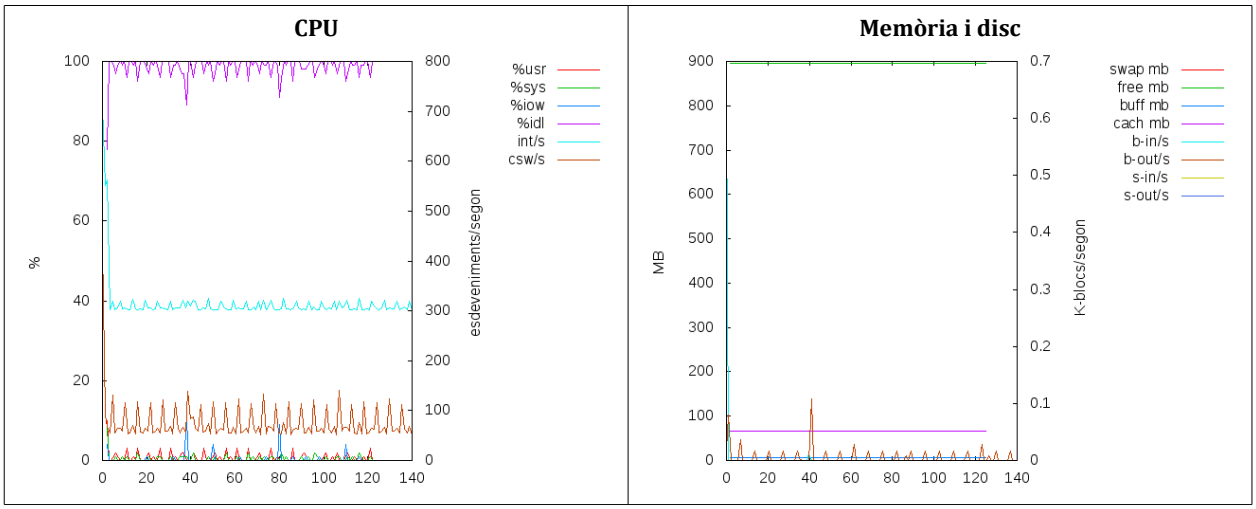
5.8.3 Avaluació de les càrregues

5.8.3.1 *Idle*

Es manté el sistema en repòs i s'observa que: existeix un nivell d'interrupció entorn als 300Hz, freqüència del *tick* de rellotge; hi han uns 900MB lliures, i hi ha un activitat d'escriptura a disc cada 5 segons, probablement deguda al procés *pdflush* encarregat de realitzar el *write-back* de la memòria cau a disc per a dispositius de bloc muntats amb la opció asíncrona (async).[83]

83 Linux contempla dues opcions de muntatge de disc: síncron i asíncron. En el primer cas, les escriptures és realitzen directament a disc en el moment de la seva petició. En el segon cas, es realitzen a la memòria cau de disc que és actualitzada periòdicament segons

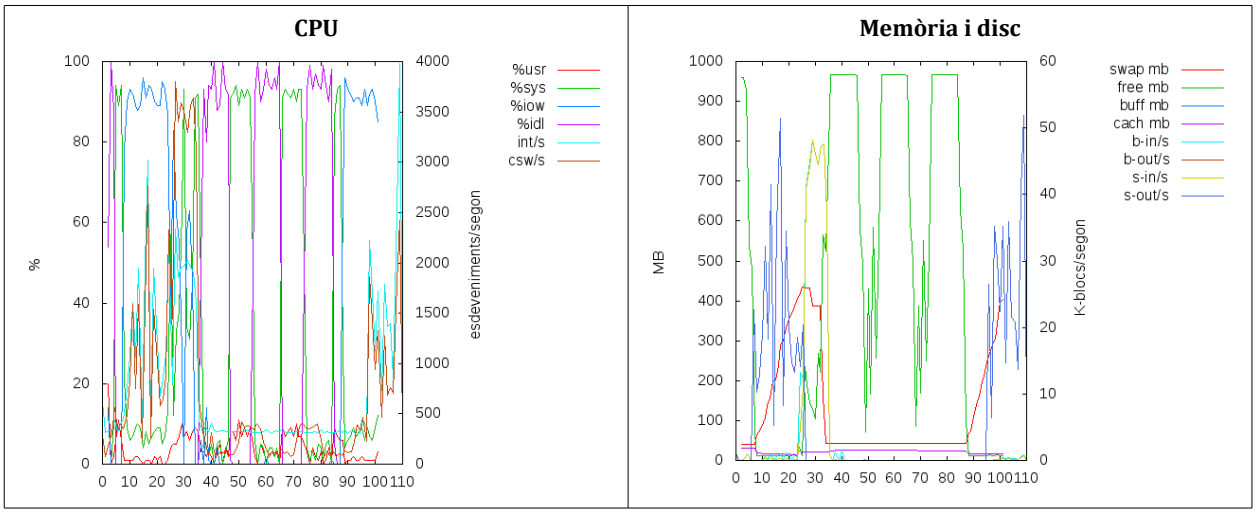
Figura 35: Sistema en repòs



5.8.3.2 *smem*

En la següent prova s'executa l'aplicació *stress*[84] perquè cada 10 segons crei 5 fils amb una diferència de 500ms que reservin 256MB de memòria cadascun i s'observa com efectivament es produeix una apropiació progressiva de la memòria lliure i certa activitat de *swap*. A més, s'aprecia com coincideixen les àrees de espera E/S (%iow) amb les activitats de lectura (s-out) i guardat de pàgines (s-in) a memòria *swap,* un increment notable de les interrupcions i canvis de context, producte probablement de les les fallades de pàgina i la interrupció del dispositiu de disc.

Figura 36: Estrès de memòria (stress -q -m 5 -l 4 --backoff 500000)



5.8.3.3 *zipuncached*

En la següent prova s'executa la comanda *bunzip2* per avaluar si és una càrrega prou adient per a la mesura ponderada del rendiment[85] i s'observa els següents trets rellevants: existeix una alternació en l'execució del codi d'usuari i el codi de nucli, s'aprecien pics d'interrupcions coincidents amb el període d'actualització de la memòria cau de disc (tipus write-back), es produeix una utilització

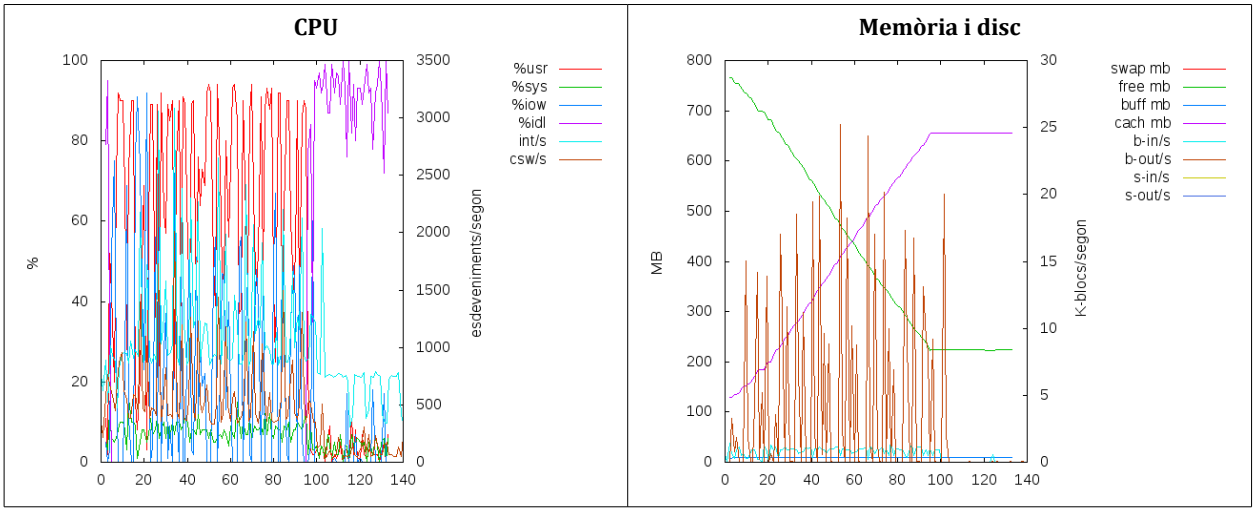
el paràmetre de nucli */proc/sys/vm/vm/dirty_writeback_centisecs* que, en l'entorn avaluat, té un valor de 500us. [MOR08]

84 Prèvia realització de certes modificacions que es poden consultar al apartat 8.3.3 de l'annex.

85 Les operacions d'un programa descompressor són: lectura (disc) , descompressió (CPU i memòria) i escriptura (disc) de manera iterativa . Un comportament que, a priori, s'intueix força balancejat pel que fa a l'ús dels principals recursos d'un sistema operatiu.

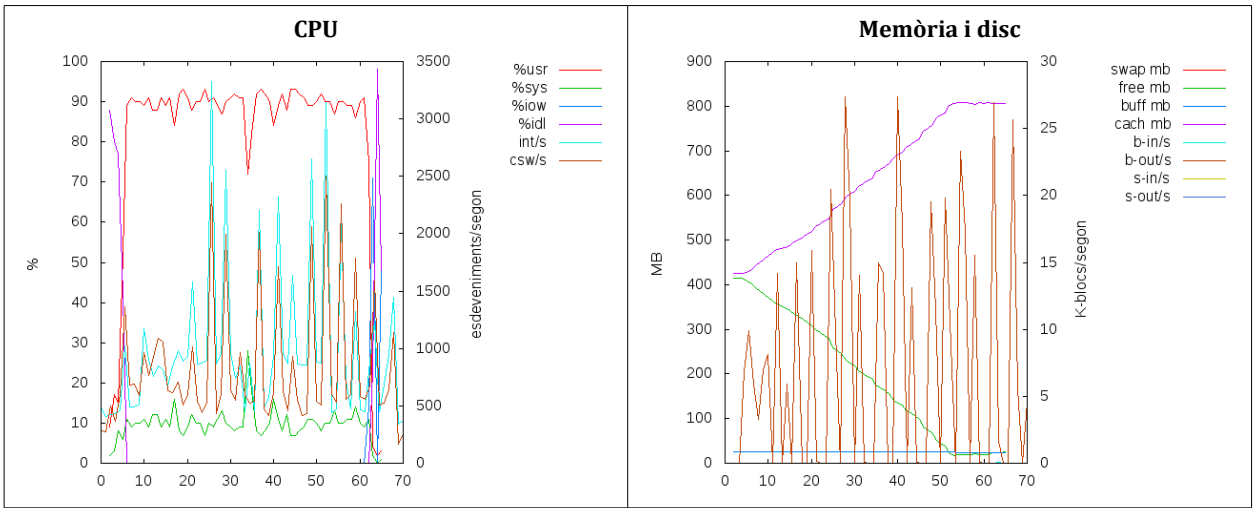
progressiva de la memòria lliure com a cau de disc i, finalment, que el nivell d'escriptura és major que el de lectura.

Figura 37: Càrrega balancejada (bunzip2 -f -k -q files/linux-3.2.32.tar.bz2 files/)



Després d'observar l'ús de la memòria cau de disc es repeteix la prova i es verifica com en aquesta segona execució els sistema operatiu serveix els blocs de disc directament de la memòria cau, ja que, per un costat, no existeix activitat de lectura de disc i, per altra costat, el temps d'execució ha disminuït. Com que en l'experiment final caldrà executar aquesta càrrega repetides vegades per a obtenir una mesura mitjana del rendiment es repeteix la prova prèvia utilització de la operació *drop_caches*[86] i se'n verifica la seva utilitat en la alliberació de la memòria cau de disc.

Figura 38: Càrrega balancejada *bunzip2* des de memòria cau de disc



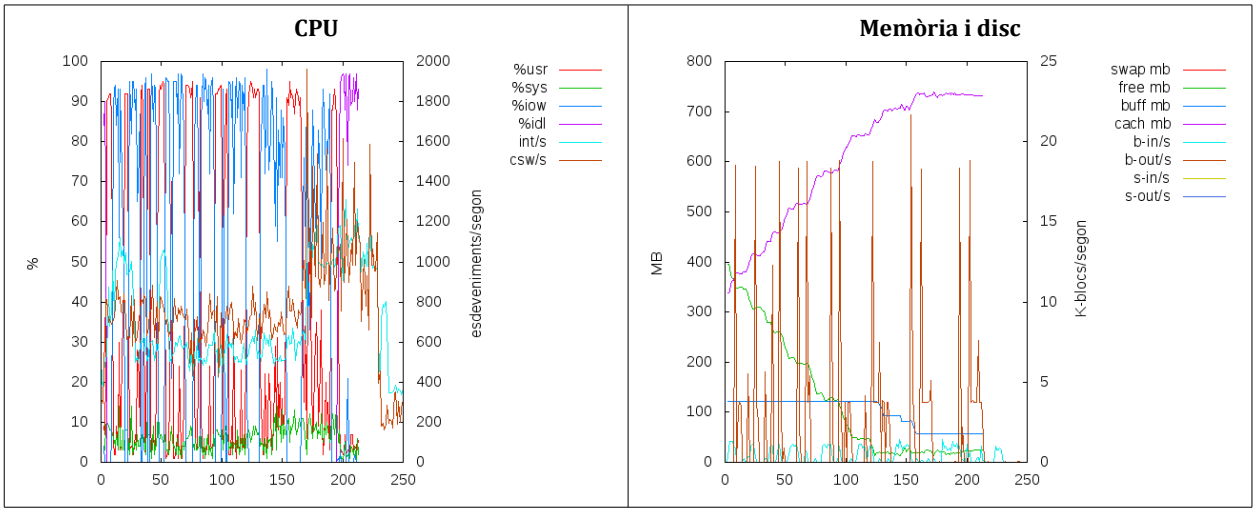
5.8.3.4 *zipu*

En aquest experiment és repeteix la comanda utilitzada en l'anterior sobre un dispositiu de bloc *flash disc* amb interfície USB del que s'espera un ordre d'accés a disc similar -ambdós discs són de tecnologia *flash*- amb la major sobrecàrrega de les comunicacions USB. Un cop realitzada la prova s'observa que, en línies generals, segueix un patró d'utilització similar, tot i que amb un pitjor rendiment aparent, els pics d'escriptura tenen menor alçada però més àrea, és a dir, són més

86 Aquesta operació es realitza a través de la interfície */proc/sys/vm/drop_caches* i permet l'escriptura de tres arguments: 1 per alliberar la memòria cau de disc, 2: per alliberar la memòria cau estructures del sistema d'arxius i 3: per allibera ambdues. [MOR08]

dispersos. S'observa, també, com la quantitat de memòria intermèdia (*buffer*) utilitzada és major i com al final de la prova aquesta entra en competició amb la memòria cau de disc. Finalment s'aprecia una major activitat de les interrupcions i major durada de la prova, fets que probablement són, respectivament, causa i conseqüència d'una major sobrecàrrega.

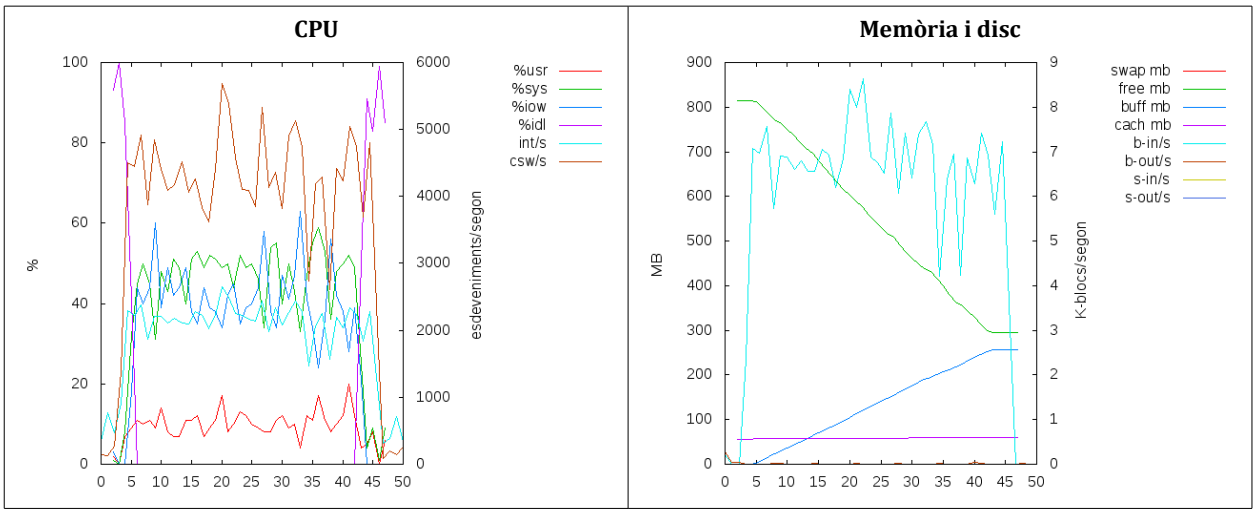
Figura 39: Càrrega balancejada (bunzip2 -f -k -q /media/usb0/linux-3.2.32.tar.bz2 files/)



5.8.3.5 *du*

En la següent prova s'utilitza la utilitat *du*[87] per recórrer les metadades del sistema d'arxius, i d'aquesta manera, realitzar una lectura aleatòria de disc[88] i, alhora, exercitar aquest subsistema del nucli de manera constant. A nivell de processador s'observa un increment notable dels canvis de context i interrupcions produïts probablement per la major intensitat en l'accés a disc d'aquesta aplicació . El codi s'executa en la majoria del temps entre el nucli i les esperes E/S pel que sembla adient quant a cobertura de codi de nucli.

Figura 40: Lectura de disc (du /)



De manera similar a l'anterior experiment s'observa, també, un efecte d'apropiació de la memòria tot i que aquest cop per part de la memòria intermèdia (*buffer*). Per altra es comprova que un cop

87 Aquesta utilitat reporta l'espai ocupat pels arxius gràcies a la informació extreta dels blocs *inode*. [KER11]
88 Ja que els blocs de *inode* es troben en cadascun dels 220 grups de blocs -per al equip utilitzat i segons la utilitat *ext3grep*. [KER11]

finalitzada la prova entorn el nivell d'aquesta només ocupa aproximadament la meitat de la memòria lliure perduda. Finalment, es verifica com aquesta quantitat correspon a la memòria cau de metadades del sistema d'arxius (*inode* i *dentri*) gestionada per la capa *slab*.

En la següent taula es comprova com no és suficient amb destruir la memòria cau de disc (*drop_caches=1*) escriure el paràmetre a la interfície de la comanda *drop_caches* vista anteriorment i com la memòria intermèdia (*buffer*) consumida durant l'execució de l'aplicació (249MB) correspon aproximadament a la utilitzada en la copia de les metadades del sistema d'arxius (*ext3_inode_cache*, *proc_inode_cache*, *inode_cache* i *dentry*).

	Inici	**Execució primer cop**	**drop_caches=1**	**Execució segon cop**
Ús memòria (comanda free)				
Used	98M	605M	347M	501M
Buffers	0M	249M	0M	149M
Cached	32M	34M	32M	34M
Ús memòria slab (comanda slabtop, memòria reportada com a used segons la comanda free)				
ext3_inode_cache	832K	191608K		
proc_inode_cache	0K	6392K		
inode_cache	672K	4476K		
dentry	1068K	55204K		

5.8.3.6 *iozone*

Per a realitzar l'estrès de disc s'avaluen dues eines de *benchmarking*, *bonnie++* i *iozone*, tot i que finalment s'acaba emprant *iozone* únicament ja que permet major versatilitat alhora de definir la la càrrega. Amb el propòsit d'ajustar els paràmetres de la utilitat s'executa perquè realitzi un escombrat d'un rang de mides de bloc (de 16KB a 16MB) i fitxer (de 16K a 1.5GB) per tal de verificar si hi han mides que afecten de manera característica l'accés a disc en el maquinari utilitzat.

En les següents figures és pot observar com el rendiment de la lectura i escriptura té una davallada significativa per sobre 500MB fet que coincideix amb les instruccions de l'aplicació on adverteix quant a l'afecte de la memòria cau de disc en mides de fitxer molt per sota del espai de memòria RAM. Pel que fa a la mida de bloc el comportament és bastant uniforme tot i que sembla presentar certes dificultats amb els blocs de 128KB.

Figura 41: Rendiment escriptura del disc (iozone -n 16k -g 1500m -y 16k -q 16m -i 0 -i 1 -a)

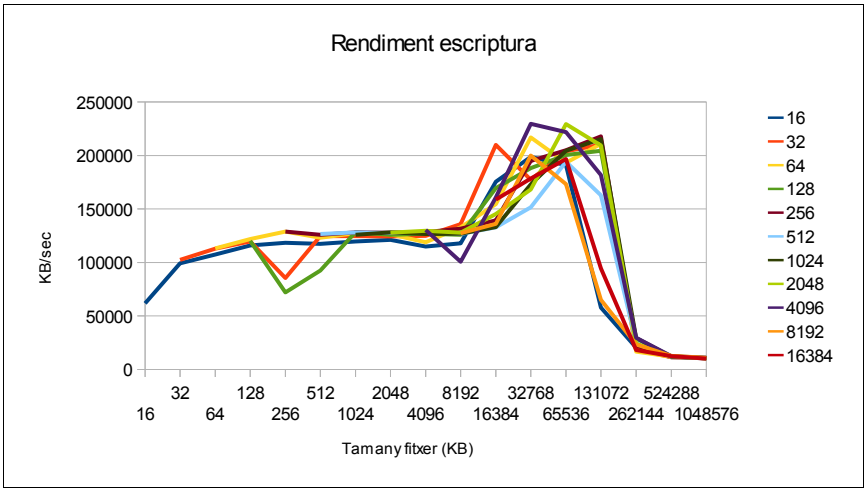
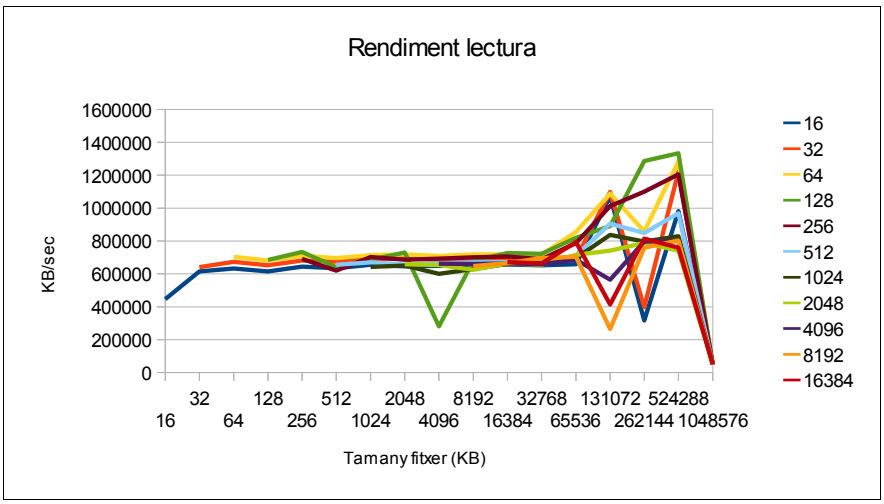


Figura 42: Rendiment lectura del disc (iozone -n 16k -g 1500m -y 16k -q 16m -i 0 -i 1 -a)

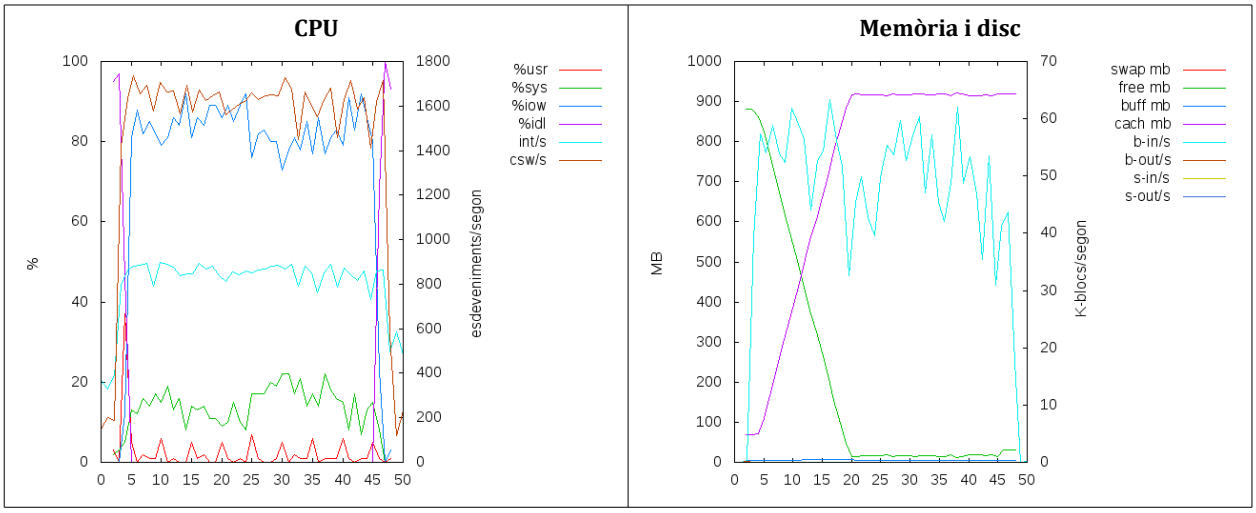


5.8.3.7 *Iozoneread*

En la següent prova s'utilitza *iozone* per a la generació d'una càrrega constant de lectura seqüencial, amb blocs de 128KB i un fitxer de 1GB. És a dir, la que correspondria a un accés a fitxer, per exemple. En les gràfiques s'observa la congestió de l'accés al disc en una tassa d'espera (%iow) molta alta i un nivell considerable de lectura de blocs (b-in).

A més, segons el resultats de l'aplicació, la diferència entre les operacions de lectura (51847 KB/s) i re-lectura (50171KB/s) no són massa notables pel que es confirma, i això es quelcom que es pot visualitzar en el nivell de lectura de la gràfica, que amb mides de fitxer properes al total de la memòria disponible al sistema s'aconsegueix evitar que el sistema eludeixi l'accés a disc servint els blocs directament des de la memòria cau.

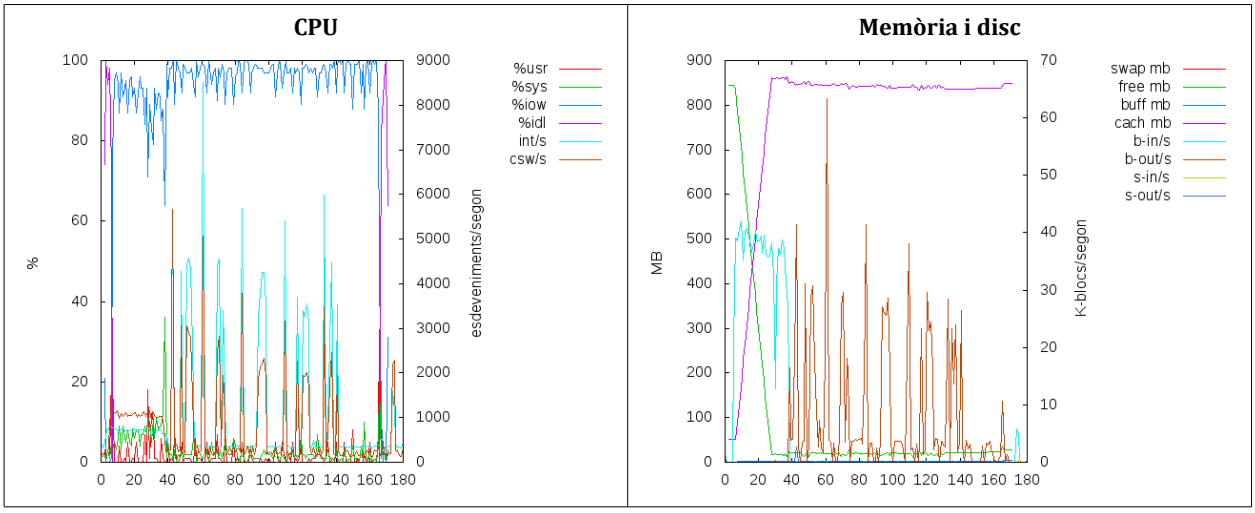
Figura 43: Lectura seqüencial (iozone -s 1g -r 128k -i 1 -f files/iozone.tmp -w)



5.8.3.8 *Iozonerand*

En la següent prova es torna a utilitza *iozone* per a la generació, aquest cop, d'una càrrega de lectura i escriptura aleatòria, similar a la que ens podríem trobar en una base de dades. Es observar la congestió en l'accés a disc en el com el alt nivell d'espera (%iow), en la part del test d'escriptura arribant al 90% -a partir del segon 40 aproximadament- així com pics esporàdics de canvi de context i interrupcions al llarg de la prova.

Figura 44: Escriptura / lectura aleatòria (iozone -s 1g -r 128k -i 2 -f files/iozone.tmp -w)



Es repeteix la prova per comprovar si l'aplicació desplaça o no a la resta d'activitats executant *zipuncached* en paral·lel i registrant les mètriques d'execució d'aquesta mitjançant *pidstat.* En la figura 77 se'n pot observar el resultat on s'aprecia clarament com *iozone* congestiona de tal manera els camins E/S que no és possible per a altres aplicacions de progressar (<2% de CPU).

Figura 45: Execució de *zipuncached* en paral·lel amb *iozone*

```
11:02:38 AM       PID    %usr %system  %guest    %CPU   CPU  Command
11:02:39 AM      8851    0.00    1.00    0.00    1.00     0  time
11:02:45 AM      8851    0.18    1.28    0.00    1.47     0  time
```

La prova es repeteix utilitzant un paràmetre (-J) que permet simular la càrrega de CPU que una aplicació real tindria entre accessos a disc mitjançant l'establiment d'una pausa dimensionada en milisegons entre peticions. A més, es pot observar com l'aplicació és incapaç de generar una mostra per segon fet indicatiu de les seves dificultats per progressar. Finalment, en la figura 77 s'aprecia l'efectivitat d'aquest paràmetre, calibrat a 300ms per la prova, quan el temps de processador de la càrrega *zipuncached* puja a un 50% aproximadament.

Figura 46: Execució de *zipuncached* en paral·lel amb *iozone -J 300*

```
11:14:22 AM       PID    %usr %system  %guest    %CPU   CPU  Command
11:14:23 AM      9826   46.00    3.00    0.00   49.00     0  bunzip2
11:14:24 AM      9826   52.53    2.02    0.00   54.55     0  bunzip2
11:14:25 AM      9826   65.05    5.83    0.00   70.87     0  bunzip2
11:14:26 AM      9826   45.00    2.00    0.00   47.00     0  bunzip2
11:14:27 AM      9826   45.54    2.97    0.00   48.51     0  bunzip2
11:14:28 AM      9826   78.19    4.66    0.00   82.84     0  bunzip2
```

5.8.3.9 *netperfudp*

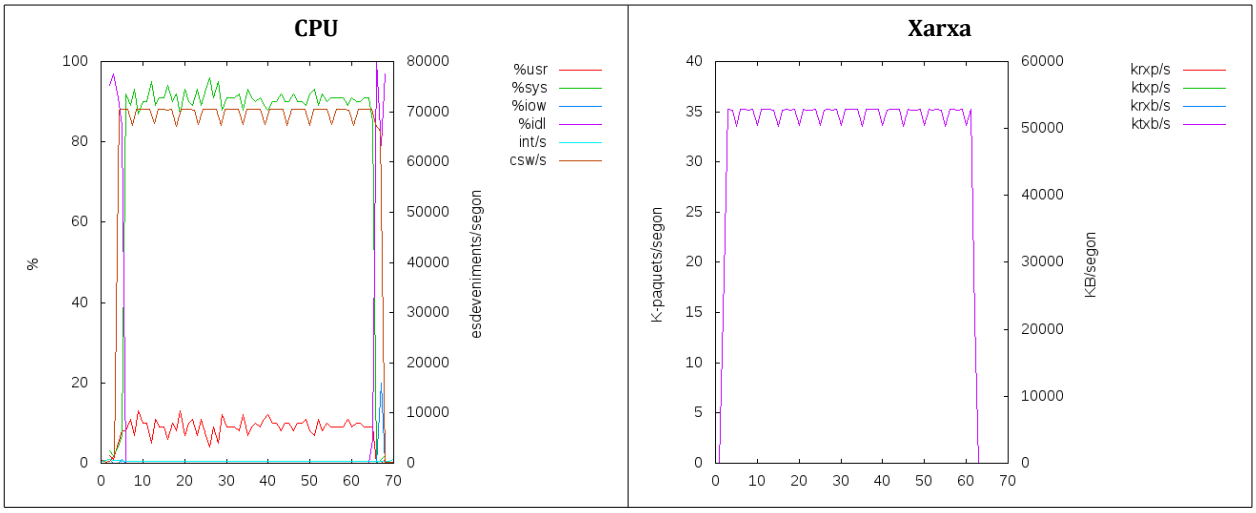
En la següent prova s'executa *netperf*, una utilitat de generació de càrrega i *benchmarking* de xarxa, executant el servidor i client en la pròpia màquina local[89]. Aquesta aplicació, segons els paràmetres utilitzats, realitza un allau constant de paquets UDP -el protocol no té control de flux- de 1472 bytes, la mida màxima emprada en la majoria de xarxes locals com és el cas d'Ethernet.[90]

89 La prova es realitza en context local per evitar que possibles interferències d'una xarxa externa afectin els resultats de les mesures.

90 Ethernet té un Maximum Transfer Unit (MTU) de 1500 bytes que restat als 8 bytes de capçalera ICMP i 20 bytes de capçalera UDP deixà un marge de 1472 bytes per a la transmissió de dades (*payload*).

En aquest cas, com en la resta de càrregues de xarxa, la següent figura mostra el nivell d'activitat de xarxa en comptes de l'activitat de memòria i disc que no és rellevant en aquestes aplicacions. Comparant les dues gràfiques s'observa un nivell molt alt de canvis de context (70000 canvis/segon) corresponent a la suma dels paquets per segon enviats (35000 paquets/segon) i rebuts (35000 paquets/segon). Aquesta sembla, per tant, una càrrega adient per detectar possibles latències tant pel que fa al context no preemptible d'interrupció així com en la cobertura dels camins de planificació i gestió de memòria.

Figura 47: Allau de paquets UDP (netperf -H localhost -t UDP_STREAM -l 60 -C -c -- -m 1472)

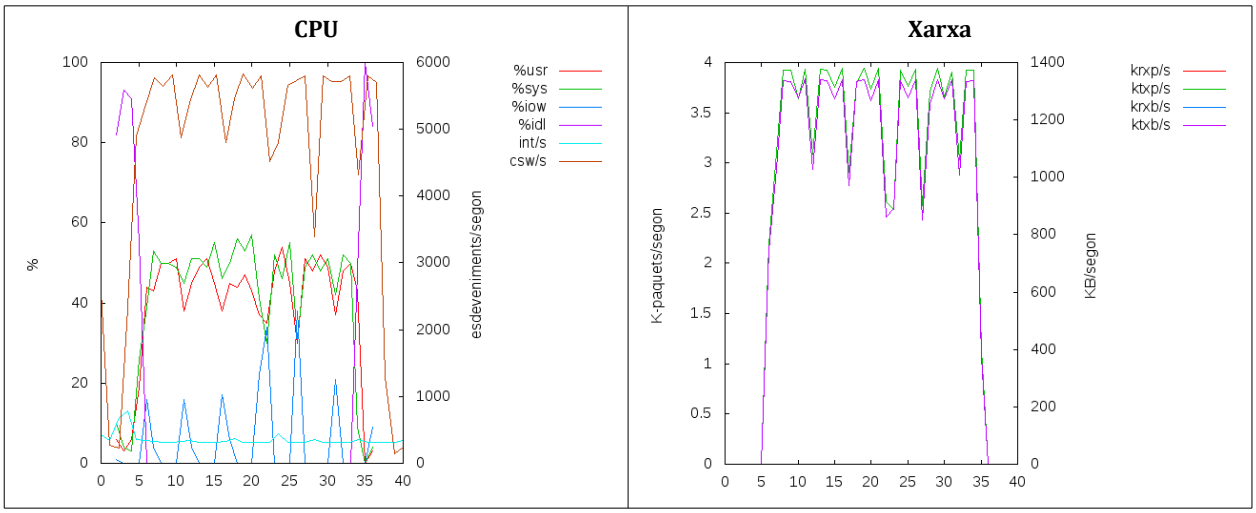


5.8.3.10 *abperf*

En la següent prova s'utilitza la utilitat *ab* pertanyent i emprada per al *benchmarking* del servidor *apache2*. Aquesta utilitat crea diversos fils i, de manera simultània, llança diverses peticions de al servidor web durant un temps limitat, en el cas de l'experiment 20 fils (-kc 20) i 60 segons (-t 60).

Durant l'execució s'observen nivells més baixos que els produïts per *netperf* sobretot pel que fa al rendiment de xarxa. Tanmateix, aquesta aplicació es valuosa pel fet que modela un perfil d'ús molt més proper a la realitat. També notar els pics de davallada que per la seva periodicitat sembla probable que es pugui tractar de l'operació d'escriptura de la memòria cau (*pdflush*).

Figura 48: Peticions web (ab -kc 20 -t 60 http://localhost/index.html)

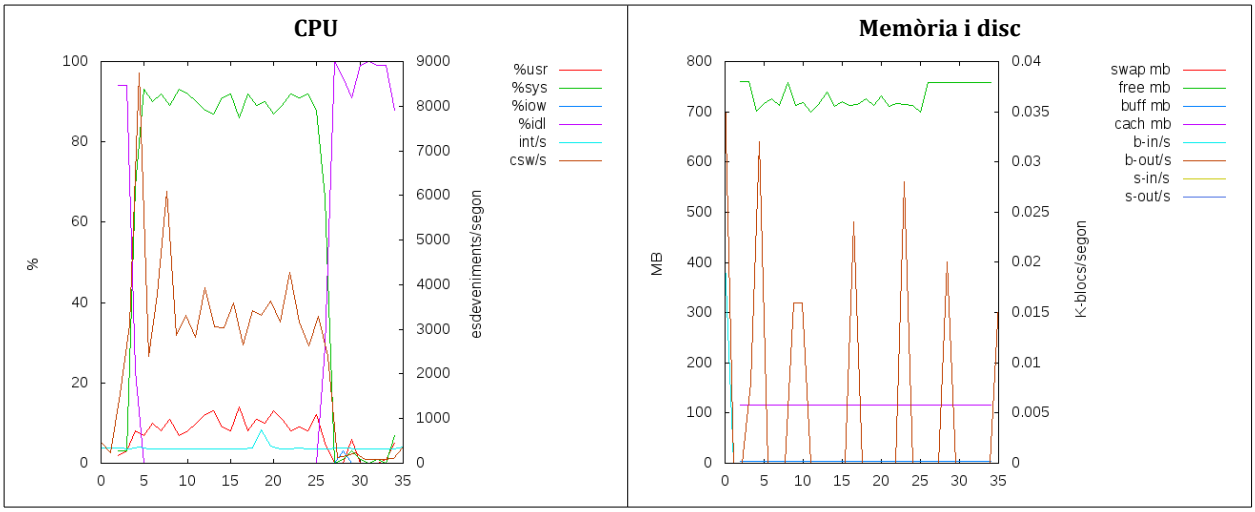


5.8.3.11 *hackbench*

Finalment, s'utilitza *hackbench* per a modelar una càrrega específica per al planificador ja que aquest és un subsistema no interrumpible[91] que està involucrat en qualsevol canvi de tasca, entre elles la necessària per a que un esdeveniment donat sigui atès.

L'aplicació executada sense paràmetres crea 400 tasques que es comuniquen enviant-se 100 missatges de 100 bytes entre elles. En les gràfiques s'observa una alta ocupació de la CPU en espai de nucli així com un alt nombre de canvis de context. En aquest aspecte sembla una prova adient per a l'experiment final.

Figura 49: Estrès planificador (hackbench)



5.9 Disseny final de l'experiment

Un cop avaluades les diferents càrregues i amb el propòsit de realitzar el disseny final de l'experiment s'agrupen les mesures realitzades segons dominis d'interès, s'estableixen uns objectius per a cada àrea, s'analitzen els valors d'aquestes mesures per a cada càrrega -veure taula 14 més endavant- i se'n realitza la tria i l'ajust de les mateixes de manera que l'avaluació del temps real i rendiment realitzat en els quatre nuclis es vegi sotmès a un nivell d'estrès alt i variat per ampliar la cobertura de codi de nucli i augmentar la probabilitat de coincidència d'esdeveniments augmentant, d'aquesta manera, la de detectar pitjors camins de latència (seccions no preemptibles de nucli i interrupcions).

5.9.1 Descriptors estadístics utilitzats

Pel que fa a les mesures de caracterització es seleccionen dos conjunts de descriptors estadístics segons el tipus d'utilització dels diferents dominis analitzats: diferència entre màxim i mínim, i mitjana aritmètica, desviació típica i màxim. En el primer cas, la diferència entre màxim i mínim s'utilitza per analitzar la utilització total d'un àrea determinada. En el segon cas, la mitjana aritmètica, la desviació típica i el màxim s'utilitza per a la mesura del nivell d'estrès en una àrea.

91 Essent el propi subsistema que realitza la planificació manipula les estructures de dades que contenen l'estat dels processos en execució i, per tant, per evitar condicions de carrera amb aquestes estructures, no permet se executada concurrentment.

5.9.2 Taula comparativa de càrregues

Taula 14: Comparativa de les càrregues utilitzades en l'experiment

	CPU									Esdeveniments						Planificació					
	MA	D	M	MA	D	M	MA	D	M	MA	D	M	MA	D	M	MA	D	M	MA	D	M
Proba	% codi d'usuari			% codi de nucli			% IO wait			Interrupcions / segon			Canvis context / segon			Espera execució			Espera bloqueig		
idle	1	1	3	0	1	2	0	1	10	312	32	562	72	25	140	0	0	1	0	0	0
smem	4	4	20	31	37	94	33	42	96	785	748	3980	754	982	3801	1	2	6	4	5	15
zipuncached	67	25	94	8	2	16	25	26	92	600	128	1072	609	351	2008	1	1	4	1	1	3
zipu	40	38	95	6	3	14	53	40	98	739	239	1313	806	219	1964	1	1	4	1	1	3
makeuncached	85	12	99	11	5	32	4	11	59	600	128	1072	349	236	1872	1	1	3	0	0	2
du0	9	2	16	49	6	64	42	6	60	2208	235	2688	4256	585	5232	1	0	2	1	0	1
iozonewrite	1	2	7	9	9	70	90	9	100	801	653	2994	746	591	2601	0	1	3	3	1	7
iozoneread	2	2	7	15	4	22	84	5	92	851	39	899	1621	77	1737	1	1	3	1	1	2
iozonerand/L	3	5	21	7	3	12	90	6	97	717	47	819	1053	85	1236	0	1	2	1	1	3
iozonerand/E	1	3	33	2	4	36	96	6	100	1175	1661	8498	757	1078	5671	0	1	4	7	2	14
pingl	11	3	25	89	3	94	0	0	0	345	9	373	306	50	425	1	0	2	0	0	0
pinge	7	2	12	61	9	78	0	0	3	2770	410	3474	6063	865	7657	2	1	3	0	0	0
netperfudp	9	2	13	91	2	96	0	0	0	318	2	325	69866	1247	70562	1	1	2	0	0	0
netperftcp	4	2	9	96	2	98	0	0	0	317	3	329	2686	10	2713	1	0	3	0	0	0
abperf	45	6	54	48	7	57	6	11	38	330	28	442	5405	594	5824	2	1	3	0	1	2
hackbench	10	2	14	90	2	93	0	0	0	344	93	743	3632	1401	8751	221	85	354	0	0	1

	Memòria			Swap						Accés a disc					
	M-m	M-m	M-m	MA	D	M	MA	D	M	MA	D	M	MA	D	M
Proba	Lliure	Buffers	Cache	Lectura Pag/s			Escriptura Pag/s			Lectura Blocs/s			Escriptura Blocs/s		
idle	408	160	24	0	0	0	0	0	0	0	1	12	5	14	108
smem	955872	356	14532	3800	12159	48116	7569	13131	51876	3990	12210	48116	7575	13135	51892
zipuncached	532936	744	518972	0	0	0	0	0	0	833	313	1540	4731	6937	25256
zipu	360036	65584	382460	0	0	0	0	1	8	412	471	1408	2196	4724	21657
makeuncached	22012	504	12316	0	0	0	0	0	0	121	521	4432	105	233	1532
du0	492288	241396	40	0	0	0	0	0	0	6663	773	8024	6	16	80
iozonewrite	678448	680	672788	0	0	0	1	6	44	13	71	716	19818	16341	79360
iozoneread	811676	2020	812552	0	0	0	2	6	28	50664	8127	63420	8	15	52
iozonerand/L	793460	688	792632	0	0	0	0	0	0	35894	6097	42144	24	63	320
iozonerand/E	23160	2436	19464	1	14	164	20	228	2588	106	666	5744	8153	11531	63372
pingl	556	128	292	0	0	0	0	0	0	0	0	0	4	9	60
pinge	280	80	160	0	0	0	0	0	0	0	0	0	4	9	48
netperfudp	496	40	0	0	0	0	0	0	0	0	0	0	7	17	68
netperftcp	6348	48	4628	0	0	0	0	0	0	0	0	0	6	15	80
abperf	6348	48	4628	0	0	0	0	0	0	0	1	4	171	340	916
hackbench	58760	32	128	0	0	0	0	0	0	0	0	0	6	11	32

Llegenda	M-m	Màxim – mínim	MA	Mitjana aritmètica	D	Màxim	M	Màxim

** Les diferències de memòria corresponen al següent sentit: decreixement per a memòria lliure i creixement per a buffers i cau de disc.*

5.9.3 Domini CPU

5.9.3.1 *Objectius*

En aquesta àrea s'analitza l'ús general del processador segons execució de codi de nucli, d'usuari, en repòs o espera E/S. En aquest aspecte són d'especial interès les càrregues que exerciten la major part del seu codi en espai de nucli ja que n'augmenten la seva cobertura i, per tant, les probabilitats de descobriment de camins de no preemptible en la mesura de la latència.

5.9.3.2 *Resultats*

Pel que fa a la CPU és pot observar com *zipuncached, zipu*, *makeuncached* i *abperf* són càrregues amb una gran presència en l'espai d'usuari, això indica que en proporció és més el codi que executen de l'aplicació que el que requereixen del nucli del sistema i, per tant, són càrregues poc interessants pel que fa a augmentar la la cobertura de codi de nucli.

Per contra, les aplicacions de càrrega de xarxa *(pingl, pinge, netperfudp, netperftcp*, *abperf*) utilitzen principalment codi de nucli així com *smem*, hackbench i du tot i que per motius diferents: comunicacions, reservació i alliberació de memòria, creació i destrucció de processos, o accés al sistema d'arxius

Finalment, s'observa com *smem, zipuncached* i les càrregues específiques de disc (*du*, *iozoneread, iozonewrite* i *iozonerand)* presenten una tassa molt alta pel que fa a la espera de peticions E/S. Cal notar, a més, que mentre les primeres presenten una mitjana relativament baixa i variada (desviació típica alta), les segons en presenten una de molt alta i mantinguda (desviació típica baixa). En conseqüència, caldrà tenir cura que les càrregues de disc no congestionin l'accés E/S de manera bloquegin altres activitats del mateix.

5.9.4 Domini esdeveniments

5.9.4.1 *Objectius*

En aquesta àrea s'analitzen el nombre d'interrupcions i canvis de context per segon. En el primer cas, a major nombre d'interrupcions major probabilitat de coincidència amb l'esdeveniment de temps real a mesurar i, per tant, previsible pitjor resposta. En el segon cas, a major nombre de canvis de context major atenció per part del nucli i, per tant, major cobertura de les seccions crítiques de codi i, en conseqüència, pitjor temps real.

5.9.4.2 *Resultats*

Quant a la freqüència de les interrupcions destaquen notablement *smem*, *du,* les operacions d'escriptura a disc (*iozonewrite* i *iozonerand/E*) i *pinge.* En el primer cas *smem* presenta una mitjana relativament baixa tot i que amb una desviació típica i màxim alts, fruit possiblement d'allaus esporàdics de fallada de pàgina i de peticions al dispositiu d'emmagatzemament en les etapes de *swapping,* tret que es pot verificar en les gràfiques.

Pel que fa a *du*, sorprèn l'alta tassa d'interrupcions sobretot si es compara amb *iozonerand*, l'altre procés de lectura aleatòria avaluat. En aquest aspecte, *du* és manifesta com una carrega adient pel que fa a l'augment de probabilitats de col·lisió d'un esdeveniment de temps real amb l'arribada d'altres interrupcions. De manera similar, les operacions d'escriptura a disc presenten un tassa força alta probablement provocada per l'accés intensiu que a disc que realitzen així com pinge suposadament per l'alta quantitat de paquets ICMP enviats i rebuts.

Quant a la freqüència dels canvis de context, destaca, de nou, *du* així com *iozonerand/L*[92], algunes de les càrregues de xarxa com *pinge*, *netperfudp* i *abperf*, i, finalment, *hackbench.* En referència a *du* i *iozonerand/L* -ambdos realitzen lectura aleatòria de disc- , cal senyalar que els màxims d'ambdues càrregues està entorn al mateix valor tot i que *du* presenta una major estabilitat (desviació típica baixa).

En aquest aspecte també són remarcables les càrregues de xarxa que no requereixen de control de flux, *netperfudp*, més notablement, o que es basen en l'enviament en paral·lel de diferents peticions, *pinge* i *abperf,* així com *Hackbench*, que utilitza creació, destrucció i comunicació entre processos.

5.9.5 Domini planificació

5.9.5.1 *Objectius*

En aquesta àrea s'analitzen el nombre de tasques en cua d'execució així com les tasques a la espera de la resolució d'alguna petició. En el primer cas, a major nombre de processos apunt per executar, major competició en el planificador i, per tant, una major probabilitat de coincidència amb l'esdeveniment de temps real. En el segon cas un elevat nombre de tasques en espera és indicatiu de la congestió en algun subsistema (contenció bus de dades, memòria principal, DMA, …) fet que podria dificultar de manera indirecta l'atenció de nous esdeveniments

5.9.5.2 *Resultats*

Pel que fa a l'estrès en el planificador destaca *hackbench* amb una mitja molt alta de processos competint per la CPU. També cal senyalar que *smem,* tot i no tenir una mitjà alta, presenta alguns pics així com un nivell força alt de tasques bloquejades coincidents amb les zones de major activitat de *swap* així com les aplicacions d'escriptura de disc (*iozonewrite* i *iozonerand/E*) que presenten un nivell força elevat de tasques en espera degut a la congestió dels camins E/S.

5.9.6 Domini memòria

5.9.6.1 *Objectius*

En aquesta àrea s'analitza el nivell de memòria lliure, l'ús que se'n fa de la memòria ocupada i la competició dels diferents usos que el sistema operatiu en fa: memòria de processos, memòria intermèdia (*buffer*) i cau de disc. En aquest cas quan major sigui la competició de memòria menor serà la optimització que el sistema operatiu en podrà fer de la jerarquia de memòries, major la gestió d'alliberació i cessió de recursos entre les diverses necessitats. Això haurà de provocar fallades de pàgina que augmentin el nombre d'interrupcions, l'exercici del *swapping*, la lectura de blocs de disc en comptes de cau i, finalment, la congestió dels camins E/S dificultant l'atenció dels esdeveniments per l'augment de la cobertura de codi no preemptible.

5.9.6.2 *Resultats*

Quant a l'ús de memòria destaquen tres tipus d'aplicacions: *smem*, pel que fa a la memòria de processos; algunes de les proves d'estrès de disc (*iozonewrite*, *iozoneread* i *iozonerand/L*), pel n que fa al cau de disc[93], i *du,* pel que fa a la memòria intermèdia (*buffer*) i la de processos. Pel que fa aquest últim cas, cal senyalar que com s'ha vist en l'avaluació de la càrrega que la memòria utilitzada per la aplicació es manté com a cau de metadades del sistema d'arxius (*inodes*).

92 S'anomena iozonerand/L a la part de l'experiment corresponent a la lectura i, de manera paral·lela, iozonerand/W a la d'escriptura.
93 Cal senyalar que el decrement de memòria lliure correspon a l'increment de memòria cau de disc i que per tant s'ha d'atribuir a aquest i no pas a un ús d'aquesta memòria per part del procés en si.

En general, s'aprecien, doncs, tres competidors notables que han de permetre exercitar els algorismes de repartició de memòria pel que fa a la distribució de la mateixa en memòria de procés, de *buffer* o cau de disc. Cal tenir cura, de nou, amb l'ajust de les càrregues per evitar que una congestió dels camins E/S bloqueigi la resta d'activitats del sistema.

5.9.7 Domini swap

5.9.7.1 *Objectius*

En aquesta àrea s'analitza el nombre de pàgines de memòria principal escrites i llegides a memòria auxiliar (*swap*). En aquest cas, un elevat nombre d'intercanvis és indicatiu de una condició d'estres de la memòria que porta a una major probabilitat de fallades de pàgina i, en conseqüència, de congestió dels camins E/S i coincidència amb l'execució de les seccions no preemptibles d'aquest subsistema.

5.9.7.2 *Resultats*

Pel que fa el *swapping* només destacar que la càrrega *smem* assoleix el seu objectiu pel que fa a la competició de memòria desplaçant temporal algunes pàgines de memòria principal a auxiliar.

5.9.8 Domini accés a disc

5.9.8.1 *Objectius*

En aquesta àrea s'analitzen els accessos a disc per lectura o escriptura. Un nivell alt d'aquestos es indicatiu d'una alta activitat E/S que podria conduir a una congestió dels camins d'accés als dispositius que impactés en l'atenció dels processos.

5.9.8.2 *Resultats*

Quant a l'accés a disc destaquen *zipuncached*, *zipu* i *makeuncached*, processos que combinen el comput amb l'escriptura a disc, així com les càrregues d'estrès de disc. Es verifica, d'aquesta manera, que les càrregues d'estrès a disc compleixen amb el seu propòsit tot i que s'evidencia la necessitat de realitza-ne l'ajust de manera que es controlin els nivells de congestió i s'evita el bloqueig de la resta d'activitats.

5.9.9 Selecció de càrregues

En general, després de les proves de caracterització, es verifica la utilitat de cadascuna de les càrregues proposades per a produir un nivell d'estrès alt en cada un dels subsistemes adreçats. A més, s'obté un model numèric del grau de càrrega en les proves realitzades en el que s'observen alguns fets inesperats com la capacitat de *du* de provocar un alt nivell d'interrupcions o la capacitat de *netperfudp* de generar un nivell molt alt de canvis de context.

Addicionalment, es constata que les aplicacions d'estrès de disc provoquen una congestió considerable que cal tenir en compte a l'hora d'ajustar-les i evitar el bloqueig de la resta d'activitats així com que les aplicacions de xarxa produeixen força intercanvi de la CPU, tot i que gens estrès en la memòria o disc, fet que demostra la seva complementarietat.

En la següent taula es recull el sumari de la valoració de les diferents càrregues un cop caracteritzades. Aquest inclou la decisió de inclusió o no inclusió, els motius, i si es considera o no un ajust addicional de la càrrega. Com a criteri d'avaluació, davant de dues càrregues similars es selecciona la de característiques més interessants evitant, d'aquesta manera, el detriment de la seva activitat en benefici de la de l'altre.

Taula 15: Sumari de l'avaluació de les càrregues

Càrrega	CPU, esdeveniments i planificació	Memòria i disc	Propòsit inicial	Valoració / Inclusió
smem	Execució de codi en el nucli, certa congestió E/S en zones de *swapping* i augment de les interrupcions.	Gran estrès del sistema de memòria que provoca *swapping* traduït en un estrès en l'accés a disc.	Provocar competició i desallotjament de pàgines de memòria, *buffers* i cau de disc.	Compleix amb el seu propòsit tot i que cal ajustar-la perquè no provoqui una congestió excessiva en l'accés a disc. **S'inclou amb certs ajustos.**[94]
zipuncached	Execució de codi en usuari i certa congestió E/S.	Pot arribar a reclamar la meitat de la memòria del sistema (500MB) en cau de disc. Té una activitat notable en escriptura.	Càrrega balancejada per a comparació del rendiment en els 4 nuclis.	Compleix amb el seu propòsit. Es tracta d'una aplicació balancejada pel que fa a CPU i E/S. **S'inclou.**
zipu	Similar a *zipuncached* amb major nivell de congestió E/S.	Similar a *zipuncached* amb major nivell d'ús de *buffers* i una tassa de la meitat d'escriptura.	Càrrega balancejada per a comparació del rendiment en els 4 nuclis. Exercitació del subsistema USB.	Tot i ser força similar a l'anterior es interessant pel que fa a les particularitats que presenta i al fet que exercita el subsistema USB. **S'inclou.**
make uncached	Execució de codi majoritàriament en espai d'usuari.	Pot arribar a reclamar un terç de la memòria principal (300MB) en cau de disc. Té una activitat notable en escriptura tot i que molt menor a *zipuncached* o *zipu*.	Càrrega balancejada per a comparació del rendiment en els 4 nuclis.	Tot i ser una càrrega força balancejada en CPU i E/S no provoca massa estrès en aquest segon. **Es descarta en detriment de *zipu* i *zipuncached*.**
du	Execució de codi repartida en espai de nucli i espera E/S. Alt nivell d'interrupcions i canvis de context.	Arriba a reclamar una cinquena part de la memòria (200MB) en cau de *i-nodes* i *dentrys*. A més, fa un ús intensiu dels buffers (200MB més) i presenta un nivell alt en la lectura a disc.	Càrrega en l'accés de lectura a disc.	Compleix amb el seu propòsit. A més, mostra altres aspectes interessants per l'avaluació no esperats. **S'inclou.**
iozonewrite	Majoria del temps en espera de E/S, arribant fins i tot al 100%. Nivell alt i sostingut d'interrupcions.	Pot arribar a reclamar pràcticament la totalitat de la memòria disponible per a cau. Activitat molt alta en escriptura de disc. Arriba a provocar *swapping* fet que congestiona encara més l'accés a disc.	Càrrega d'escriptura seqüencial a disc (tipus fitxer).	Compleix amb el seu propòsit tot i que presenta un bloqueig de la CPU molt alt i congestiona en gran mesura la memòria principal i secundaria. **Es descarta per la elevada congestió que provoca.**

94 Veure apartat 8.6.5 de l'annex.

Càrrega	CPU, esdeveniments i planificació	Memòria i disc	Propòsit inicial	Valoració / Inclusió
iozoneread	Majoria del temps en espera de E/S.	Pot arribar a reclamar la totalitat de la memòria per a cau. Arriba a provocar *swapping* congestionant temporalment el sistema.	Càrrega de lectura seqüencial a disc.	Compleix amb el seu propòsit. **S'inclou executada alternativament amb *iozonerand* per tal de no localitzar excessivament la cobertura de codi.**
iozonerand	Majoria del temps en espera de E/S, arribant fins i tot al 100% en la part d'escriptura. Nivell alt i irregular d'interrupcions, amb els majors pics observats tot i que una mitjana menor a l'escriptura seqüencial.	Pot arribar a reclamar pràcticament la totalitat de la memòria disponible. En l'escriptura arriba a provocar *swapping* en uns nivells considerables fet que congestiona temporalment el sistema.	Càrrega de lectura/escriptura aleatòria (similar a l'accés a base de dades).	Compleix amb el seu propòsit tot i que presenta un bloqueig de la CPU molt alt i congestiona en gran mesura la memòria principal i secundaria. **S'inclou executada alternativament amb *iozoneread* i ajustada perquè produeixi menor congestió.**[95]
pingl	Majoria del temps executant codi en el nucli.	Res notable a comentar.	Càrrega en l'espai de nucli.	Compleix amb el seu propòsit tot i que de manera menys efectiva que *netperfudp*. **Es descarta en detriment de *netperfudp*.**
pinge	Majoria del temps executant codi en el nucli. A més, alt nivell d'interrupcions i canvis de context.	Res notable a comentar.	Càrrega en l'espai de nucli i el sistema d'interrupcions.	Compleix amb el seu propòsit tot i que de manera menys efectiva que *netperfudp*. **Es descarta en detriment de *netperfudp*.**
netperfudp	Majoria del temps executant codi en el nucli. Alt nivell d'interrupcions i el major nivell de canvis de context observat.	Res notable a comentar.	Càrrega en l'espai de nucli i el sistema d'interrupcions. Simulació d'una aplicació sense control de flux.	Compleix amb el seu propòsit a més presenta un nivell molt alt de canvis de context. **S'inclou.**
netperftcp	Majoria del temps executant codi en el nucli.	Res notable a comentar.	Càrrega en l'espai de nucli i el sistema d'interrupcions. Simulació d'una aplicació amb control de flux.	Compleix amb el seu propòsit tot i que de manera similar a *abperf,* però sense modelar un procés real. **Es descarta en detriment de *abperf*.**

95 En executar-se juntament amb smem no és imprescindible que la mida del fitxer superi la quantitat de memòria total del sistema ja que la disponible serà molt menor.

Càrrega	CPU, esdeveniments i planificació	Memòria i disc	Propòsit inicial	Valoració / Inclusió
abperf	Execució del codi repartida en usuari i nucli amb alt nivell de canvis de context.	Res notable a comentar.	Càrrega del servei web.	Compleix amb el seu propòsit. És interessant perquè modela un procés real. **S'inclou.**
hackbench	Majoria del temps executant codi en el nucli, alt nivell de canvis de context i estrès del planificador.	Res notable a comentar.	Càrrega del planificador.	Compleix amb el seu propòsit. **S'inclou.**

5.10 Execució de l'experiment

L'experiment final consta de les característiques dissenyades seguint els criteris i propietats desitjades establerts fins aquest punt i que a mode de resum es recullen a continuació.

5.10.1 Nuclis avaluats

L'experiment (apartat 8.6, annex) consta de l'execució durant 8 hores[96] de l'entorn de càrrega i mesura dissenyat en apartats anteriors de quatre sistemes operatius amb nuclis versió 3.2.32 de Linux configurats de manera diferent amb la diferencia més notable de que cadascun d'ells utilitza un nivell de preempció de nucli més agressiu que l'anterior:

- no preempt : sense preempció de nucli.
- voluntary : preempció explícita en diferents punts del codi.
- preempt : preempció en qualsevol punt excepte seccions crítiques i interrupcions.
- rt : preempció en qualsevol punt del nucli, pedaç RT incorporat.

5.10.2 Mesura de la latència

Durant aquest temps es registren mètriques de latència mitjançant la programació de diversos temporitzadors -utilitzant *cyclictest*- a freqüències diferents amb el propòsit de disposar d'un cobertura temporal notable per a un anàlisi del comportament en temps real prou genèric. Més concretament s'utilitzen els següents intervals, triats de manera que no es produeixin efectes de sincronització entre elles[97], més concretament:

- 500RT : temporitzador associat a tasca de temps real de 500us de interval.[98]
- 1100RT : temporitzador associat a tasca de temps real de 1.100us de interval.
- 2300RT : temporitzador associat a tasca de temps real de 2.300us de interval.
- 5300RT : temporitzador associat a tasca de temps real de 5.300us de interval.
- 111000RT : temporitzador associat a tasca de temps real de 111.000us de interval.
- 270000RT : temporitzador associat a tasca de temps real de 270.000us de interval.
- 550000NORT : temporitzador associat a tasca de 550.000us de interval.

96 De l'ordre de 10^7 mostres de latència en el millor dels casos pel motius exposats a l'apartat 5.5.3

97 Si s'utilitzessin, per exemple, dos temporitzadors amb freqüències de repetició múltiples l'àrea de cobertura temporal es veuria reduïda en el pitjor dels casos a l'interval menor al repetir-se el desfase temporal dels esdeveniments durant tota la prova.

98 Entorn al doble de la pitjor latència en proves i seguint les regles aconsellades per la OSADL segons es descriu a l'apartat 5.5.2

Cal senyalar que mentre que l'objectiu de tots els temporitzadors excepte l'últim és de registrar el temps d'atenció del sistema operatiu, és a dir la latència d'aquest i que inclou en termes generals: latència de resolució de rellotge -segons precisió-, latència d'interrupció -des de que es produeix l'esdeveniment fins que es marca la tasca com executable- i latència de planificació -a partir de que està marcada com executable fins que s'inicia la seva execució. Pel que fa al últim temporitzador pretén mesurar la latència d'atenció per una tasca sense prioritats de temps real, hipotèticament la itneracció humana amb la interfície home-màquina.

5.10.3 Mesura del rendiment

En la mesura del rendiment s'utilitza *bunzip* que realitza la descompressió del codi font del nucli de Linux versió 3.2.32.[99]L'execució d'aquest programa es realitza mitjançant la comanda *time* instrumentada per a que reculli les següents mètriques:

- percentatge de processador utilitzat per l'aplicació durant la seva execució
- segons transcorreguts durant l'execució
- segons transcorreguts en espai d'usuari
- segons transcorreguts en nucli
- segons transcorreguts en espera de planificació
- nombre de canvis de context voluntaris
- nombre de canvis de context involuntaris
- nombre de fallades de pàgina menors
- nombre de fallades de pàgina majors

La mesura es realitza per a dos instàncies executades paral·lelament sobre dos sistemes d'arxius allotjats en dispositius físics diferents: el propi disc dur on està instal·lat el sistema operatiu i l'espai de *swap* i una unitat *flash* connectada al port USB.

5.10.4 Entorn de càrrega

S'executen diferents aplicacions de càrrega que estressen el processador, la memòria, el disc i les funcions de xarxa. El detall de les aplicacions executades i els seus paràmetres és pot trobar a l'apartat 8.6 de l'annex i es resumeix de la següent manera.

5.10.4.1 *stressmem*

Execució cada 10 segons de 7 fils amb un desfase de 250ms reservant cadascun d'ells 128MB de memòria. Paral·lelament, temporització de 12 segons al final dels quals es força (*kill*) la finalització del procés per evitar allaus descontrolats de *swapping* tal com s'ha detectat a les proves d'ajust.

5.10.4.2 *stressdisc*

Execució alternativa de *iozone* per a la lectura seqüencial sobre un fitxer de 1GB i per a la lectura i escriptura aleatòria de un fitxer de 300MB amb un interval de 300ms de temps d'accés per evitar la congestió del sistema i bloqueig de la resta d'activitats. Paral·lelament, execució de *du* sobre l'arbre complet del sistema d'arxius on es realitza la prova.

5.10.4.3 *stressnet*

Execució paral·lela cada 20 segons de *netperf* i *ab* de manera que es produeix, per un costat, un allau de peticions UDP i per un altre la creació de 20 processos concurrents que realitzen alhora

99 És pot descarregar a *kernel.org*.

peticions al servidro web allotjat en el propi entorn on es realitza l'avaluació. En ambdós casos es genera una alta càrrega d'interrupcions i canvis de context.

5.10.4.4 *stresssched*

Execució de *hackbench* per a la creació, comunicació a parells i destrucció de 400 tasques assolint un estrès del planificador notable i un increment notable dels canvis de context.

5.10.5 Balanceig de les càrregues

Per al balanceig de les càrregues, totes elles executades amb polítiques de planificació repartiment just, s'ajusta mitjançant la comanda *nice* que permet seleccionar el paràmetre de planificador del mateix nom. L'ajust es realitza de manera experimental, executant totes les càrregues alhora, tal com es realitzarà a la prova final i monitoritzant l'ús que en fan dels recursos amb *vmstat*, *iostat*, *pidstat* i *htop*. El detall d'aquest ajust es pot trobar, de nou, a l'apartat 8.6 de l'annex.

5.11 Assoliment del temps real

Per a l'anàlisi de l'assoliment del temps real s'utilitzen els histogrames generats a partir dels registres de latència per a diferents configuracions de nucli i diferents freqüències d'esdeveniment així com els descriptors estadístics triats per aquestos mateixos. En aquest segon cas, es realitza un anàlisi del compliment dels terminis pel que fa al temps real *soft* i *hard real-time* suposant un criteri d'èxit donat i utilitzant el percentil 99.999% i el pitjor cas respectivament. Finalment s'analitza el determinisme de cada configuració de nucli utilitzant la desviació estàndard.

5.11.1 Anàlisi de latència a diferents freqüències

En aquesta secció s'analitzen els histogrames obtinguts amb el registre de latències a diferents freqüències. Per aquests tenim les següents dades que ens permeten veure la distribució de mostres recollides:

- Variables d'entrada:
  - Interval programat del temporitzador: 500us, 1100us, 2300us, 111ms, 270ms per a tasques amb prioritats de temps real i 550ms per a una tasca sense prioritats de temps real. Representat per quatre histogrames representats en una sola gràfica.
  - Quatre configuracions de preempció de nucli: no preempt, voluntary, preempt i rt. Representades en quatre gràfiques diferents.
- Variables de sortida:
  - Latències registrades. Representades en dos eixos X/Y latència (X) i freqüència de repetició (Y) d'aquesta latència.

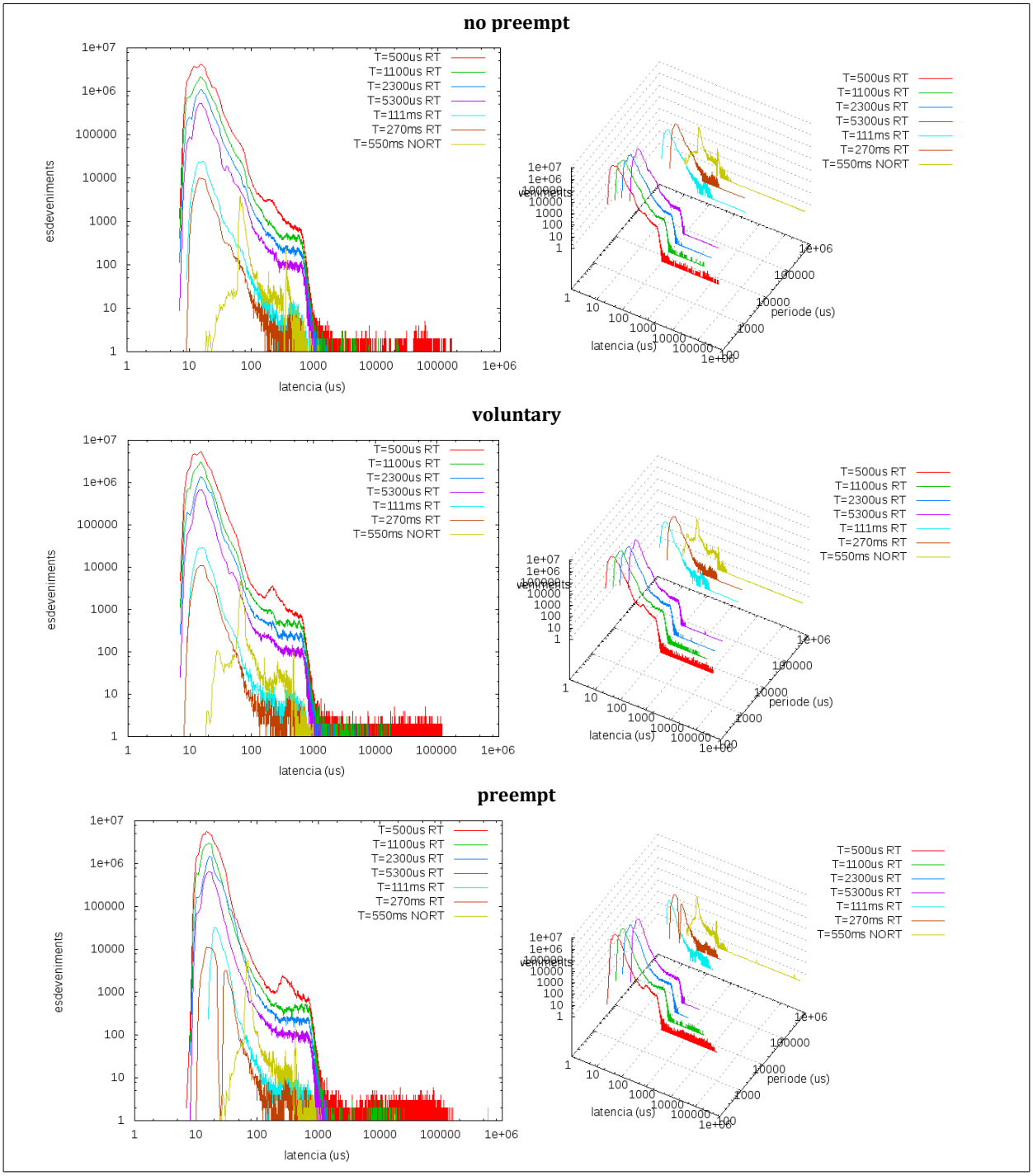
En aquestes es pot observar dos distribucions ben diferenciades: les de les configuracions de nucli sense el pedaç RT i la configuració del nucli que inclou el pedaç RT. Mentre que les primeres no es diferencien notablement en la distribució, tot i que lleugerament en els pitjors casos, la segona té una distribució menys dispersa que la resta.

5.11.1.1 *Nuclis sense el pedaç RT*

Les latències recollides, pel que fa als processos amb prioritat de temps real, estan en l'ordre dels 10us fins a 1 segon i estan distribuïdes majorment en dos distribucions amb forma de campana entre 10us i 100us i 100us i 1000us. Les pitjors latències disminueixen progressivament a mida que augmenta el nivell de preempció per a totes les freqüències programades excepte la de 500us i més

notablement per les freqüències baixes. No s'aprecia massa diferència en la distribució de dades de la tasca sense prioritat de temps real excepte pel que fals pitjors casos.

Figura 50: Histogrames de latència de nuclis sense el pedaç RT

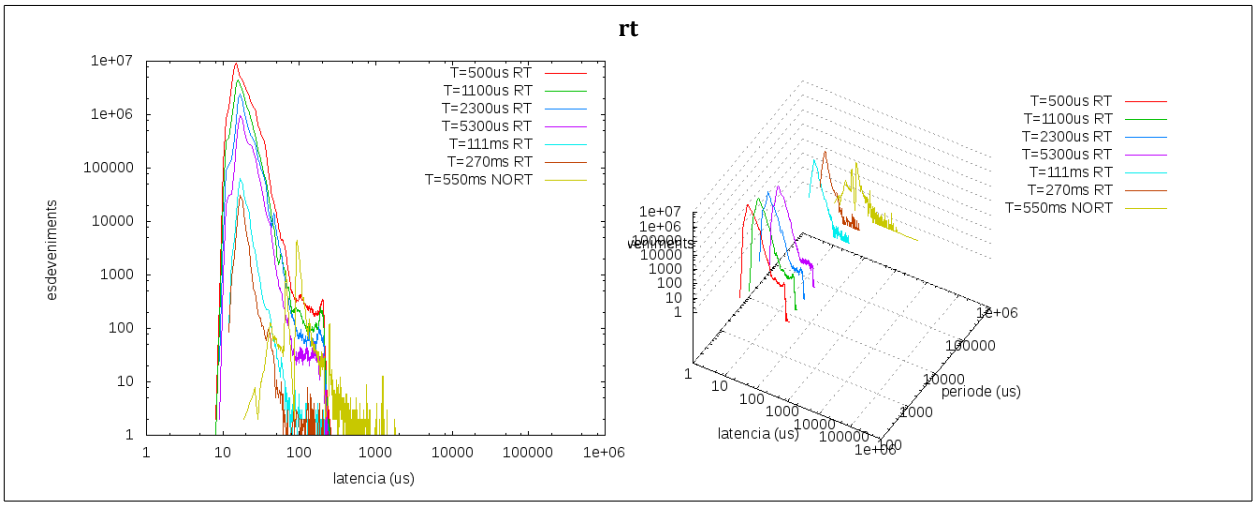


5.11.1.2 *Nucli amb el pedaç RT*

En el nucli amb pedaç RT, per contra, s'aprecia una millora notable tant pel que fa a la distribució de les latències, menys dispersa, com pel que fa als pitjors temps, molt propers a la fi de la segona campana. En aquest cas la dues campanes, pel que fa als processos amb prioritat de temps real,

estan en l'ordre entre 10us i 90us, la primera, i 90us i 200us, la segona. També s'aprecia una millora per al procés sense prioritats de temps real.

Figura 51: Histogrames de latència de nuclis amb el pedaç RT



5.11.2 Compliment de terminis

Quant al compliment dels terminis s'analitzen els percentils més alts recollits en les següents taules i s'estableix un criteri d'èxit de manera que aquests es consideren satisfets sempre que la latència estigui per sota un 75% del interval de temps programat (en verd)[100]. Per altra banda es consideren dos tipus de requeriments de temps real: *soft-real time*, en el que l'incompliment d'un termini no invalida el sistema però li resta valor i on el criteri d'èxit s'estableix en el percentil 99,999%,[101] i *hard-real time*, en el que l'incompliment del termini l'invalida, és adir, utilitzant el pitjor cas.

5.11.2.1 Soft-real time

Analitzant els resultats pel percentil 99,999% es pot observar com la configuració sense preempció podria ser viable per a períodes no massa exigents, del ordre de les centenes de milisegon. A mida que augmenta el nivell de preempció, augmenta també, i de manera notable, les exigències que es podrien veure satisfetes fins al ordre dels milisegons. No és però fins aplicar el pedaç RT quan assolim l'èxit per a totes les freqüències provades i s'arriba a nivell de centèsimes de microsegon.

Taula 16: Taula de latències en el percentil 99,999% (1 mostra de cada 10^5)

1 de 10^5 99,999%	500RT	1100RT	2300RT	5300RT	111000RT	270000RT	550000NORT
no preempt	123453	14713	3886	3627	982	1293	999787
voluntary	104840	13582	2897	2705	6268	884	992231
preempt	134468	16451	1120	1127	1113	1072	998117
rt	211	212	213	213	212	212	8081

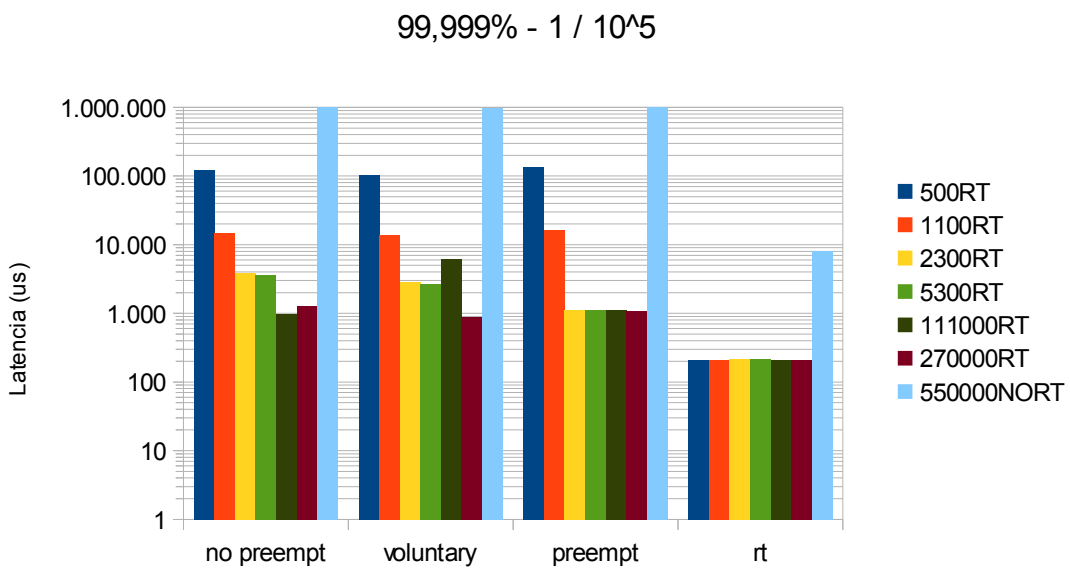
Si s'extreu una gràfica d'aquests valors es pot observar la magnitud de l'impacte en els nuclis sense el pedaç RT pel que fa a l'empitjorament de la latència a mida que la freqüència de peticions creix. En el cas dels processos sense prioritats de temps real aquesta pota arribar fins a un segon.

Per altra banda, el comportament del nucli amb el pedaç RT sota diferents freqüències d'esdeveniment és totalment uniforme i no presenta diferències. A més, la tasca sense prioritats de

100 En consideració que un cop atesa la tasca aquesta encara ha d'executar la lògica de procés implementada.
101 Equivalent a un cas de fracàs de cada 10^5.

temps real també es veu beneficiada per la major preemptibilitat del nucli veient disminuïda la seva latència fins a l'odre de les desenes de milisegon.

Figura 52: Gràfica de latències en el percentil 99,999% (1 mostra de cada 10^5)



5.11.2.2 *Hard-real time*

Analitzant els resultats pel pitjor dels casos es pot observar que per a la majoria de freqüències mostrejades no hi ha massa diferència entre la configuració sense preempció i la configuració amb preempció voluntària. És a dir, podem dir que aquesta segona configuració es prou bona per a la majoria dels casos però que en cap circumstància garanteix res. En canvi, en la configuració amb preempció implícita es pot observar un guany de un ordre de magnitud pels períodes per sobre del milisegon i arribant a satisfer latències entorn al milisegon per períodes de centenes de milisegons.

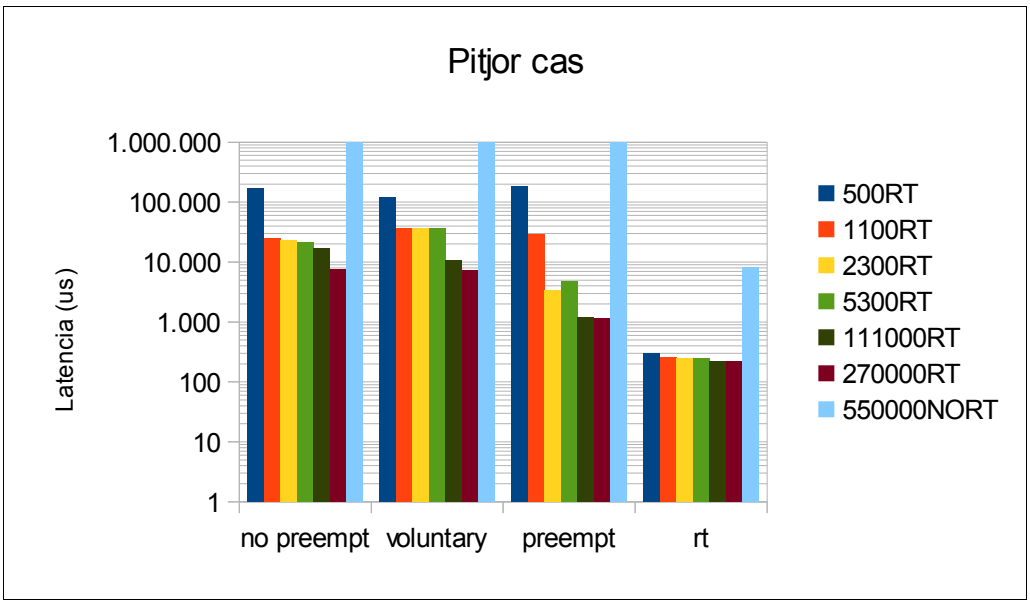
Per altra banda, únicament el nucli amb el pedaç RT aplicat, i amb la prioritat dels processos de temps real per sobre de l'execució de bona part de codi les interrupcions, és capaç de garantir el criteri d'èxit establert mostrant una uniformitat pel que fa al comportament sota diferents exigències i arribant a latències del ordre de les centenes de microsegon.

Taula 17: Taula de latències de pitjor cas (en microsegons)

Pitjor cas	500RT	1100RT	2300RT	5300RT	111000RT	270000RT	550000NORT
no preempt	169676	24352	23349	21322	17270	7616	999787
voluntary	122721	36891	36831	36372	10721	7192	992231
preempt	183908	28479	3350	4666	1202	1153	998117
rt	295	257	247	247	218	217	8081

Si s'observa la gràfica extreta d'aquestes es pot apreciar en comparació amb la del 99,999% i per a les modalitats sense preempció i preempció voluntària com la resposta en el pitjor cas és més uniforme fet indicatiu que una prova més curta hagués pogut estar enganyosa. Per altra banda el nuclis preempt i rt presenten resultats força i molt semblants respectivament, fet que apunta a una diferència menor entre el pitjor cas i el gruix de la mostra.

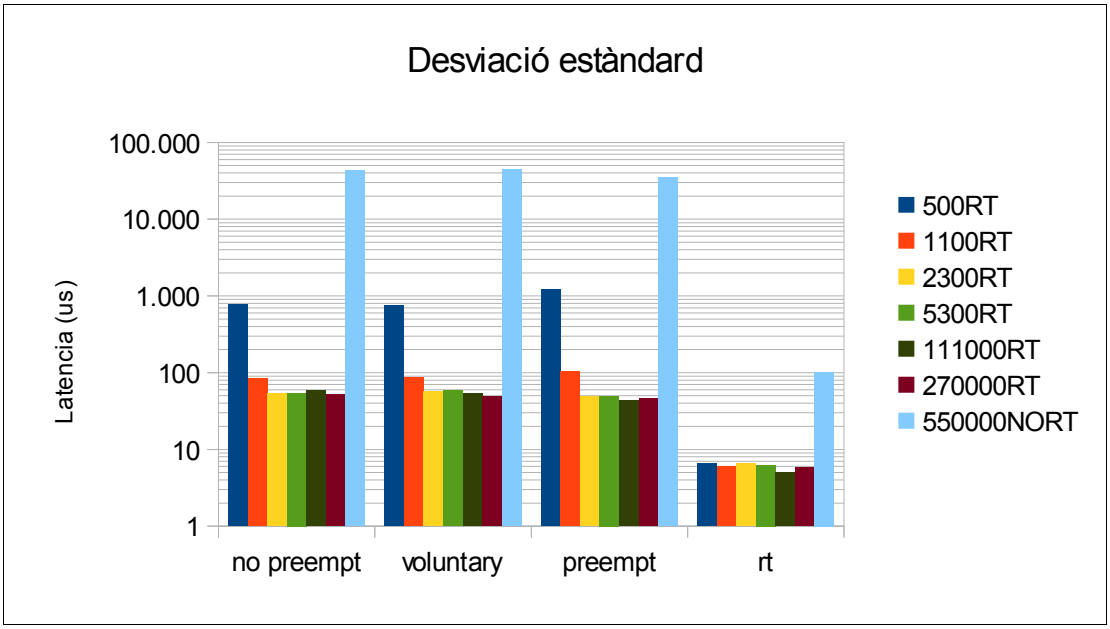
Figura 53: Gràfica de latències de pitjor cas



5.11.3 Determinisme

Pel que fa al determinisme si s'observen els valors de la desviació estàndard es pot apreciar com els nivells pel que fa als nuclis sense el pedaç RT és molt similar, del ordre de les desenes de microsegon arribant fins al milisegon per l'interval més exigent de 500us. Per altra banda, en el pedaç RT aquest valor baixa per sota dels 10us fet indicatiu d'una dispersió de valors molt baixa i per tant d'un gran determinisme.

Figura 54: Desviació estàndard de latències registrades



5.12 Impacte en el rendiment

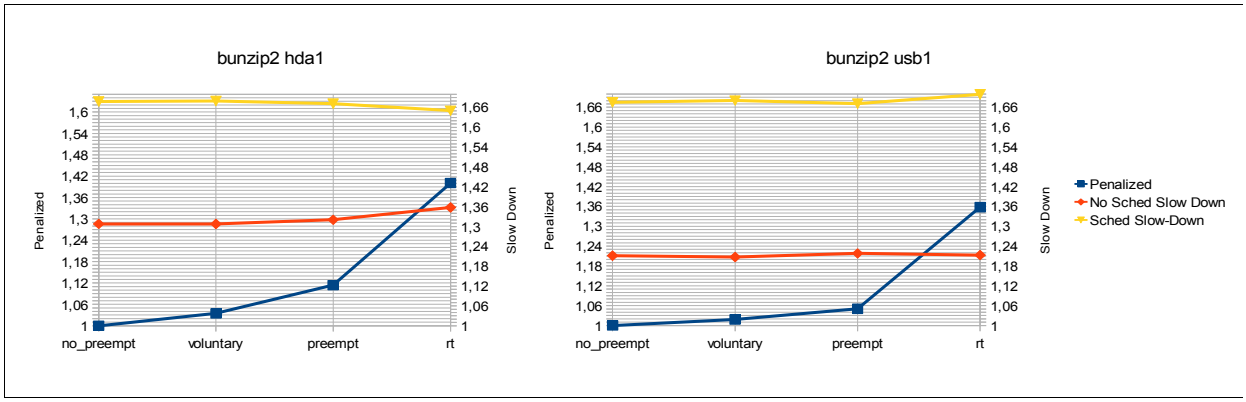
En aquest apartat es realitza s'examinen les dades recollides en la mesura del rendiment per mirar d'establir si existeix una davallada de rendiment produïda per l'assoliment del temps real i quina magnitud podria tenir aquesta. Per això, s'estudia el comportament de les mètriques de rendiment escollides -penalització i ralentització- tant pel que fa a la prova realitzada sobre el disc del sistema com per la prova realitzada sobre un disc en USB i s'intenta determinar si part d'aquesta penalització podria estar induïda per factors no directament relacionats amb la implementació del sistema i, en aquest sentit, no seria completament atribuïbles a l'assoliment del temps real.

A continuació, es realitza l'anàlisi de la resta de mesures obtingudes tan pel que fa a la seva magnitud, a través de la mitjana geomètrica de les repeticions de prova realitzades, com pel que fa a la seva relació amb la pèrdua de rendiment del sistema, mitjançant el coeficient de correlació de Pearson obtingut per a totes les proves. [102]

5.12.1 Anàlisi de rendiment

Observant les gràfiques de les mètriques de rendiment obtingudes en les dos proves realitzades es constata com mentre la penalització segueix una progressió lineal creixent pel que fa a les tres primeres configuracions de nucli, en arribar a la que utilitza el pedaç RT es dispara notablement arribant fins un factor de 1,42 -un 42% de penalització- en la prova a disc de sistema per a un factor de 1,36 -un 36% de penalització- en la prova a disc en USB.

Figura 55: Rendiment per a diferents configuracions de nucli



Quant a la progressió de la proporció dels temps en espera en referència al temps d'execució (*slowdown*), s'observa com en les dues primeres configuracions de nucli es manté molt similar mentre que a partir d'aquest punt segueix línies divergents en les dues proves realitzades: decreixent pel que fa a la ralentització deguda al planificador i creixent per altres efectes en la prova a disc de sistema i creixent per planificador i decreixent per altres efectes en la prova a disc en USB. A més, mentre en les dues proves la ralentització provocada pel planificador es manté

102 Mesura del grau de dependència entre dos variables. Pot prendre valors en l'interval [-1..1]. El signe denota la direcció de la dependència (directa o inversa) i la proximitat als extrems és indicativa de una major relació. Tot i que la interpretació d'aquest coeficient esta estretament lligada al context i significat de les variables alguns autors suggereixen la següent relació (valor absoluts): entre 0,1 i 0,3 dependència baixa, entre 0,3 i 0,5 dependència mitja, entre 0,5 i 1 dependència alta.

entorn al mateix factor (1,66) la que és efecte d'altres causes és força menor en la prova realitzada en disc USB, entorn 1,24 en vers 1,36.

D'aquestes diferències se'n dedueix que part de la davallada de rendiment en la prova realitzada sobre disc de sistema pugi estar motivada per factors aliens a la pròpia implementació del nucli, si més no de manera directa, i que pugi tenir relació amb algun efecte provocat per la congestió E/S del disc de sistema al allotjar, també, la partició de *swap*.

En aquest aspecte, hi ha la impressió que la prova realitzada a disc USB ha estat capaç d'aïllar millor possibles efectes externs, tot i que possiblement provocats de manera indirecta, a la implementació del nucli i, en aquest aspecte, sigui una mesura més fiable per establir el compromís assumit en l'assoliment d'una millor resposta en temps real.

Per tant, s'estableix hipotèticament que la diferència en la penalització del rendiment entre una i altre prova no és pas efecte de cost en l'assoliment del temps real i que, consegüentment, el pes d'aquest és limitaria, en principi, a un 30% del total mentre el 10% restant, només detectat en la prova executada sobre el disc de sistema, correspondria a un efecte secundari, possiblement relacionat amb la congestió dels camins E/S.

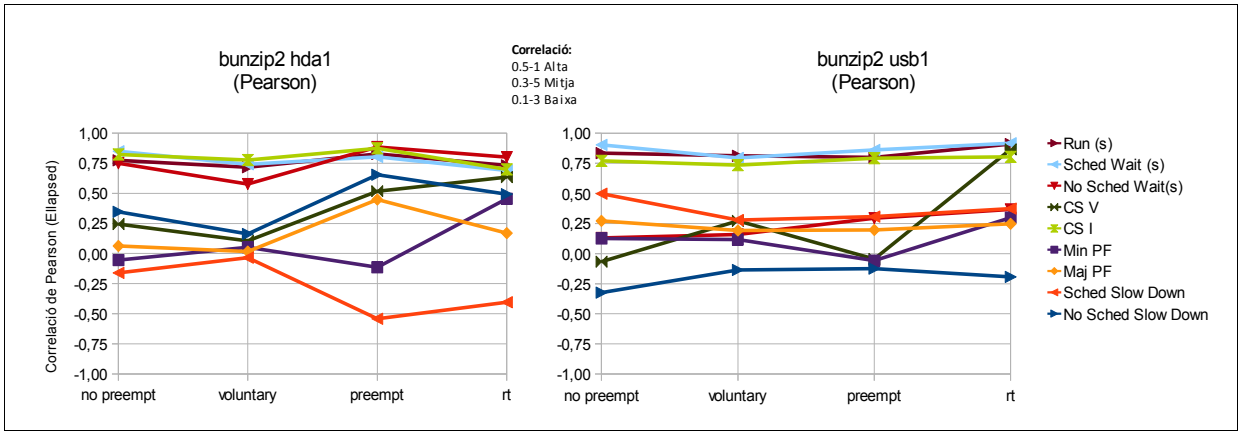
5.12.1.1 *Dependència entre variables*

Examinant el coeficient de correlació de Pearson, calculat en relació amb la penalització mesurada, s'estableix el grau de dependència lineal de cadascuna de les variable mesurades respecte a la penalització de rendiment. Aquest grau, tot i que no indica necessàriament una relació causa-efecte, dóna una idea del pes hipotètic de cadascun dels factors mesurats i, en aquest aspecte, serveix per vehicular l'anàlisi dels mateixos per intentar establir, d'aquesta manera, relacions entre aquests valors i la davallada de rendiment.

Taula 18: Llegenda de mesures del rendiment

Run	segons execució efectiva de la tasca	**Penalized**	proporció de penalització
Sched. Wait	segons en espera de planificació	**Sched Slow-Down**	proporció temps en espera planificador
No Sched. Wait	segons en espera per altres causes principalment accés E/S	**No Sched Slow-Down**	proporció temps en espera altres
Min PF	fallades de pàgina menors	**CS V**	canvis de context voluntaris
Maj PF	fallades de pàgina majors	**CS I**	canvis de context involuntaris

Figura 56: Coeficient de Pearson en vers el rendiment



Observant les gràfiques traçades per aquest coeficient s'observen diversos trets. En primer lloc, que els temps d'espera en cua del planificador (*sched. wait*) afecta, evidentment, al rendiment de l'aplicació però que també ho fa el temps d'execució (*run*), ambdós amb un coeficient entre 0,5 i 1. Això sembla indicar que part de la penalització no és només fruit de la hipotètica expulsió de la tasca per altres tasques de major prioritat si no que també ho és de la pròpia execució que és menys eficient. A més, aquest és un tret que té un comportament uniforme pel que fa a les dues proves realitzades.

En segon lloc, que el temps en espera per raons alienes al planificador (*no sched. wait*), suposadament espera E/S, està íntimament lligat al rendiment pel que fa a la prova realitzada sobre disc de sistema -entre 0,5 i 1- però que, en canvi, no sembla tenir tal repercussió pel que fa a la realitzada sobre disc en USB -entre 0 i 0,5. Aquest tret vindria a reforçar la hipòtesi anteriorment formulada que la prova de rendiment en disc de sistema s'ha vist afectada per la congestió a disc i que, per tant, la penalització de rendiment obtinguda en aquesta prova no es podria atribuir directament al assoliment del temps real.

En tercer lloc, que el nivell de fallades de pàgina majors (*Maj PF*) -entorn 0'5- ha tingut una influència considerable en el rendiment pel que fa a la prova realitzada en disc de sistema per a la configuració *preempt*. Paral·lelament, s'observa com la proporció de temps en espera deguda al planificador (*sched. slow down)* -entorn -0,5- sembla haver tingut una influència positiva en el rendiment -ralentització inversament proporcional a penalització- mentre la proporció de temps en espera per altres causes (*no sched. slow down)* -entre 0,5 i 0,75- n'ha augmentat el seu.

Aquest fet incongruent, influència positiva la ralentització sobre el rendiment, s'explica perquè en les repeticions de la prova en que la proporció de temps en espera de planificador ha baixat mentre el rendiment ha pujat és simplement degut a que l'augment en el temps en espera per la resta de causes ha pujat i que, consegüentment, cal atribuir a aquest el pes de la penalització. Aquest fet explicaria la diferència de resultats per aquesta configuració de nucli entre la prova realitzada en disc de sistema, en la que trenca la linealitat de la seva progressió, i la realitzada en disc extern USB, en el que la manté, i reforçaria, de nou, la hipòtesi que la prova realitzada en disc de sistema ha estat afectada per afectes no directament relacionats l'assoliment del temps real.

En quart lloc, que la influència dels canvis de context involuntaris en les dues proves i les quatre configuracions provades -entorn 0,75- sembla tenir una influència notable en el rendiment, fet comprensible, per altra banda, pel cost en la realització d'aquest procés. Per contrapartida, el pes dels canvis de context voluntaris sembla no tenir massa pes -entre -0,25 i 0,25- excepte en els següents casos: configuració *preempt* en prova a disc de sistema, pel ja exposat, i nucli amb el pedaç RT incorporat, per raons desconegudes.

5.12.1.2 *Valor de les variables*

Tot seguit s'analitza el valor de les variables a través de la mitjana geomètrica[103] (taula 19) així com la normalització d'aquest mateix valor en l'interval [0..1][104], que serveix per poder comparar la evolució de variables de diferent magnitud quant a la proporció de creixement o decreixement segons la modalitat de preempció (figura 57).

Examinant aquests valors el primer que crida la atenció és el gran nombre de canvis de context voluntaris (*CS V*) registrats en les proves executades amb el nucli amb el pedaç RT, tant pel que fa a la prova executada sobre disc en USB -entorn a 27 cops més- però sobretot pel que fa a la executada sobre disc de sistema -entorn 60 cops més. L'origen en l'augment tant desproporcionat en aquesta

103 Menys sensible als valors extrems que la mitjana aritmètica.
104 Mitjançant la normalització anomenada *ranging*, que és calcula de la següent manera: $x' = (x - min)/(max - min)$

mesura per al pedaç RT podria estar en l'efecte provocat per la preempció de seccions crítiques esmentat esmentat en l'apartat 4.5.4.1 i que bàsicament consisteix en què la possibilitat de planificar una nova tasca durant la execució d'una secció crítica de la anterior, aspecte que pels resultats en la latència sembla imprescindible per l'assoliment de cert grau de determinisme, comporta la probabilitat de què la nova tasca planificada bloquegi la seva execució en algun punt futur degut a que requereix entrar en la mateixa secció crítica protegida per el *lock* adquirit amb anterioritat.

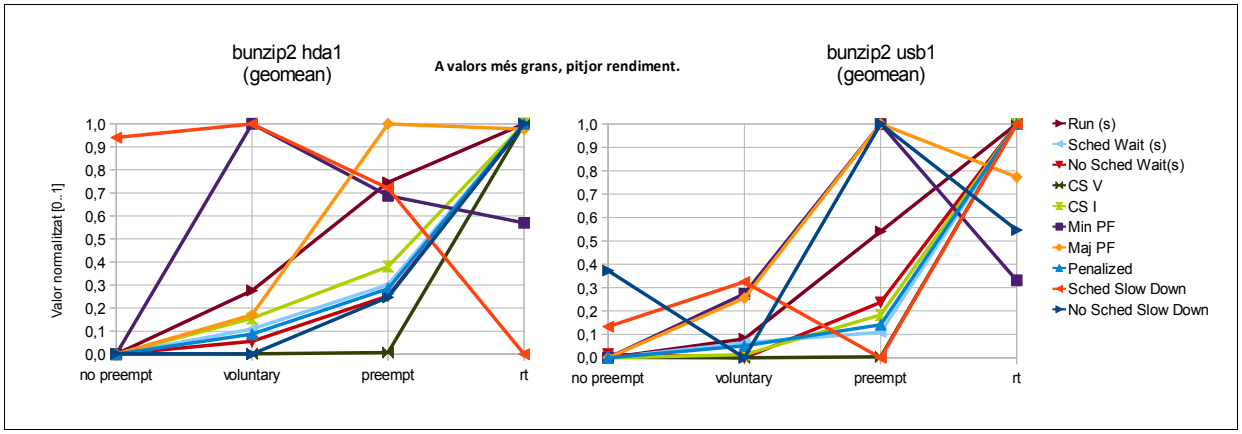
Així doncs, si, per exemple, l'aplicació que realitza la descompressió sobre disc de sistema es interrompuda durant l'execució d'alguna secció crítica relacionada amb l'accés E/S per l'aplicació que realitza la descompressió sobre disc USB cap la possibilitat que aquesta segona hagi de bloquejar-se en algun punt d'execució futur si requereix l'execució de la mateixa secció crítica relacionada amb l'accés E/S. Aquest aspecte queda solucionat implícitament en la resta de configuracions de nucli avaluades al no permetre la preempció de seccions crítiques però es manifesta en la configuració amb el pedaç RT en permetre-la. [105]

Taula 19: Mitjana geomètrica de les mesures de rendiment

HDA1	% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	No Sched Wait(s)	CS V	CS I	Min PF	Maj PF
no preempt	7,94	997,30	75,10	8,41	83,54	627,44	284,26	289	393790	1073,87	3,49
voluntary	7,75	1031,88	76,16	8,72	84,90	650,82	294,12	332	402019	1079,27	3,91
preempt	7,37	1110,87	76,84	10,35	87,20	692,23	329,33	481	413975	1077,58	5,94
rt	5,74	1396,87	77,09	11,34	88,45	842,74	462,68	28418	446855	1076,95	5,88

USB1	% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	No Sched Wait(s)	CS V	CS I	Min PF	Maj PF
no preempt	10,82	674,67	72,66	3,98	76,67	454,91	141,48	313	360687	1076,26	3,01
voluntary	10,19	686,87	72,87	4,01	76,90	466,70	140,67	298	361255	1082,48	3,54
preempt	10,54	708,70	73,24	4,95	78,21	475,14	153,06	330	368315	1099,05	5,11
rt	8,19	916,13	73,66	5,84	79,53	640,03	192,99	8879	402418	1083,80	4,63

Figura 57: Mitjana geomètrica normalitzada de les mètriques de rendiment



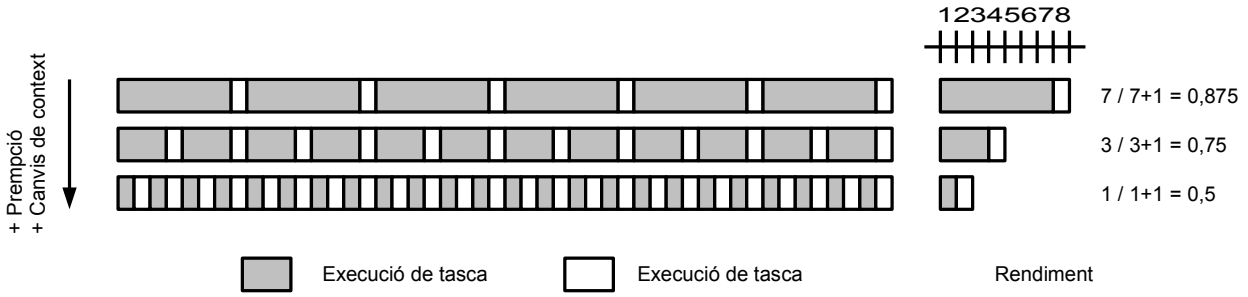
Pel que fa a la resta de mesures, senyalar, per una banda, que l'evolució de la majoria d'aquestes és consistent en les dues proves realitzades i amb els efectes fins ara observats en el coeficient de

105 En aquest aspecte cal senyalar que durant la realització d'aquest projecte s'han pogut constatar moviments en la branca RT encarats a solucionar aquesta contrapartida fonamentats en la introducció d'un nou mode de preempció anomenat *lazy preemption* utilitzat en les tasques sense prioritat RT (SCHED_OTHER) que no permet la preempció per part d'una tasca que no sigui de temps real d'una secció crítica en execució.[GLE12]

Pearson i l'anàlisi de les mètriques de rendiment. És a dir, es detecta una progressió lineal de penalització en les tres primeres configuracions, excepte en la configuració *preempt* de la prova realitzada sobre disc de sistema que augmenta lleugerament pels efectes ja exposats amb anterioritat. En canvi, aquests mateixos resultats empitjoren notablement per la configuració amb el pedaç RT. En aquest aspecte es podria dir que la penalització avança progressivament amb la millora del temps de resposta en el sistema però que es dispara notablement en assolir l'últim estadi amb el pedaç RT.

Per altra banda, es constata com el temps d'execució (*run*) creix de manera considerable amb les diferents modalitats de preempció. Aquest efecte podria explicar-se pel cost produït en l'augment dels punts de preempció del sistema[106], la disminució en la proporció de longitud de codi executat per canvi de context (figura 58) o a efectes secundaris en l'augment de la preempció com poden ser una major pol·lució de la jerarquia de memòries (RAM i memòria cau)[107]. Observant, tanmateix, el pes d'aquest increment pel que fa a l'execució de codi de nucli (*sys*) i de codi d'usuari (*user*) pel que fa a les dues últimes modalitats de preempció (*preempt* i *rt*) es diria que aquest pes recau més en els efectes secundaris esmentats que no pas en un increment directe en el cost de la implementació.

Figura 58: Increment del nombre de canvis de context



Finalment, senyalar que la reducció d'influència del temps d'espera en cua de planificador observat en el coeficient de Pearson es verifica en observar com la ralentització deguda a aquest efecte decreix a mida que augmenta el nivell de preempció, així com que la ralentització provocada per la resta de causes creix de manera complementaria, única i exclusivament per les proves realitzades en el disc de sistema. Aquesta observació, per tant, vindria a reforçar la idea que la penalització constatada en aquesta prova en concret és producte, per un costat, del canvi d'implementació de les diferents configuracions amb major nivell de preempció però, també, per un altre d'un efecte de la congestió de l'accés E/S.

5.13 Resum de resultats

En aquest apartat es mostra una taula de resum dels resultats obtinguts tan pel que fa al a millora del temps real (latència) com per la davallada de rendiment (penalització) en les quatre configuracions de nucli avaluades.

106 En aquests punts cal realitzar una sèrie de comprovacions com es pot observar a les figures 18 i 19 dels apartats 4.4.1 i 4.4.2 respectivament que tot i no tenir un gran pes poden arribar a ser significatives en ser repetides en diversos punts de l'execució del nucli.

107 Aquest efecte s'explica perquè l'augment en la preempció comporta inevitablement la interrupció d'activitats que queden sense finalitzar els blocs de codi de les quals no es veuran beneficiats dels efectes de la jerarquia de memòries al perdre la seva localitat temporal. És a dir, en emprendre la seva activitat de nou, i al haver transcorregut un temps des de la seva interrupció, els algorismes de reemplaçament de la memòria cau o el sistema d'intercanvi de memòria principal i auxiliar (*swap*) poden haver decidit durant la seva interrupció expulsar el seu codi per prioritzar a un altre de major localitat temporal.

5.13.1 Latència

Quant a la latència es mostren les pitjors figures obtingudes per a les diferents freqüències d'esdeveniment analitzades per a prioritats de temps real així com un freqüència d'esdeveniment de tall que estableix a partir de quin punt la configuració de nucli testada ha satisfet el criteri d'èxit establert en les proves realitzades (latència < 75% del període). Finalment, es mostra la millora de cada nivell de preempció en referència al nivell de preempció anterior, a major factor millora més gran.

Taula 20: Resum de resultats de latència obtinguts

Configuració	Latència pitjor cas	Latència 99'999%	Freqüència de tall pitjor cas	Freqüència de tall 99,999%	Factor de millora pitjor cas	Factor de millora 99'999%
No preempt	169ms	123ms	270ms	110ms	-	-
Voluntary	122ms	104ms	270ms	5,3ms	x1,38	x1,18
Preempt	183ms	134ms	110ms	2,3ms	x0,67	x0,78
RT	295us	213us	500us	500us	x623,42	x629,11

5.13.2 Rendiment

Pel que fa al rendiment es mostren els resultats de la mitja geomètrica de les mètriques escollides en la avaluació de la prova en disc USB: penalització (*penalized*), en referència al model sense preempció de nucli (*no preempt*), i ralentització, deguda al planificador (*scheduler slow down*) i altres (*no scheduler slow down*). Per a qualsevol de les tres mètriques, a menor factor millor rendiment.

Configuració	Penalized	Scheduler Slow Down	No Scheduler Slow Down
No preempt	x1,000	1,211	1,675
Voluntary	x1,018	1,207	1,680
Preempt	x1,050	1,218	1,671
RT	x1,358	1,213	1,699

5.13.3 Consideracions

En la interpretació d'aquests resultats s'han de tenir en compte les següents consideracions:

- Els resultats són dependents del maquinari i càrrega utilitzada així com de la distribució de Linux, versió de nucli i pedaç de RT avaluats. En aquest aspecte han de servir per comparar els resultats entre diferents configuracions de nucli i no pas per establir figures útils per a altres sistemes i configuracions.
- Els resultats de rendiment no inclouen la prova de disc a sistema per consider-la menys fiable pel que fa a l'atribució de la davallada de rendiment a l'assoliment del temps real pels efectes detectats en la congestió E/S.
- Els resultats de la prova realitzada amb la configuració *preempt* poden estar influenciats per una certa sobrecàrrega en la congestió E/S no observada en la resta de configuracions. En aquest aspecte, la pitjor resposta en temps real no hauria de ser considerada, en principi, com un efecte directe en la major preempció tot i que, possiblement, si que ho és un de indirecte, fruit d'una gestió menys eficient dels recursos per part del sistema operatiu.

6 Conclusions

En aquest projecte hem avaluat les capacitats en l'assoliment del temps real del sistema operatiu Linux. Per això ha estat necessari definir quines són les característiques que conformen el temps real, examinar els components del sistema operatiu que en poden estar involucrats, investigar quin són els avenços i alternatives disponibles per la millora del comportament de Linux en aquest plà i, finalment , avaluar les diferents opcions per mirar de determinar fins a quin punt es possible el seu assoliment i quin contrapartida comporta pel que fa al rendiment.

Quant a les característiques del temps real hem establert que l'assoliment del mateix requereix del compliment d'uns terminis temporals prefixats i que aquest, alhora, depèn de la latència o temps d'atenció del sistema operatiu. A més, hem determinat que aquest requisit només és abastable quan el sistema es comporta de manera determinista, garantint, en els pitjors dels casos, un temps de resposta acotat. Finalment, hem classificat el temps real en diferents categories segons el grau d'exigència en el compliment dels terminis: *hard real-time* si l'incompliment invalida el sistema i *soft-real time* o *firm real-time* si l'incompliment del mateix no l'invalida però li fa perdre valor.

Pel que fa al sistema operatiu, i més concretament Linux, hem estudiat la seva arquitectura basada en la divisió en espais, nucli i usuari, i en la preempció, procés pel qual s'interrompt una tasca per donar pas a una altre. En referència als espais d'execució, hem vist que el nucli és un mode elevat amb accés als dispositius de maquinari i que l'espai d'usuari, on s'executen les aplicacions, accedeix a aquestos a través del nucli mitjançant les anomenades crides de sistema. Hem revisat aquells components que poden influir en el temps d'atenció com són el planificador, les interrupcions, el rellotge del sistema i, principalment, la capacitat de preempció del nucli. A més, hem vist com la sincronització, mecanisme utilitzat per evitar l'accés concurrent a les seccions crítiques, zones de codi on es manipulen les estructures de control del sistema, introdueix dificultats afegides com són les inversions de prioritat o els camins de codi no preemptibles. Finalment, hem estudiat alguns components que, dissenyats per augmentar la eficacia del sistema, introdueixen un cert grau de indeterminisme en el seu comportament com són la jerarquia de memòries o la planificació E/S.

A continuació, hem examinat els diferents avenços en determinisme i temps de resposta que ha experimentat Linux en la última década i hem analitzat les diferents alternatives que ofereix a dia d'avui. En aquest aspecte, hem après que inicialment el nucli no era preemptible, que els primers esforços van estar fonamentats en la introducció de punts de preempció en diferents localitzacions del codi tot i que, més endavant, és va optar per estendre la preempció a tot l'espai de nucli, a excepció de les seccions crítiques.

També, hem vist com aquestes millores en la preempció, desenvolupades inicialment com a *patches* han vist el seu camí a la línia principal del nucli i que s'han materialitzat en tres modalitats de preempció diferents: sense preempció (NO PREEMPT), preempció implícita (VOLUNTARY) i nucli preemptible (PREEMPT). Finalment, hem conegut l'existència del pedaç RT, on, de manera paral·lela al *mainline*, es desenvolupen una sèrie d'avanços encarats a la millora del temps real a Linux com poden ser l'execució d'interrupcions en context de procés per tal de què sigui preemptibles o l'implementació de *mutex* amb herència de prioritats que permetin la preempció de seccions crítiques, els dos principals esculls de la línia principal en l'assoliment del temps real.

En la última ètapa del projecte, i amb l'objectiu d'avaluar el grau d'assoliment dels mecanismes analitzats, hem preparat en un entorn de d'avaluació adequat quatre configuracions de nucli, cadascuna amb un dels tres nivells diferents de preempció i una última amb el pedaç RT. Hem establert les propietats del experiment que ens hauria de permetre aquesta avaluació, hem definit les mètriques i descriptors estadístics que hauriem d'utilitzar en el seu estudi, hem modelat una càrrega de treball amb l'objectiu d'augmentar la probabilitat de detectar els pitjors temps de resposta i, finalment, hem executat l'experiment d'avaluació sobre cadascuna de le quatre configuracions preparades.

Un cop finalitzat l'experiment, hem recollit i analitzat les dades extretes i hem pogut constatar l'eficàcia de les diferents modalitats de preempció pel que fa a l'assoliment del temps real, de manera notable la configuració amb el pedaç RT. En aquest aspecte, és aquesta última modalitat la única que ha aconseguit satisfer totes les freqüències d'esdeveniments imposades demostrant una capacitat de reposta en el maquinari utilitzat en l'avaluació per sota dels 300us així com un determinisme molt elevat amb una desviació típica entorn els 6us, és a dir un 2%.

Per contrapartida, en la mesura del rendiment, hem pogut observar els mals resultats d'aquesta configuració amb un 35% de penalització i, a través del anàlisi de les distintes variables recollides durant l'experiment, hem establert la hipòtesi de què és deguda a la gran quantitat de canvis de context voluntaris fruit probablement del bloqueig de la tasca avaluada produït en interrompre una segona tasca durant l'execució d'una secció crítica en la que aquesta hauria adquirit un *lock* sobre un recurs que més endavant necessitaria la tasca bloquejada.

En l'experiment hem pogut constatar, també, la dependència dels resultats al maquinari i nivell de càrrega imposat i en aquest aspecte ens hem trobat amb que, per un costat, una de les modalitats de preempció (*preempt*) donava una resposta totalment inesperada al obtenir pijtors resultats de latència pel que fa al pitjor cas que no pas la modalitat anterior (*voluntary*) en teoria menys eficaç i que, per altre costat, hi havia una diferència significativa en les mesures de rendiment pel que fa a la prova realitzada sobre el propi disc de sistema, on també s'allotja la partició de *swap*, en comparació amb les realitzades sobre un disc extern.

A més, amb l'execució de l'experiment, hem pogut verificar la necessitat de seguir una sèrie de regles en el disseny de les proves i com aquestes haurien de satisfer, com indiquen alguns dels autors experts en la materia, una sèrie de requisits mínims per millorar la fiablitat dels seus resultats. Així doncs, hem comprovat com a diferents freqüències d'esdeveniment així com amb diferents duracions de l'experiment les respostes de latència podien variar significativament.

En definitiva, creiem que el projecte compleix amb el seu objectiu inicial al donar resposta de quines són les diferents alternatives de nucli de Linux enfocades a la millora del temps real i de determinar el seu grau d'assoliment en aquest aspecte així com d'establir les contrapartides que suposa aquesta millora pel que fa al rendiment.

6.1 Línies obertes

En l'execució de l'experiment hem modelat diverses càrregues centrades en el processador, l'ús de la memòria RAM i l'accés a disc principalment. Hem modelat càrregues amb funcions de xarxa tot i que les hem executat en mode local, sense xarxa, per intentar aïllar els efectes aleatòris que aquesta pogués provocar en l'avaluació de les quatre configuracions de nucli. En aquest aspecte, creiem que seria interessant repetir les proves amb diferents carregues, modelar aquestes amb un perfil més proper al que seria una aplicació real i davant de diferentes condicions intentar aïllar i esbrinar

l'origen d'alguns dels efectes trobats, establir quins són productes directes d'una o altre implementació o quins ho són d'indirectes de manera que, tot i no ser un efecte d'aquesta, si que ho són d'una situació provocada per aquesta.

En aquesta línia, creiem que seria interessant, també, esbrinar els origens de les pitjors latencies pel que fa a les configuracions sense el pedaç RT. Per això, es podria utiltizar les funcions de traça de funcions del nucli (FTRACE) que ja s'han habilitat en la preparació de la prova però que finalment no hem tingut oportunitat d'explorar per falta de temps. Un cop esbrinats aquests origens es podria establir si són o no evitables i quines maneres hi haurien d'eliminar-los o, si més no, pal·liar-los.

Pel que fa a l'experiment realitzat, creiem que hi haurien altres proves que serien interessant com podria ser l'avaluació de la modalitat de nucli amb preempció (*preempt*) però habilitant l'execució d'interrupcions en context de procés a través del paràmetre de línia de comandes de nucli *threadirqs.* També creiem d'interés repetir les proves sense el bloqueig de pàgines de memòria (opció -m a *cyclictest*) o sense la desactivació de les modalitats d'estalvi d'energia (interfície */dev/cpu_dma_latency*) així com executar l'experiment durant un període de temps més prolongat, com a mínim 24 hores, tal com recomanen els experts a OSADL o Red Hat.

Durant la preparació de l'experiment hem modificat fins a tres aplicacions amb el propòsit d'adaptar-les a les nostres necessitats. En aquest aspecte, hem introduït a *cyclictest* la possibilitat d'executar aplicacions externes. Tanmateix, hem comprovat com el temps de creació i eliminació d'un procés era excesivament alt i força indeterminat. Creim que seria interessant redissenyar aquesta modificació de tal manera que en comptes d'executar programes externs incorporés una sèrie de funcions que li permetessin modelar certes càrregues. En aquest aspecte, una de les possibilitats seria incloure les funcions de l'aplicació *stress* que han demostrat ser força eficients pel que fa a la generació de càrregues. També es podrien ampliar aquestes funcions per mirar que, a part de modelar càrregues específiques sobre un subsistema determinat, modelessin càrregues complexes a través d'aquestes, per exemple de servidor web o de base de dades.

Hem modificat, també, la utilitat *time* per a recollir la comptabilitat de temps del planificador tot i que hem pogut comprovar que la implementació realitzada només era capaç de recollir aquestes dades pel que fa al procés executat i no tota la sèrie de processos fills que aquest, alhora, pogués executar. En aquest aspecte, es podria millorar la modificació de tal manera que un cop finalitzat el procés executat s'analitzes el seu arbre de processos fills i es recopilés i mostres la seva informació pel que fa als temps de planificador comptabilitzats.

En aquest projecte hem estudiat, analitzat i avaluat les característiques que permeten una millor resposta al sistema operatiu Linux deixant de banda els efectes del maquinari que també tenen un impacte significatiu en aquesta, sobretot pel que fa al determinisme. En aquest aspecte, seria interessant realitzar un estudi més centrat en el maquinari on es podria analitzar en una arquitectura donada els efectes de la configuració de memòria utilitzada, la jerarquia de memòries (memòria cau, MMU, TLB...), la contenció del bus del sistema, el DMA o els algoritmes de predicció de salt de la CPU, per exemple.

Finalment, l'experiment s'ha realitzat sobre un sistema uniprocessador. Es podria repetir també amb un configuració de multiples processadors (SMP). En aquest cas caldria també, però, tenir en compte la migració de processos entre processadors i els efectes en l'establiment de l'afinitat dels processos per a cada una de les CPU. En aquest cas, a més, seria interessant verificar quina és l'escalabilitat dels resultats obtinguts.

7 Bibliografia

[BER10] Berger, Robert. (2010). *Getting real (time) about embedded GNU/Linux*. [en línia]. http://www.embedded.com/design/operating-systems/4204740/Getting-real--time--about-embedded-GNU-Linux [data de consulta:]

[BOH09] Bohmer, Remy. (2009). *HOWTO: Build an RT-application*. [en línia]. https://rt.wiki.kernel.org/index.php/HOWTO:_Build_an_RT-application [data de consulta: 12/01/2013]

[DIS13] DistroWatch. (2013). *DistroWatch Page hit ranking*. [en línia]. http://distrowatch.com/dwres.php?resource=popularity [data de consulta: 13/01/2012]

[DIS12] DistroWatch.com. (2012). *DistroWatch Page Hit Ranking*. [en línia]. http://distrowatch.com/dwres.php?resource=popularity [data de consulta: 09/11/12]

[ELN11] elinux.org. (2011). *High Resolution Timers*. [en línia]. http://elinux.org/High_Resolution_Timers#How_to_detect_if_your_timer_system_supports_high_resolution [data de consulta: 05/11/2012]

[EMD09] Emde, Carsten. (2009). *Path analysis vs. empirical determination of a system'sreal-time capabilities: The crucial role of latency tests*. [en línia]. https://www.osadl.org/fileadmin/dam/presentations/RTLWS11/cemde-path-analysis-vs-empirical-latency.pdf [data de consulta: 13/01/2013]

[GAM00] Gamble, Nigle. (2000). *[PATCH] Latest preemptible kernel (low latency) patch available*. [en línia]. http://lkml.org/lkml/2000/11/22/128 [data de consulta: 14/11/2012]

[GLE12] Gleixner, Thomas. (2012). *[ANNOUNCE] 3.6.4-rt11*. [en línia]. http://thread.gmane.org/gmane.linux.rt.user/9039 [data de consulta: 19/01/2013]

[IEEE01] IEEE. (2001). *1003.1 Standard for Information Technology - Portable Operating System Interface (POSIX)*. USA:IEEE Computer Society Press.

[INT04] Intel. (2004). *IA-PC HPET (High Precision Event Timers) Specification*. [en línia]. http://www.intel.com/content/www/us/en/software-developers/software-developers-hpet-spec-1-0a.html [data de consulta: 07/11/2012]

[INT08] Intel. (2008). *Mobile Intel® Atom™ ProcessorN270 Single Core*. [en línia]. http://www.intel.com/content/www/us/en/mobile-computing/mobile-atom-n270-single-core-datasheet-.html?wapkw=n270 [data de consulta: 07/11/2012]

[INT08] Intel. (2008). *Mobile Intel® Atom™ ProcessorN270 Single Core*. [en línia]. http://www.intel.com/content/dam/doc/datasheet/mobile-atom-n270-single-core-datasheet-.pdf [data de consulta: 12/01/2013]

[ISO07] ISO/IEC JTC1/SC22/WG14 working group. (2007). "6.7.8 Initialization". *ISO/IEC 9899:1999 (The Ansi C Standard C99)*. - :ISO/IEC

[KER11] Kernel.org. (2011). *The Second Extended Filesystem*. [en línia]. https://www.kernel.org/doc/Documentation/filesystems/ext2.txt [data de consulta: 16/01/2013]

[KER12] Kernel.org. (2012). *schedstats*. [en línia]. https://www.kernel.org/doc/Documentation/scheduler/sched-stats.txt [data de consulta: 15/01/2013]

[KER13] Kernel.org. (2013). *Kernel Parameters*. [en línia]. https://www.kernel.org/doc/Documentation/kernel-parameters.txt [data de consulta: 12/01/2013]

[KER13B] Kernel.org. (2013). *RT project tree*. [en línia]. http://www.kernel.org/pub/linux/kernel/projects/rt/ [data de consulta: 12/01/2013]

[LAY10] Layton, Jeffrey B. (2010). *2.6.33 is Out! Say Good Bye to the Anticipatory Scheduler*. [en línia]. http://www.linux-mag.com/id/7724/ [data de consulta: 08/11/12]

[LEE12] Leemhuis, Thorsten. (2012). *What's new in Linux 3.6*. [en línia]. http://www.h-online.com/open/features/What-s-new-in-Linux-3-6-1714690.html [data de consulta: 13/01/2012]

[LEH89] Lehoczky, John P., Sha, Lui & Ding, Ye. (1989). "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behaviour". *Proceedings of the 10th Real-Time Systems Symposium*. Santa Monica, California :IEEE Computer Society Press

[LIU73] Liu & Layland. (1973) . "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *Journal of the ACM*. (núm. 20/1, pàg. 46-61).

[LOV07] Love, Robert. (2007). *Linux System Programming: Talking Directly to the Kernel and C Library*. California:O'Reilly.

[LOV10] Love, Robert. (2010). *Linux Kernel Development*. India:Pearson.

[MAR04] M. Marquez, Francisco. (2004). *UNIX. Programación avanzada*. Madrid:Ra-Ma.

[GUI05] Mc Guire, Nicholas. (2005). *Benchmarking - Cache issues*. [en línia]. http://dslab.lzu.edu.cn:8080/docs/publications/NicholasMcGuire.pdf [data de consulta: 13/01/2013]

[MSF06] Microsoft. (2006). *Hardware Interrupt Priorities--Background and Usage*. [en línia]. http://support.microsoft.com/kb/99357/en-us [data de consulta: 05/01/13]

[MOL00] Molnar, Ingo. (2000). *[patch] latest 'guaranteed low latency' patch against 2.2.14*. [en línia]. http://lkml.org/lkml/2000/2/7/26 [data de consulta: 14/11/2012]

[MOL05] Molnar, Ingo. (2005). *Re: PREEMPT_RT and I-PIPE: the numbers, take 3*. [en línia]. https://lkml.org/lkml/2005/6/30/9 [data de consulta: 13/01/2013]

[MOL07] Molnar, Ingo. (2007). *Modular Scheduler Core and Completely Fair Scheduler [CFS]*. [en línia]. http://lwn.net/Articles/230501/ [data de consulta: 22/10/2012]

[MOR08] Morreale, Peter W. & van Riel, Rik. (2008). *Documentation for /proc/sys/vm/**. [en línia]. http://www.kernel.org/doc/Documentation/sysctl/vm.txt [data de consulta: 16/01/2003]

[OSA13] OSADL. (2013). *OSADL Realtime QA Farm*. [en línia]. https://www.osadl.org/QA-Farm-Realtime.qa-farm-about.0.html [data de consulta: 12/01/2013]

[PAL10] Pallipadi, Venkatesh. (2010). *[PATCH 0/6] Proper kernel irq time accounting* . [en línia]. http://lwn.net/Articles/405889/ [data de consulta: 07/11/2012]

[RHT12] Red Hat. (2012). *Are hardware power management features causing latency spikes in my application?* . [en línia]. https://access.redhat.com/knowledge/articles/65410 [data de consulta: 05/01/13]

[SAL10] Sally, Gene. (2010). "Chapter 12: Real Time". *Pro Linux Embedded Systems*. USA :Apress

[SHA86] Sha, Lui & Lehoczky, John P. (1986) . "Performance of real-time bus scheduling algorithms". *ACM Performance Evaluation Review*. (núm. 14/1, pàg. 43-45).

[SOS99] Sound On Sound. (1999). *Dealing With Computer Audio Latency*. [en línia]. http://www.soundonsound.com/sos/apr99/articles/letency.htm [data de consulta: 12/01/2013]

[TOR11] Torvalds, Linus. (2011). *Linux 3.0-rc1*. [en línia]. https://lkml.org/lkml/2011/5/29/204 [data de consulta: 12/01/2013]

[WIL12] Williams, Clark. (2013). *rt test utils*. [en línia]. http://git.kernel.org/?p=linux/kernel/git/clrkwllms/rt-tests.git [data de consulta: 13/01/2013]

[WIL02] Williams, Robert. (2002). *Linux Scheduler Latency*. North Carolina:Red Hat, Inc..

8 Annex

8.1 Configuració detallada del maquinari

En l'avaluació de les capacitats en temps real de Linux s'ha utilitzat una Netbook DELL Inspiron 910 amb les següents característiques[108]:

8.1.1 CPU

dmidecode -t4

```
# dmidecode 2.9
SMBIOS 2.5 present.

Handle 0x0005, DMI type 4, 40 bytes
Processor Information
	Socket Designation: U1
	Type: Central Processor
	Family: Other
	Manufacturer: Intel
	ID: C2 06 01 00 FF FB E9 BF
	Version: C0
	Voltage: 1.2 V
	External Clock: 533 MHz
	Max Speed: 2048 MHz
	Current Speed: 1600 MHz
	Status: Populated, Enabled
	Upgrade: Other
	L1 Cache Handle: 0x0006
	L2 Cache Handle: 0x0007
	L3 Cache Handle: Not Provided
	Serial Number: Not Specified
	Asset Tag: Not Specified
	Part Number: Not Specified
	Core Count: 1
	Core Enabled: 1
	Thread Count: 2
	Characteristics: None
```

cat /proc/cpuinfo

```
processor	: 0
vendor_id	: GenuineIntel
cpu family	: 6
model		: 28
model name	: Intel(R) Atom(TM) CPU N270   @ 1.60GHz
stepping	: 2
microcode	: 0x20a
cpu MHz		: 800.000
cache size	: 512 KB
```

108 Extretes a través de la Destktop Management Interface (DMI), una interfície estàndard d'accés a informació detallada sobre els components de maquinari que es troben en un ordinador de sobretaula.

```
fdiv_bug            : no
hlt_bug             : no
f00f_bug            : no
coma_bug            : no
fpu                 : yes
fpu_exception       : yes
cpuid level         : 10
wp                  : yes
flags               : fpu vme de tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe nx constant_tsc arch_perfmon pebs bts
aperfmperf pni dtes64 monitor ds_cpl est tm2 ssse3 xtpr pdcm movbe lahf_lm dtherm
bogomips            : 3193.31
clflush size        : 64
cache_alignment     : 64
address sizes       : 32 bits physical, 32 bits virtual
power management:
```

8.1.2 Memòria cau

dmidecode -t7

```
# dmidecode 2.9
SMBIOS 2.5 present.

Handle 0x0006, DMI type 7, 19 bytes
Cache Information
       Socket Designation: L1 Cache
       Configuration: Enabled, Socketed, Level 1
       Operational Mode: Write Back
       Location: Internal
       Installed Size: 32 KB
       Maximum Size: 32 KB
       Supported SRAM Types:
              Burst
              Pipeline Burst
              Asynchronous
       Installed SRAM Type: Asynchronous
       Speed: Unknown
       Error Correction Type: Unknown
       System Type: Unknown
       Associativity: Unknown

Handle 0x0007, DMI type 7, 19 bytes
Cache Information
       Socket Designation: L2 Cache
       Configuration: Enabled, Socketed, Level 2
       Operational Mode: Write Back
       Location: External
       Installed Size: 512 KB
       Maximum Size: 4096 KB
       Supported SRAM Types:
              Burst
              Pipeline Burst
              Asynchronous
       Installed SRAM Type: Burst
       Speed: Unknown
       Error Correction Type: Unknown
```

```
      System Type: Unknown
      Associativity: Unknown
```

8.1.3 Memòria RAM

dmidecode -t16 -t17

```
# dmidecode 2.9
SMBIOS 2.5 present.

Handle 0x0010, DMI type 16, 15 bytes
Physical Memory Array
      Location: System Board Or Motherboard
      Use: System Memory
      Error Correction Type: None
      Maximum Capacity: 1 GB
      Error Information Handle: Not Provided
      Number Of Devices: 1

Handle 0x0011, DMI type 17, 27 bytes
Memory Device
      Array Handle: 0x0010
      Error Information Handle: No Error
      Total Width: 32 bits
      Data Width: 32 bits
      Size: 1024 MB
      Form Factor: SODIMM
      Set: 1
      Locator: M1
      Bank Locator: Bank 0
      Type: DDR2
      Type Detail: Synchronous
      Speed: 533 MHz (1.9 ns)
      Manufacturer:
      Serial Number:
      Asset Tag:
      Part Number:
```

8.1.4 Disc dur

hparm -i /dev/hda

```
/dev/hda:

 Model=STEC PATA 32GB, FwRev=E5221-10, SerialNo=STM00008E829
 Config={ HardSect NotMFM Fixed DTR>10Mbs }
 RawCHS=16383/16/63, TrkSize=0, SectSize=512, ECCbytes=4
 BuffType=unknown, BuffSize=unknown, MaxMultSect=1, MultSect=off
 CurCHS=59711/16/63, CurSects=60188688, LBA=yes, LBAsects=60188688
 IORDY=yes, tPIO={min:120,w/IORDY:120}, tDMA={min:120,rec:120}
 PIO modes:  pio0 pio1 pio2 pio3 pio4
 DMA modes:  mdma0 mdma1 mdma2
 UDMA modes: udma0 udma1 udma2 udma3 udma4 *udma5 udma6
 AdvancedPM=no WriteCache=enabled
 Drive conforms to: Unspecified:  ATA/ATAPI-3,4,5,6

 * signifies the current active mode
```

fdisk (opció 'p')

```
Disk hda: 30.8 GB, 30816608256 bytes
255 heads, 63 sectors/track, 3746 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x000b71d9

Device Boot      Start         End      Blocks   Id  System
  hda1   *           1        3589    28822528   83  Linux
  hda2            3589        3747     1268737    5  Extended
  hda5            3589        3747     1268736   82  Linux swap / Solaris
```

8.1.5 Bus PCI

lspci -v

```
00:00.0 Host bridge: Intel Corporation Mobile 945GME Express Memory Controller Hub
(rev 03)
      Subsystem: Dell Device 02b0
      Flags: bus master, fast devsel, latency 0
      Capabilities: <access denied>
      Kernel driver in use: agpgart-intel

00:02.0 VGA compatible controller: Intel Corporation Mobile 945GME Express Integrated
Graphics Controller (rev 03) (prog-if 00 [VGA controller])
      Subsystem: Dell Device 02b0
      Flags: bus master, fast devsel, latency 0, IRQ 16
      Memory at f0000000 (32-bit, non-prefetchable) [size=512K]
      I/O ports at 1800 [size=8]
      Memory at d0000000 (32-bit, prefetchable) [size=256M]
      Memory at f0300000 (32-bit, non-prefetchable) [size=256K]
      Expansion ROM at <unassigned> [disabled]
      Capabilities: <access denied>
      Kernel driver in use: i915

00:02.1 Display controller: Intel Corporation Mobile 945GM/GMS/GME, 943/940GML
Express Integrated Graphics Controller (rev 03)
      Subsystem: Dell Device 02b0
      Flags: bus master, fast devsel, latency 0
      Memory at f0080000 (32-bit, non-prefetchable) [size=512K]
      Capabilities: <access denied>

00:1b.0 Audio device: Intel Corporation N10/ICH 7 Family High Definition Audio
Controller (rev 02)
      Subsystem: Dell Device 02b0
      Flags: bus master, fast devsel, latency 0, IRQ 22
      Memory at f0540000 (64-bit, non-prefetchable) [size=16K]
      Capabilities: <access denied>
      Kernel driver in use: snd_hda_intel

00:1c.0 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 1 (rev 02)
(prog-if 00 [Normal decode])
      Flags: bus master, fast devsel, latency 0
      Bus: primary=00, secondary=02, subordinate=02, sec-latency=0
      I/O behind bridge: 00004000-00004fff
      Memory behind bridge: f0100000-f01fffff
      Prefetchable memory behind bridge: 0000000040800000-0000000040afffff
      Capabilities: <access denied>
```

```
	Kernel driver in use: pcieport

00:1c.1 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 2 (rev 02)
(prog-if 00 [Normal decode])
	Flags: bus master, fast devsel, latency 0
	Bus: primary=00, secondary=03, subordinate=03, sec-latency=0
	I/O behind bridge: 00003000-00003fff
	Memory behind bridge: f0200000-f02fffff
	Prefetchable memory behind bridge: 0000000040600000-00000000407fffff
	Capabilities: <access denied>
	Kernel driver in use: pcieport

00:1c.2 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 3 (rev 02)
(prog-if 00 [Normal decode])
	Flags: bus master, fast devsel, latency 0
	Bus: primary=00, secondary=04, subordinate=04, sec-latency=0
	I/O behind bridge: 00002000-00002fff
	Memory behind bridge: 40100000-405fffff
	Prefetchable memory behind bridge: 00000000f0600000-00000000f06fffff
	Capabilities: <access denied>
	Kernel driver in use: pcieport

00:1d.0 USB Controller: Intel Corporation N10/ICH 7 Family USB UHCI Controller #1
(rev 02) (prog-if 00 [UHCI])
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0, IRQ 23
	I/O ports at 1820 [size=32]
	Kernel driver in use: uhci_hcd

00:1d.1 USB Controller: Intel Corporation N10/ICH 7 Family USB UHCI Controller #2
(rev 02) (prog-if 00 [UHCI])
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0, IRQ 19
	I/O ports at 1840 [size=32]
	Kernel driver in use: uhci_hcd

00:1d.2 USB Controller: Intel Corporation N10/ICH 7 Family USB UHCI Controller #3
(rev 02) (prog-if 00 [UHCI])
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0, IRQ 18
	I/O ports at 1860 [size=32]
	Kernel driver in use: uhci_hcd

00:1d.3 USB Controller: Intel Corporation N10/ICH 7 Family USB UHCI Controller #4
(rev 02) (prog-if 00 [UHCI])
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0, IRQ 16
	I/O ports at 1880 [size=32]
	Kernel driver in use: uhci_hcd

00:1d.7 USB Controller: Intel Corporation N10/ICH 7 Family USB2 EHCI Controller (rev
02) (prog-if 20 [EHCI])
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0, IRQ 23
	Memory at f0544000 (32-bit, non-prefetchable) [size=1K]
	Capabilities: <access denied>
	Kernel driver in use: ehci_hcd
```

```
00:1e.0 PCI bridge: Intel Corporation 82801 Mobile PCI Bridge (rev e2) (prog-if 01
[Subtractive decode])
	Flags: bus master, fast devsel, latency 0
	Bus: primary=00, secondary=05, subordinate=05, sec-latency=32
	Capabilities: <access denied>

00:1f.0 ISA bridge: Intel Corporation 82801GBM (ICH7-M) LPC Interface Bridge (rev 02)
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0
	Capabilities: <access denied>

00:1f.1 IDE interface: Intel Corporation 82801G (ICH7 Family) IDE Controller (rev 02)
(prog-if 8a [Master SecP PriP])
	Subsystem: Dell Device 02b0
	Flags: bus master, medium devsel, latency 0, IRQ 19
	I/O ports at 01f0 [size=8]
	I/O ports at 03f4 [size=1]
	I/O ports at 0170 [size=8]
	I/O ports at 0374 [size=1]
	I/O ports at 1810 [size=16]
	Kernel driver in use: PIIX_IDE

00:1f.3 SMBus: Intel Corporation N10/ICH 7 Family SMBus Controller (rev 02)
	Subsystem: Dell Device 02b0
	Flags: medium devsel, IRQ 19
	I/O ports at 18a0 [size=32]

02:00.0 System peripheral: JMicron Technology Corp. SD/MMC Host Controller
	Subsystem: Dell Device 02b0
	Flags: bus master, fast devsel, latency 0, IRQ 16
	Memory at f0100000 (32-bit, non-prefetchable) [size=256]
	[virtual] Expansion ROM at 40800000 [disabled] [size=64K]
	Capabilities: <access denied>
	Kernel driver in use: sdhci-pci

02:00.2 SD Host controller: JMicron Technology Corp. Standard SD Host Controller
(prog-if 01)
	Subsystem: Dell Device 02b0
	Flags: fast devsel, IRQ 16
	Memory at f0100400 (32-bit, non-prefetchable) [size=256]
	Capabilities: <access denied>

02:00.3 System peripheral: JMicron Technology Corp. MS Host Controller
	Subsystem: Dell Device 02b0
	Flags: bus master, fast devsel, latency 0, IRQ 16
	Memory at f0100800 (32-bit, non-prefetchable) [size=256]
	Capabilities: <access denied>
	Kernel driver in use: jmb38x_ms

03:00.0 Network controller: Broadcom Corporation BCM4312 802.11b/g LP-PHY (rev 01)
	Subsystem: Broadcom Corporation Device 04b5
	Flags: bus master, fast devsel, latency 0, IRQ 17
	Memory at f0200000 (64-bit, non-prefetchable) [size=16K]
	Capabilities: <access denied>
	Kernel driver in use: b43-pci-bridge

04:00.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL8101E/RTL8102E PCI
```

```
Express Fast Ethernet controller (rev 02)
      Subsystem: Dell Device 02b0
      Flags: bus master, fast devsel, latency 0, IRQ 18
      I/O ports at 2000 [size=256]
      Memory at f0610000 (64-bit, prefetchable) [size=4K]
      Memory at f0600000 (64-bit, prefetchable) [size=64K]
      [virtual] Expansion ROM at f0620000 [disabled] [size=128K]
      Capabilities: <access denied>
      Kernel driver in use: r8169
```

8.2 Descarrega, aplicació del pedaç i configuració d'un nucli RT

A dia d'avui existeixen diversos procediments, i moltes facilitats, per a la obtenció i instal·lació d'un sistema operatiu Linux. Per aquest projecte, més concretament, s'han seguit eines i passos que a continuació s'exposen a mode de guia general.

8.2.1 Instal·lació d'una distribució de Linux

El primer pas, si no es disposa de Linux, és la descarrega i instal·lació d'una distribució: un conjunt format per nucli, llibreries, controladors, i aplicacions que configuren un sistema operatiu complet i llest per a ser utilitzat. Per aquest projecte s'ha utilitzat una distribució Debian que té com a característiques: que és de propòsit general, que està desenvolupada íntegrament per la comunitat lliure, que és una de les més populars [DIS12] i que és la base d'una gran quantitat d'altres distribucions.

Existeixen principalment dos branques de la distribució: la estable, batejada com a *Squeeze* en la versió 6.0, i la *testing*, batejada com a *Wheezy* en la versió 6.0. La primera es distingeix en què es més madura mentre la segona, que utilitza les últimes versions de nucli, incorpora les noves funcionalitats. Debian es pot obtenir de diverses maneres, la més comú és la descàrrega.

```
# Plana web per a la descàrrega de Debian
http://www.debian.org/distrib/
```

Si cal instal·lar el nou sistema operatiu des d'un disc flash USB es pot crear el disc d'instal·lació gràcies a eines com Universal USB Installer.

```
# Plana web per a la creació d'un USB bootable
http://www.pendrivelinux.com/
# Eina de creació d'un USB bootable
http://www.pendrivelinux.com/universal-usb-installer-easy-as-1-2-3/
```

8.2.2 Descarrega d'un nou nucli

Són diverses les motivacions per instal·lar-se i configurar un nou nucli, entre elles: la inclusió de noves funcionalitats, la optimització del codi, la inclusió estàtica de mòduls dinàmics, o, com és el cas del projecte, la investigació i aprenentatge. El nucli es pot descarregar de diverses maneres, una de elles mitjançant la seva plana web.

```
# Plana web per a la descàrrega del nucli
http://www.kernel.org/
# Descarrega del nou nucli
cd /usr/src
sudo wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.2.32.tar.bz2
```

Un cop descarregat és necessari descomprimir-lo i compilar-lo, ja que aquest es distribueix en codi font. Per aquest propòsit es pot utilitzar l'empaquetador de fitxers *tar*. Es recomanable utilitzar */usr/src* com indica l'especificació del Filesystem Hierarchy System (FHS[109]).

```
# Descompressió i desempaquetatge del codi font del nucli
sudo tar xjf /usr/src/linux-3.2.32.tar.bz2
cd /usr/src/linux-3.2.32
```

A continuació, abans de compilar-lo, cal repassar les opcions de configuració seleccionant les més adients pel cas i l'equip plantejat. Per modificar la configuració podem fer servir qualsevol dels menús que incorpora .

```
# Configuració pregunta a pregunta, no recomanable
sudo make oldconfig
# Menú textual, amb cerca de opcions i ajuda de les opcions
sudo make menuconfig
# Menú gràfic amb ajuda. Necessita llibreries gràfiques Qt
sudo make xconfig
# Menú gràfic amb ajuda. Necessita llibreries gràfiques Gtk+2.0
sudo make gconfig
```

8.2.3 Obtenció i aplicació del pedaç RT

Si el que es vol generar és un nucli amb un pedaç especial, com és el cas del RT, abans de configurar i compilar, cal descarregar-se el pedaç del repositori i, a continuació, aplicar-lo amb la comanda *patch*.

```
# Descarrega de pedaç per al nucli.
sudo wget http://kernel.org/pub/linux/kernel/projects/rt/3.2/<nom_actual>.bz2
# Aplicació del pedaç des del directori on resideix
sudo patch -p1 < {/path/to/patch/file}
```

8.2.4 Compilació del nucli

Un cop descarregat i correctament copiat a una carpeta de destí adient es realitza la compilació executant el *script* del make del que es poden veure les opcions invocant-lo amb l'argument *help*. En el cas d'aquest projecte s'utilitza l'argument que indica que en la compilació s'inclogui la generació de paquets de tipus Debian. El compilador dona la possibilitat d'assignar la tasca de compilació a més de un nucli mitjançant la comanda -j *n*, on *n* és el núcli, bé sigui multiprocessador o via *threading*, oferint un menor temps de compilació.[110]

```
# Veure opcions de compilació del nucli
sudo make help
# Compilar el nucli amb un target de paquet de Debian
sudo make deb-pkg – j 2
```

Finalment, s'instal·len els nous paquets, situats al directori superior de compilació, utilitzant la comanda *dpkg*. Aquesta operació realitza, també, les actualitzacions necessàries als fitxer d'inici del equip de tal manera que en arrancar ens trobem amb la possibilitat d'utilitzar el nou nucli.

109

110 Les compilacions realitzades per aquest projecte han estat realitzades en la plataforma utilizada en l'assaig i han tarda una mitjana de 4 hores. Una segona opció hagués estat compilar-les en una altre estació de treball amb més potència, preferiblement amb el mateix repertori d'instruccions -de la familia x86- o, en cas contrari, mitjançant compilació creuada.

```
# Instal·lar el nou paquet
dpkg – i <nom_del_fitxer_image>.deb
# Comprovació de la nova imatge preparada per carregar
ls /boot
```

Un cop arrancat el nou nucli podem verificar que efectivament es tract del compilat amb la comanda *uname*.

```
# Imprimir versió de nucli i detalls
uname -a
```

8.2.5 Instal·lació del nucli

Finalment, i gràcies a haver utilitzat la opció *deb-pkg* en la compilació del codi , es procedeix a la instal·lació del nou nucli utilitzant el gestor de paquets de Debian (*dpkg*).

```
#Instal·lació del nou nucli
dpkg -i
```

8.3 Adaptació d'aplicacions

En aquest apartat es presenten les modificacions realitzades en les aplicacions utilitzades per a la caracterització de les càrregues i execució de l'experiment d'avaluació del temps real i rendiment de les quatre configuracions de nucli avaluades.

8.3.1 Modificació de l'aplicació *cyclictest*

La aplicació *cyclictest* permet la mesura de la latència mitjançant la utilització de temporitzadors. Tanmateix no permet l'execució o simulació d'una carrega de treball pel que no serveix per simular l'execució d'una tasca determinada de manera periòdica.

La següent modificació afegeix a aquesta aplicació la possibilitat d'executar una aplicació externa (-x) i, a més, de controlar-ne el seu temps màxim en microsegons (-u). Està realitzada sobre la versió 0.85 de l'aplicació -fitxer únic *cyclictest.c*- que pot descarregar-se del paquet d'utilitats rt-test que es troba a: http://git.kernel.org/?p=linux/kernel/git/clrkwllms/rt-tests.git;a=summary.

L'execució de les noves opcions es realitza mitjançant els arguments *-x cmd* i *-z cmd_timeout*, on *cmd* és l'aplicació a executar de manera periòdica (*fork* i *wait*) i *cmd_timeout* el temps transcorregut en microsegons després del qual es finalitzarà l'execució de l'aplicació de manera forçada (*sigtimedwait* i *kill*).

Nom del fitxer: cyclictest.c

```
diff cyclictest.c.orig cyclictest.c h
<<< cyclictest.c.orig
>>> cyclictest.c h
27a28
> #include <wait.h>
140a142,143
>      char *cmd;
>      long cmd_timeout;
909a913,1003
>
>            /* once everything is done launch command */
>            if (par->cmd != NULL) {
>                  int i = 0;
```

```
>                   char *args[5];
>                   pid_t pid;
>                   struct timespec tout;
>                   sigset_t sigchld, sigfull;
>
>             #if 1
>                   if ((par->policy == SCHED_FIFO || par->policy == SCHED_RR) &&
par->prio > 1) {
>                         int prion;
>                         char prio[3];
>                         args[i++] = "/usr/bin/chrt";
>                         prion = par->prio;
>                         if (par->policy == SCHED_FIFO && par->prio > 1) {
>                               /* if FIFO give less priority so we can kill him */
>                               prion--;
>                               args[i++] = "-f";
>                         }
>                         else {
>                               args[i++] = "-r";
>                         }
>                         snprintf(prio, sizeof(prio), "%u", prion);
>                         args[i++] = prio;
>                   }
>             #endif
>                   args[i++] = (char *)par->cmd;
>                   args[i] = (char *)NULL;
>
>                   /* add SIGCHLD to blocked signals */
>                   if (par->cmd_timeout) {
>                         sigemptyset(&sigchld);
>                         sigaddset(&sigchld, SIGCHLD);
>                         if (sigprocmask(SIG_BLOCK, &sigchld, NULL) < 0) {
>                               perror("pthread_sigmask");
>                               exit(-1);
>                         }
>                   }
>
>                   switch (pid = fork()) {
>                   case 0:
>                         sigfillset(&sigfull);
>                         if (sigprocmask(SIG_UNBLOCK, &sigfull, NULL) < 0) {
>                               perror("pthread_sigmask");
>                               exit(-1);
>                         }
>                         execvp(args[0], args);
>                         /* no break */
>                   case -1:
>                         perror("fork");
>                         for (i=0; i<sizeof(args)/sizeof(args[0]); i++) {
>                               char *p = args[i];
>                               if (p == NULL)
>                                     break;
>                               fprintf(stderr, "%u=[%s] ", i, p);
>                         }
>                         exit(-1);
>                         break;
>                   default:
```

```
>                         if (!par->cmd_timeout) {
>                               if (waitpid(pid, NULL, 0) < 0) {
>                                     perror("waitpid");
>                                     exit(-1);
>                               }
>                         }
>                         else {
>                               tout.tv_sec = par->cmd_timeout / USEC_PER_SEC;
>                               tout.tv_nsec = (par->cmd_timeout % USEC_PER_SEC) *
1000;
>                               do {
>                                     /* wait untiol terminates or timeout */
>                                     if (sigtimedwait(&sigchld, NULL, &tout) < 0) {
>                                           /* other signal than sigchld or timeout
*/
>                                           if (errno == EINTR || errno == EAGAIN)
{
>                                                 kill (pid, SIGKILL);
>                                           }
>                                           else {
>                                                 perror("sigtimedwait");
>                                                 exit(-1);
>                                           }
>                                     }
>                                     break;
>                               } while (1);
>                               if (waitpid(pid, NULL, 0) < 0) {
>                                     perror("waitpid");
>                                     exit(-1);
>                               }
>                         }
>                         break;
>                   }
>             }
1006a1101,1102
>              "-x       --exec CMD        exec CMD (full path, use script for args,
chrt at /bin/usr for rt)\n"
>              "-z TOUT  -cmdtout TOUT     timeout for command in us \n",
1026a1123,1124
> static char* cmd;
> static long cmd_timeout = 0;
1251a1350,1355
>             case 'x':
>                   cmd = optarg;
>                   break;
>             case 'z':
>                   cmd_timeout = atol(optarg);
>                   break;
1751a1856,1857
>             par->cmd = cmd;
>             par->cmd_timeout = cmd_timeout;
```

8.3.2 Modificació de l'aplicació *time*

La aplicació *time* de GNU permet executar un programa de manera que al finalitzar se'n reporten diverses mètriques registrades durant la seva execució. Tanmateix, no incorpora la possibilitat d'informar sobre les mètriques registrades amb la opció de compilació SCHEDSTATS, més

concretament el temps d'execució de la tasca i el temps d'espera de la tasca en la cua del planificador.

La següent modificació afegeix a aquesta aplicació la possibilitat d'incloure els estadístics esmentats, temps en execució (%a) i temps en cua d'espera (%b), tot i que només reporta els temps corresponents al procés executat sense incorporar els temps dels fills pel que no és útil per aplicacions que basen el seu funcionament en la creació i execució d'altres processos com pot ser el cas de un *script*.

La modificació ha estat realitzada sobre la versió 1.7-23.1 del paquete *time* de GNU que es pot trobar com a paquet Debian a: http://packages.debian.org/source/squeeze/time. Aquesta modificació inclou les següents accions: creació del fitxer schedstats.c amb les rutines de captura de les noves mètriques i modificació dels fitxers resuse.c, resuse.h i time.c per a incloure les estructures de dades i crides necessaries per l'utilització de les mateixes.

Nom del fitxer: schedstats.c

```
/* schedstats.c */

#include <stdio.h>
#include <signal.h>

/*
 * get_stats() -- we presume that we are interested in the first three
 *     fields of the line we are handed, and further, that they contain
 *     only numbers and single spaces.
 */
void get_stats(char *buf, long long *run_ticks, long long *wait_ticks,
    long long *nran)
{
    char *ptr;

    /* sanity */
    if (!buf || !run_ticks || !wait_ticks || !nran)
      return;

    /* leading blanks */
    ptr = buf;
    while (*ptr && isblank(*ptr))
      ptr++;

    /* first number -- run_ticks */
    *run_ticks = atoll(ptr);
    while (*ptr && isdigit(*ptr))
      ptr++;
    while (*ptr && isblank(*ptr))
      ptr++;

    /* second number -- wait_ticks */
```

```
    *wait_ticks = atoll(ptr);
    while (*ptr && isdigit(*ptr))
      ptr++;
    while (*ptr && isblank(*ptr))
      ptr++;

    /* last number -- nran */
    *nran = atoll(ptr);
}

char get_state(char *buf)
{
  char *ptr;

  /* sanity */
  if (!buf)
    return;

  ptr = index(buf, ')') + 2;
  return *ptr;
}

void get_schedstats(pid_t pid, long long *wait, long long *run)
{
    FILE *fp;
    char filename[64], state;
    char procbuf[512];
    long long nran;

#if 1
    sigset_t            mask;
    siginfo_t           info;

    sigemptyset(&mask);
    sigaddset(&mask, SIGCHLD);
    /* block until queued signals in mask (SIGCHLD only) */
    if (sigwaitinfo(&mask, &info) == -1) {
        perror("sigwaitinfo() failed");
    }
#else
   /* wait for process to become zombie */
    snprintf(filename, sizeof(filename), "/proc/%d/stat", pid);
    state = '\0';
    do {
      usleep(10000);
      if (fp = fopen(filename, "r")) {
```

```
          if (fgets(procbuf, sizeof(procbuf), fp))
            state = get_state(procbuf);
          fclose(fp);
        }
    } while (state != 'Z');
#endif

      getchar();

    *wait = 0;
    *run = 0;
    /* get stats after exec */
    snprintf(filename, sizeof(filename), "/proc/%d/schedstat", pid);
    if (fp = fopen(filename, "r")) {
        if (fgets(procbuf, sizeof(procbuf), fp))
            get_stats(procbuf, run, wait, &nran);
        fclose(fp);
    }

    /* nanoseconds to microseconds */
    *run /= 1000;
    *wait /= 1000;
}
```

Nom del fitxer: resuse.h

```
diff resuse.h.orig resuse.h
<<< resuse.h.orig
>>> resuse.h
18a19,20
> #define SCHEDSTATS
>
51a54,57
> #ifdef SCHEDSTATS
>   long long wait;
>   long long run;
> #endif
```

Nom del fitxer: resuse.c

```
diff resuse.c.orig resuse.c
<<< resuse.c.orig
>>> resuse.c
22a23,24
> #define SCHEDSTATS
>
25a28,31
```

```
> #ifdef SCHEDSTATS
> #include "schedstats.h"
> #endif
>
80a87,90
> #ifdef SCHEDSTATS
>     get_schedstats(pid, &resp->wait, &resp->run);
> #endif
>
```

Nom del fitxer: time.c

```
diff time.c.orig time.c
<<< time.c.orig
>>> time.c
20a21,23
>
> #define SCHEDSTATS
>
104a108
> #ifndef SCHEDSTATS
107a112,116
> #else
> static const char *const default_format =
> "%Uuser %Ssystem [%arun %bwait] %Eelapsed %PCPU (%Xavgtext+%Davgdata
%Mmaxresident)k\n\
> %Iinputs+%Ooutputs (%Fmajor+%Rminor)pagefaults %Wswaps";
> #endif
126a136,139
> #ifdef SCHEDSTATS
>   "\tSched. running: %a\n",
>   "\tSched. waiting: %b\n",
> #endif
331a345
> #ifndef SCHEDSTATS
333a348,351
> #else
>   unsigned long long r;       /* Elapsed real microseconds.  */
>   unsigned long long v;       /* Elapsed virtual (CPU) microseconds.  */
> #endif
350a369
> #ifndef SCHEDSTATS
354a374,380
> #else
>   /* Convert all times to microseconds */
>   r = resp->elapsed.tv_sec * 1000000 + resp->elapsed.tv_usec;
>
```

```
>   v = resp->ru.ru_utime.tv_sec * 1000000 + resp->ru.ru_utime.TV_USEC +
>     resp->ru.ru_stime.tv_sec * 1000000 + resp->ru.ru_stime.TV_USEC;
> #endif
375a402
> #ifndef SCHEDSTATS
385a413,418
> #else
>            fprintf (fp, "%ld:%02ld.%03ld", /* -> m:s.ms */
>                   resp->elapsed.tv_sec / 60,
>                   resp->elapsed.tv_sec % 60,
>                   resp->elapsed.tv_usec / 1000);
> #endif
416a450
> #ifndef SCHEDSTATS
419a454,458
> #else
>           fprintf (fp, "%ld.%03ld",
>                   resp->ru.ru_stime.tv_sec,
>                   resp->ru.ru_stime.TV_USEC / 1000);
> #endif
421a461
> #ifndef SCHEDSTATS
424a465,469
> #else
>           fprintf (fp, "%ld.%03ld",
>                   resp->ru.ru_utime.tv_sec,
>                   resp->ru.ru_utime.TV_USEC / 1000);
> #endif
440a486
> #ifndef SCHEDSTATS
443a490,494
> #else
>           fprintf (fp, "%ld.%03ld",
>                   resp->elapsed.tv_sec,
>                   resp->elapsed.tv_usec / 1000);
> #endif
472a524,535
> #ifdef SCHEDSTATS
>         case 'a':             /* schedstats runtime */
>           fprintf (fp, "%lld.%03ld",
>                   resp->run / 1000000,
>                   (resp->run % 1000000) / 1000);
>           break;
>         case 'b':             /* schedstats waittime */
>           fprintf (fp, "%lld.%03ld",
>                   resp->wait / 1000000,
```

```
>                   (resp->wait % 1000000) / 1000);
>            break;
> #endif
603a667,672
> void
> signal_handler(int signo)
> {
>     // do nothing
> }
>
615a685,701
>
> #if 0 //#ifdef SCHEDSTATS
>     sigset_t            mask;
>     struct sigaction    action;
>
>     // mask out the SIGCHLD signal so that it will not interrupt us,
>     // (side note: the child inherits the parents mask)
>     /* mask out SIGCHLD ...*/
>     sigemptyset(&mask);
>     sigaddset(&mask, SIGCHLD);
>     sigprocmask(SIG_BLOCK, &mask, NULL);
>     /* ..and install dummy handler so it's not ignored */
>     action.sa_handler = signal_handler;
>     sigemptyset(&action.sa_mask);
>     action.sa_flags = SA_SIGINFO; // make it a queued signal
>     sigaction(SIGCHLD, &action, NULL);
> #endif
```

8.3.3 Modificació de l'aplicació *stress*

La aplicació *stress* permet la simulació d'una càrrega especialitzada en l'ús intensiu de diversos subsistemes com són la CPU, la memòria principal (RAM) o la memòria auxiliar (disc), entre altres. Permet el control de temps de la seva execució tot i que la realització de l'experiment s'ha considerat oportú realitzar un parell de modificacions: suport de temporitzadors d'alta resolució (-u) i limitació del temps d'execució per nombre de iteracions (-l).

Està realitzada sobre la versió 1.0.4 de l'aplicació -fitxer únic *stress.c*- que pot descarregar-se a: http://weather.ou.edu/~apw/projects/stress/.

Nom del fitxer: stress.c

```
diff stress.c.orig stress.c
<<< stress.c.orig
>>> stress.c
31a32
> #include<sys/time.h>
65a67,69
> /* Make code readable */
> #define USEC_PER_SEC          1000000
>
```

```
77a82,84
> long long do_usec_timeout = 0;
> long long do_loops = 0;
>
82a90
>   sigset_t nset;
103c111
<     }
---
>     }
241a250,274
>       else if (strcmp (arg, "--usec") == 0 || strcmp (arg, "-u") == 0)
>         {
>           if (do_timeout)
>             {
>               err (stderr, "--usec not compatible with --timeout");
>               exit (1);
>             }
>           assert_arg ("--used");
>           do_usec_timeout = atoll_s (arg);
>           if (do_usec_timeout <= 0)
>             {
>               err (stderr, "invalid usec timeout value: %llius\n",
do_usec_timeout);
>               exit (1);
>             }
>         }
>       else if (strcmp (arg, "--loops") == 0 || strcmp (arg, "-l") == 0)
>         {
>           assert_arg ("--loops");
>           do_loops = atoll_s (arg);
>           if (do_loops <= 0)
>             {
>               err (stderr, "invalid number of loops value: %lli\n", do_loops);
>               exit (1);
>             }
>         }
257a291,294
>   sigemptyset (&nset);
>   sigaddset (&nset, SIGALRM);
>   sigprocmask(SIG_UNBLOCK, &nset, NULL);
>
261a299
>       struct itimerval usec_timeout;
266c304
<
---
>
292a331,337
>       /* If we are supposed to respect a usec timeout, calculate it.  */
>       if (do_usec_timeout)
>         {
>           memset((void *)&usec_timeout, 0, sizeof(usec_timeout));
>           usec_timeout.it_value.tv_sec = do_usec_timeout / USEC_PER_SEC;
>           usec_timeout.it_value.tv_usec = do_usec_timeout % USEC_PER_SEC;
>         }
298c343,346
```

```
<               alarm (timeout);
---
>               if (!do_usec_timeout)
>                 alarm (timeout);
>               else
>                 setitimer(ITIMER_REAL, &usec_timeout, NULL);
319c367,370
<               alarm (timeout);
---
>               if (!do_usec_timeout)
>                 alarm (timeout);
>               else
>                 setitimer(ITIMER_REAL, &usec_timeout, NULL);
339c390,393
<               alarm (timeout);
---
>               if (!do_usec_timeout)
>                 alarm (timeout);
>               else
>                 setitimer(ITIMER_REAL, &usec_timeout, NULL);
360c414,417
<               alarm (timeout);
---
>               if (!do_usec_timeout)
>                 alarm (timeout);
>               else
>                 setitimer(ITIMER_REAL, &usec_timeout, NULL);
451c508
<       err (stderr, "failed run completed in %lis\n", runtime);
---
>       err (stderr, "failed run completed in %lis.\n", runtime);
455c512
<       out (stdout, "successful run completed in %lis\n", runtime);
---
>       out (stdout, "successful run completed in %lis. %lli loops done.\n", runtime,
do_loops);
464c521
<   while (1)
---
>   while (1) {
465a523,525
>     if (do_loops && !--do_loops)
>       break;
>   }
473c533
<   while (1)
---
>   while (1) {
474a535,537
>     if (do_loops && !--do_loops)
>       break;
>   }
531a595,596
>       if (do_loops && !--do_loops)
>         break;
603a669,670
>       if (do_loops && !--do_loops)
>         break;
```

```
755c822,824
<     "     --hdd-bytes B  write B bytes per hdd worker (default is 1GB)\n\n"
---
>     "     --hdd-bytes B  write B bytes per hdd worker (default is 1GB)\n"
>     " -u  --usec USEC    timeout after USEC microseconds\n"
>     " -l  --loops LOOPS  number of loops for the tests\n\n”
```

8.4 Scripts de caracterització de càrregues

Els següents *scripts* han estat implementats per a realitzar la recol·lecció de les dades monitoritzades en la caracterització de les càrregues i l'execució de l'experiment d'avaluació del temps real i rendiment de les quatre configuracions de nucli ficades a prova.

8.4.1 Monitorització de CPU, memòria i disc

El següent *script* utilitza *vmstat* per a registrar diverses mètriques del sistema en relació a l'ús de la CPU, la memòria i el disc. Crea un directori per guardar els resultats de cada configuració de nucli, segons nom, i redirigeix la sortida de la utilitat de monitorització.

Nom del fitxer: profile

```
#!/bin/bash
if [ $# -ne 1 ]; then
       echo "Us $0 <fitxer>"
       exit
fi
mkdir -p $(uname -r)
vmstat 1 -n > $(uname -r)/$1
```

8.4.2 Monitorització de I/O

El següent *script* utilitza *iostat* per a registrar diverses mètriques del sistema en relació a l'ús de disc. Crea un directori per guardar els resultats de cada configuració de nucli, segons nom, i redirigeix la sortida de la utilitat de monitorització.

Nom del fitxer: profileio

```
#!/bin/bash
if [ $# -ne 1 ]; then
       echo "Us $0 <fitxer>"
       exit
fi
FILE=$(uname -r)/$1
mkdir -p $(uname -r)
iostat -xd | grep -i "device" > $FILE
iostat -xd 1 | stdbuf -oL grep -i hda >> $FILE
```

8.4.3 Monitorització de xarxa

El següent *script* utilitza *netstat* per a registrar diverses mètriques del sistema en relació a l'ús de xarxa. Crea un directori per guardar els resultats de cada configuració de nucli, segons nom, i després de calcular la diferència entre cadascuna de les mostres, per a calcular-ne el ràtio per segon, redirigeix la sortida de la utilitat de monitorització.

Nom del fitxer: profilenet

```
#!/bin/bash
```

```
# Comprovar arguments
if [ $# -ne 1 ]; then
      echo "Us $0 <fitxer>"
      exit
fi
FILE=$(uname -r)/$1
mkdir -p $(uname -r)

# Escriure capcalera
echo "rxp/s rxb/s txp/s txb/s" > $FILE

# Executar netstat
stdbuf -oL netstat -iec \
| sed 's/ \+/ /g' \
| awk -v FS="[: ]" -v OFS=" " '/lo/{lo=1}/^$/ && lo && rxpa{print rxp-rxpa,rxb-
rxba,txp-txpa,txb-txba;fflush()}/^$/ &&
lo{lo=0;rxpa=rxp;rxba=rxb;txpa=txp;txba=txb}lo && /RX bytes/{rxb=$4;txb=$9}lo && /RX
packets/{rxp=$4}lo && /TX packets/{txp=$4}' \
>> $FILE
```

8.4.4 Prioritat dels processos en temps real en execució

Aquest senzill *script*, emprat en la selecció de la prioritat per a les tasques de temps real, utilitza la comanda *ps* per a mostrar una llista dels processos amb polítiques de planificació de temps real (SCHED_FIFO o SCHED_RR) així la prioritat fixa assignada.

Nom del fitxer: prio

```
ps -Leo ppid,pid,cmd,stat,class,rtprio | grep -v "\-$"
```

8.5 Scripts de visualització de gràfiques de caracterització

8.5.1 Visualització de gràfiques des de shell

El següent *script* executa *gnuplot* passant-li com a paràmetres els arguments *FILE* i *PNG* utilitzats per indicar el fitxer obtingut amb els *scripts* de l'apartat anterior i si cal (1) o no cal (0) generar un gràfic de format *png*, respectivament. Permet executar fins quatre *scripts* de *gnuplot* diferents: *plotcpu*, *plotmem* i *plotio*.

Nom del fitxer: plot

```
#!/bin/bash
if [ $# -ne 2 ]; then
      echo "Us $0 cpu|mem|io <fitxer.pro>"
      exit
fi
DIR=$(dirname $0)
gnuplot -e 'FILE="'$2'"' -e 'PNG=0' "$DIR/plot$1.gp"
```

Paral·lelament el següent *script* força la generació d'un gràfic *png* al finalitzar la visualització de la gràfica.

Nom del fitxer: dopng

```
#!/bin/bash
if [ $# -ne 2 ]; then
      echo "Us $0 cpu|mem|io <fitxer.pro>"
      exit
fi
```

```
DIR=$(dirname $0)
mkdir png -p
gnuplot -e 'FILE="'$2'"' -e "PNG=1" "$DIR/plot$1.gp"
```

8.5.2 Monitorització de processador

El següent *script* de *gnuplot* mostra una gràfica de l'activitat dels processadors amb les dades extretes amb *vmstat*.

Nom del fitxer: plotnet.gp

```
# Gnuplot script
# Mostra grafiques relatives a CPU
# extretes amb 'profile' basat en vmstat

# Configurar estil
set key outside                         # Mostrar llegenda fora grafica
set style data lines  # Pintar amb linies
set ytics nomirror    # Mostrar marques a eix y1
set y2tics nomirror   # Mostrar marques a eix y2

# Configurar terminal de sortida
# fitxer png o wxWidget
if (PNG==1) \
       set term png; \
       set output "png/".FILE.".cpu.png";\
else \
       set term wxt title FILE

# Etiquetes eixos Y
set ylabel "%"
set y2label "esdeveniments/segon"

# Pintar grafica
# x1y1: %cpu a usuari, %cpu a nucli, %cpu espera E/S, %cpu en repos
# x1y2: interrupcions/segon, canvis de context/segon
plot   FILE using 0:13 title "%usr" axes x1y1, \
      FILE using 0:14 title "%sys" axes x1y1, \
      FILE using 0:16 title "%iow" axes x1y1, \
      FILE using 0:15 title "%idl" axes x1y1, \
      FILE using 0:11 title "int/s" axes x2y2, \
      FILE using 0:12 title "csw/s" axes x2y2

# Si es terminal esperar pulsacio tecla
if (PNG!=1) \
       pause -1;
```

8.5.3 Monitorització de memòria i disc

El següent *script* de *gnuplot* mostra una gràfica de l'activitat de memòria i disc amb les dades extretes amb *iostat*.

Nom del fitxer: plotmem.gp

```
# Gnuplot script
# Mostra grafiques relatives a CPU
# extretes amb 'profile' basat en vmstat

# Configurar estil
set key outside                         # Mostrar llegenda fora grafica
```

```
set style data lines  # Pintar amb linies
set ytics nomirror    # Mostrar marques a eix y1
set y2tics nomirror   # Mostrar marques a eix y2

# Configurar terminal de sortida
# fitxer png o wxWidget
if (PNG==1) \
      set term png; \
      set output "png/".FILE.".mem.png";\
else \
      set term wxt title FILE

# Etiquetes eixos Y
set ylabel "MB"
set y2label "K-blocs/segon

# Pintar grafica
# y1: memoria swap, lliure, buffers, cache en KB
# y2: blocs llegits, blocs escripts, swap llegit, swap escrit
plot   FILE using 0:($3/1000) title "swap mb" axes x1y1, \
      FILE using 0:($4/1000) title "free mb" axes x1y1,  \
      FILE using 0:($5/1000) title "buff mb" axes x1y1, \
      FILE using 0:($6/1000) title "cach mb" axes x1y1, \
      FILE using 0:($9/1000) title "b-in/s" axes x2y2, \
      FILE using 0:($10/1000) title "b-out/s" axes x2y2,\
      FILE using 0:($7/1000) title "s-in/s" axes x2y2, \
      FILE using 0:($8/1000) title "s-out/s" axes x2y2

# Si es terminal esperar pulsacio tecla
if (PNG!=1) \
      pause -1;
```

8.5.4 Monitorització de xarxa

El següent *script* de *gnuplot* mostra una gràfica de l'activitat de xarxa amb les dades extretes amb *netstat* i tractades amb el *script* de la secció 8.4.3 .

Nom del fitxer: plotnet.gp

```
# Gnuplot script
# Mostra grafiques relatives a CPU
# extretes amb 'profilenet' basat en netstat

# Configurar estil
set key outside                        # Mostrar llegenda fora grafica
set style data lines  # Pintar amb linies
set ytics nomirror    # Mostrar marques a eix y1
set y2tics nomirror   # Mostrar marques a eix y2

# Configurar terminal de sortida
# fitxer png o wxWidget
if (PNG==1) \
      set term png; \
      set output "png/".FILE.".net.png";\
else \
      set term wxt title FILE

# Etiquetes eixos Y
set ylabel "K-paquets/segon"
```

```
set y2label "KB/segon"

# Pintar grafica
plot FILE using 0:($1/1000) title "krxp/s" axes x1y1, \
     FILE using 0:($3/1000) title "ktxp/s" axes x1y1, \
     FILE using 0:($2/1000) title "krxb/s" axes x1y2, \
     FILE using 0:($4/1000) title "ktxb/s" axes x1y2

# Si es terminal esperar pulsacio tecla
if (PNG!=1) \
       pause -1
```

8.6 Scripts utilitzats durant l'execució de l'experiment

Els següents *scripts* són els utilitzats durant l'experiment per a la generació de les càrregues i la mesura de les latències descrites en l'apartat 5.6.

8.6.1 Inici de l'experiment

El. següent *script* realitza l'arrancada de l'experiment d'avaluació. Bàsicament és tracta d'un guió d'execució de la resta dels que es presente en aquesta apartat: avaluació de la latència a diferents freqüències d'esdeveniment, simulació d'interacció humana, execució de procés balancejat per a la mesura del rendiment, escombrat de memòria, estrès de xarxa, estrès de disc i estrès de planificador. Finalment el guió iniciat el caracterització de tota la prova. Aquestes dades no s'han utilitzat en l'avaluació ja que es registraven per si hi havia alguna anomalia en l'experiment poder detectar el seu origen, fet que no ha ocorregut.

Nom del fitxer: alpha

```
#!/bin/bash

# Verificacio permisos d'administrador
if [ $(id -u) != 0 ]
then
       echo "Es requereixen permisos d'administrador"
       exit;
fi

# Verificacio de USB montat
if [ $(mount | grep -c "on /media/usb0 type") -ne 1 ];
then
       # Intentar montar
       mount -t vfat -o rw,umask=0 /dev/sda1 /media/usb0
       if [ $? -ne 0 ];
       then
              echo "Es requereix USB montat a /media/usb0"
              exit;
       fi
fi

# Creacio de carpeta per recollir resultats
mkdir -p ../results/$(uname -r)

# Els diferents guions es cridaran embolcallats
# en un script propi per poder detindre'ls amb
# killall -KILL sense matar accidentalment
# els cyclictest que es detenen amb -INT
```

```
# Rastreig de latencies
# Execucio cyclictest RT a
# 500, 1100, 2300, 5300, 111000, 270000 us
./cyc 500 &
./cyc 1100 &
./cyc 2300 &
./cyc 5300 &
./cyc 111000 &
./cyc 270000 &

# Simulacio HMI
# Execucio cyclictest no-RT a
# 550ms
nice -n -5 ./cycnort 550000 &

# Process balacejat CPU+I/O per a
# comparar rendiments.
# Executat sobre disc local
./uzip files/linux-3.2.32.tar.bz2 &
# Executat sobre pendrive
./uzip /media/usb0/linux-3.2.32.tar.bz2 &

# Realitzar escombrat (reservacio/alliberacio)
./stressmem &

# Realitzar estres de xarxa
# ping extern + netperf udp + ab
./stressnet

# Realitzar estres de disc consistent en..
# du / + iozone lectura 1G sequencial
# + iozone lectura/escriptura 300M aleatoria
./stressdisc &

# Realitzar estres de planificador amb hackbench
./stresssched &

# Perfilar tota la sessió
echo $(date +%y%m%d_%a_%H%M) >> ../results/$(uname -r)/alphaomega.pro
sudo chrt 1 vmstat 1 -n >> ../results/$(uname -r)/alphaomega.pro &
```

8.6.2 Finalització de l'experiment

Aquest és el *script* utilitzat per finalitzar l'experiment. Bàsicament, finalitza de manera abrupta tots els processos involucrats en el mateix amb senyals KILL[111] i de manera controlada els processos d'avaluació de latència amb senyals de INT[112].

Nom del fitxer: omega

```
#!/bin/bash

# Verificar permisos d'administrador
if [ $(id -u) != 0 ]
then
      echo "Es requereixen permisos d'administrador"
      exit;
```

111 Senyal de l'estàndard POSIX que força la finalització incondicional i immediata del procés que la rep.
112 Senyal de l'estàndard POSIX que indica la petició de finalització al procés que la rep.

```
fi

# Enviar "Ctrl-C" a tots els cyclictest
sudo killall -INT cyclictest

# Enviar "KILL" a mesures rendiment
sudo killall -KILL uzip
sudo killall -KILL bunzip2

# Enviar "KILL" a stress memoria
sudo killall -KILL stressmem
sudo killall -KILL stress

# Enviar "KILL" a stress de disc
sudo killall -KILL stressdisc
sudo killall -KILL du
sudo killall -KILL iozone

# Enviar "KILL" a stress de xarxa
sudo killall -KILL stressnet
sudo killall -KILL netperf
sudo killall -KILL netserver
sudo killall -KILL ab

# Enviar "KILL" a stress de xarxa
sudo killall -KILL stressshed
sudo killall -KILL hackbench

# Enviar "KILL" a qualsevol resta
sudo killall -KILL sleep
sudo killall -KILL time

# Enviar "KILL" a zombies
for x in $(ps -ef | grep defunct | awk '{print $3}'); do
  sudo kill -KILL $x > /dev/null
done

# Finalitzar perfilat
sudo killall -INT vmstat
```

8.6.3 Avaluació de la latència

Aquest és el *script* utilitzat per realitzar l'avaluació de la latència. Bàsicament, executa una instància de *cyclictest* segons paràmetres passats. Al finalitzar elimina entrades buides -amb nombre de repeticions cero- del histograma.

Nom del fitxer: cyc

```
#!/bin/bash
if [ $# -ne 1 ]
then
      echo "Us: "$0" <usegons>"
else
      if [ $1 -lt 500 ]
      then
            echo "Minim 500us"
            exit
      fi
```

```
      FOLD=../results/$(uname -r)
      DATE=$(date +%y%m%d_%a_%H%M)
      FNDEF=$FOLD/h$1_$DATE
      FNTMP=$FNDEF.tmp
      if [ ! -d $FOLD ]
      then
            mkdir ../cycres
            mkdir $FOLD
      fi
      echo $DATE > $FNTMP
      echo [$DATE] Inici cyclictest amb interval $1us
      sudo cyclictest -h5000000 -m -n -p51 -q -i$1 >> $FNTMP
      DATE=$(date +%y%m%d_%a_%H%M)
      echo $DATE >> $FNTMP
      grep -v "000000$" $FNTMP > $FNDEF
      rm $FNTMP
      echo [$DATE] Fi
fi
```

Paral·lament es crea un segon fitxer similar que executa *cyclictests* sense prioritat fixa i planificador de temps real (ni SCHED_FIFO, ni SCHED_OTHER).

Nom del fitxer: cycnort

```
#!/bin/bash

# Garantir permisos
if [ $(id -u) != 0 ]
then
      echo "Es requereixen permisos d'administrador"
      exit
fi

# Repeat until doomsday
while true;
do
  # Buidar caches per obligar a
  # llegir de disc
  sync
  sudo sysctl vm.drop_caches=3 > /dev/null
  # Realitzar du silencios
  # Estampar temps i executar prova
      echo "$(date)," >> ../results/$(uname -r)/du.tim
  tim nice -n -2 sudo du / --exclude="/proc" >/dev/null 2>>../results/$(uname
-r)/du.tim
done &

# Repeat until doomsday
while true;
do
  # Buidar caches per obligar a
  # llegir de disc
  sync
  sudo sysctl vm.drop_caches=3 > /dev/null
  # TEST 01 - Lectura sequencial (x2)
  # Estampar temps i executar prova
      echo "$(date)," >> ../results/$(uname -r)/iozoneread.tim
  tim nice -n 1 iozone -s 1g -r 128k -i 1 -f files/io1000M.tmp -w >/dev/null 2>>
../results/$(uname -r)/iozoneread.tim
```

```
  sleep 20
  # TEST 02 - Lectura / escriptura aleatoria (x1)
  # amb 100ms de pausa entre peticio
  # Estampar temps i executar prova
  echo "$(date)," >> ../results/$(uname -r)/iozonerand.tim
  tim nice -n 1 iozone -s 300m -r 128k -i 2 -J 50 -f files/io300M.tmp -w >/dev/null
2>> ../results/$(uname -r)/iozonerand.tim
  sleep 20
done &
```

8.6.4 Execució de càrrega balancejada

Aquest *script* és l'encarregat d'executar de manera sistemàtica l'aplicació triada per a modelar una càrrega balancejada (CPU i E/S) en l'avaluació del rendiment. A cada iteració, abans d'instanciar l'aplicació que modela la càrrega, realitza un buidat de memòria cau i memòries intermèdies per garantir que aquesta efectivament realitza lectura i escriptura de disc tal com si s'hagués iniciat per primer cop.

Nom del fitxer: uzip

```
#!/bin/bash

# Garantir permisos
if [ $(id -u) != 0 ]
then
       echo "Es requereixen permisos d'administrador"
       exit
fi

# Comprovar arguments
if [ $# -ne 1 ]
then
  echo "Us: "$0" <nom_fitxer_bz2>"
  exit
fi

# Garantir directori desti
mkdir -p ../results/$(uname -r)

# Repetir until doomsday
while true;
do
  # Buidar caches per obligar a
  # llegir de disc
  sync
  sudo sysctl vm.drop_caches=3 > /dev/null
  # Reposar durant 20 segons per donar temps
  # al sistema a comencar a reclamar memoria
  # que despres li competirem
       sleep 20
  # Estampar temps i executar prova
       echo "$(date)," >> ../results/$(uname -r)/bunzip.tim
       tim bunzip2 -f -k $1 2>> ../results/$(uname -r)/bunzip.tim
done &
```

8.6.5 Escombrat de memòria

Aquest *script* executa l'aplicació *stressmem* modificada de tal manera que es creen 7 fils d'execució amb un interval de temps de 250ms en el que es reserven i modifiquen, per tal de garantir l'apropiació de l'espai, fins 128MB de memòria. Això provoca l'exclusió de la memòria cau de disc, força el mecanisme de *write-back* i, en certes ocasions, obliga al sistema operatiu a realitzat el guardat i recuperació de pàgines de memòria a disc.

Addicionalment, s'inicia un temporitzador de tal manera que passat el temps definit a *WATCHDOG* es força la finalització de l'aplicació per evitar que el sistema entre en un allau de *swapping* tal com es detecta en les proves de caracterització.

Nom del fitxer: stressmem

```
#!/bin/bash

# Garantir permisos
if [ $(id -u) != 0 ]
then
      echo "Es requereixen permisos d'administrador"
      exit
fi

WATCHDOG=12  # Segons watchdog allau swapping
DELAY=20     # Espera entre escombrats
MEM=128M     # Memoria per thread
THR=7        # Nombre de threads
BKOFF=250000 # us retras entre threads

# Repeat until doomsday
while true;
do
  # Estresar la memoria pero sense consumir massa CPU (nice -n 2)
  nice -n 2 stress -q -m $THR -l 2 --backoff $BKOFF --vm-bytes $MEM &
  # Watchdog (per si entra en "allau") + delay
  sleep $WATCHDOG; sudo killall -q -KILL stress > /dev/null 2> /dev/null
  echo $DELAY $WATCHDOG | awk -v ORS="" '{print $1-$2}' | xargs -0 sleep
done &
```

8.6.6 Estrès de disc

Aquest *script* executa diverses aplicacions -du i iozone amb diferents paràmetres- que provoquen activitat constant a disc amb patrons d'accés diferents -lectura de i-nodes, lectura i escriptura seqüencial i aleatoria.

Nom del fitxer: stressdisc

```
#!/bin/bash

# Garantir permisos
if [ $(id -u) != 0 ]
then
      echo "Es requereixen permisos d'administrador"
      exit
fi

# Repeat until doomsday
while true;
do
```

```
  # Buidar caches per obligar a
  # llegir de disc
  sync
  sudo sysctl vm.drop_caches=3 > /dev/null
  # Realitzar du silencios
  # Estampar temps i executar prova
       echo "$(date)," >> ../results/$(uname -r)/du.tim
  tim nice -n -2 sudo du / --exclude="/proc" >/dev/null 2>>../results/$(uname
-r)/du.tim
done &

# Repeat until doomsday
while true;
do
  # Buidar caches per obligar a
  # llegir de disc
  sync
  sudo sysctl vm.drop_caches=3 > /dev/null
  # TEST 01 - Lectura sequencial (x2)
  # Estampar temps i executar prova
       echo "$(date)," >> ../results/$(uname -r)/iozoneread.tim
  tim nice -n 1 iozone -s 1g -r 128k -i 1 -f files/io1000M.tmp -w >/dev/null 2>>
../results/$(uname -r)/iozoneread.tim
  sleep 20
  # TEST 02 - Lectura / escriptura aleatoria (x1)
  # amb 100ms de pausa entre peticio
  # Estampar temps i executar prova
  echo "$(date)," >> ../results/$(uname -r)/iozonerand.tim
  tim nice -n 1 iozone -s 300m -r 128k -i 2 -J 50 -f files/io300M.tmp -w >/dev/null
2>> ../results/$(uname -r)/iozonerand.tim
  sleep 20
done &
```

8.6.7 Estrès de xarxa

Aquest *script* executa diverses aplicacions de *benchmark* de xarxa sobre el propi sistema -avaluació de servei web *apache* i enviament sistemàtic de paquets UDP a *localhost*- amb el propòsit de provocar una activitat local constant que com es comprova en les proves de caracterització provoca un alt nombre d'interrupcions i canvis de context per segon.

Nom del fitxer: stressnet

```
#!/bin/bash

# Garantir directori resultats
mkdir ../results/$(uname -r) -p

# Repeat until doomsday
while true;
do
  # 20 processos concurrents demanant peticions
  # HTTP al webserver @ localhost durant 60 segons
  # Estampar temps i executar prova
       echo "$(date)," >> ../results/$(uname -r)/ab.log
  ab -kc 20 -t 60 http://localhost/index.html >> ../results/$(uname -r)/ab.log
2>/dev/null
  sleep 20
done &
```

```
# Engegar servidor netperf
netserver

# Repeat until doomsday
while true;
do
  # Durant 60 segons enviar el maxim de paquets UDP de 1472BYTES
  # reservant 4M pels buffer d'enviament i recepcio per a
  # no aturar-se ni descartar paquets
  # Estampar temps i executar prova
      echo "$(date)," >> ../results/$(uname -r)/netperfudp.tim
  tim netperf -H localhost -t UDP_STREAM -l 60 -C -c -- -m 1472 -s 4M -S 4M
>/dev/null 2>> ../results/$(uname -r)/netperfudp.tim
  sleep 20
done &
```

8.6.8 Estrès de planificador

Aquest *script* executa sistemàticament l'aplicació *hackbench* de manera que es creen fins 400 tasques que es comuniquen entre elles mitjançant *sockets*. L'objectiu d'aquesta aplicació és la de provocar un pic d'estrès en el planificador.

Nom del fitxer: stresssched

```
#!/bin/bash

# Garantir directori resultats
mkdir ../results/$(uname -r) -p

# Repeat until doomsday
while true;
do
  # Estressar planificador amb 400 tasques
  # passant-se missatges a parells
  # Estampar temps i executar prova
      echo "$(date)," >> ../results/$(uname -r)/hackbench.tim
  tim hackbench >/dev/null 2>> ../results/$(uname -r)/hackbench.tim
  sleep 20;
done &
```

8.6.9 Report de mètriques per avaluar el rendiment

Aquest *script* s'utilitza pel capturar les mètriques seleccionades per l'avaluació del rendiment. Únicament executa la comanda *time* de GNU amb els paràmetres rebuts i indicant el format de sortida desitjat.

Nom del fitxer: tim

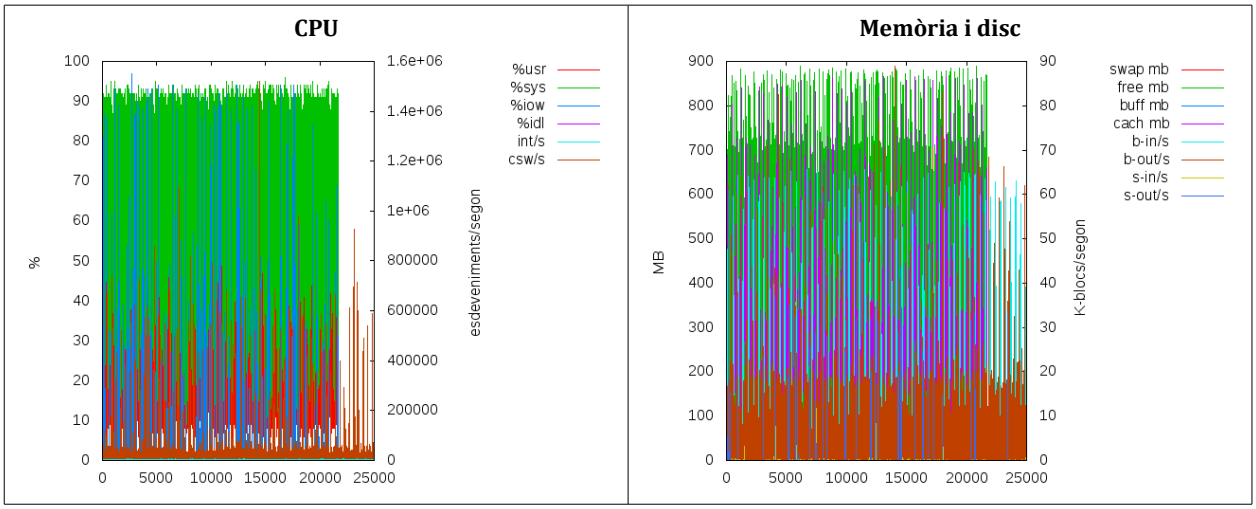
```
#!/bin/sh
#%C = comanda
#%P = percentatge CPU
#%e = segons transcorreguts
#%U = segons en espai d'usuari
#%S = segons en espai de sistema
#%w = canvis de context voluntaris
#%c = canvis de context involuntaris
#%R = fallades de pagina menors
#%F = fallades de pagina majors
/usr/bin/time -f "%C, %Pp, %ee, %Uu, %Ss, %ar, %bw, %wv, %ci, %Rm, %FM" $@
```

8.7 Registre d'activitats durant les proves

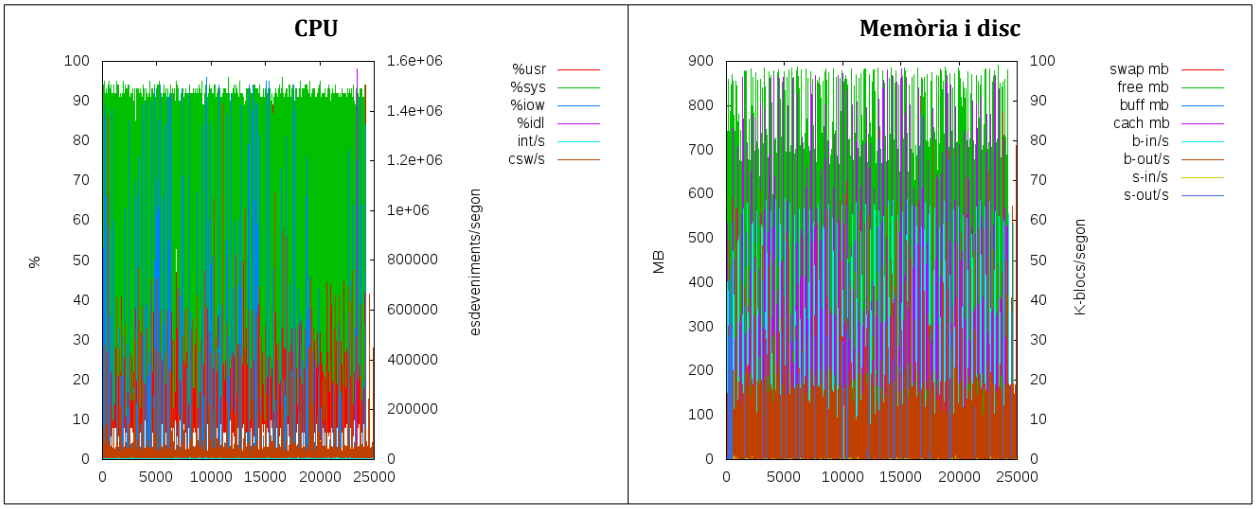
Amb el propòsit de monitoritzar l'experiment final de tal manera que es pogués detectar qualsevol anomalia durant el mateix es realitza el registre al llarg de la durada d'aquest de les mètriques utilitzades en la caracterització de càrregues. En tot els casos es verifica que l'activitat, a grosso modo, és correcta, és a dir, obté els valors modelats en la caracterització del experiment.

Com a fet notable simplement comentar l'augment en l'ordre de magnitud del nombre de canvis de context en el nucli amb el pedaç RT possiblement produït pel fet d'executar interrupcions en context de procés i passar aquestes, per tant, a requerir canvis de context.

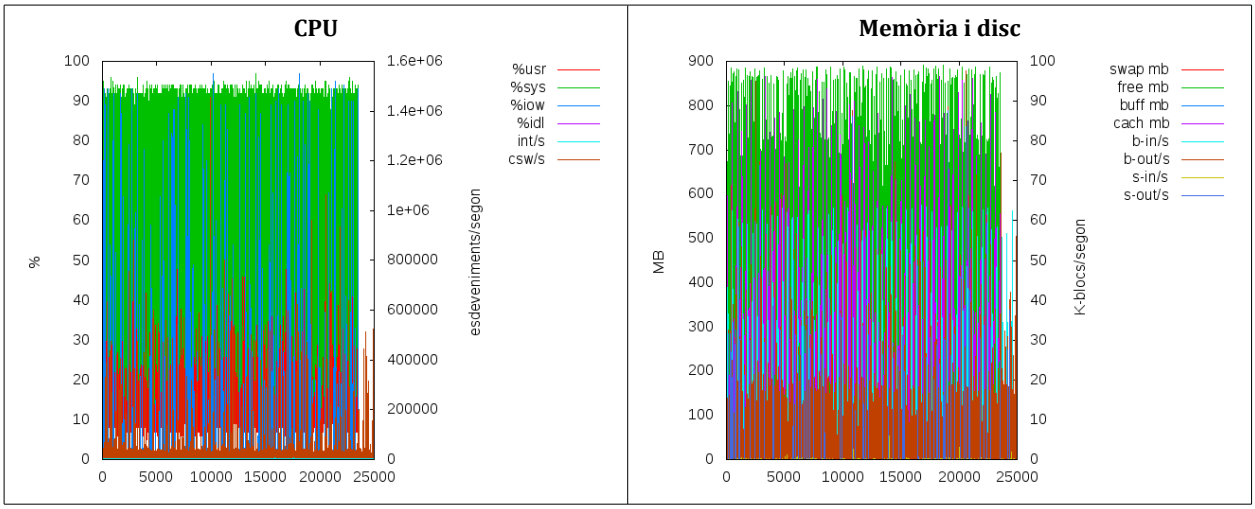
8.7.1 No preempt



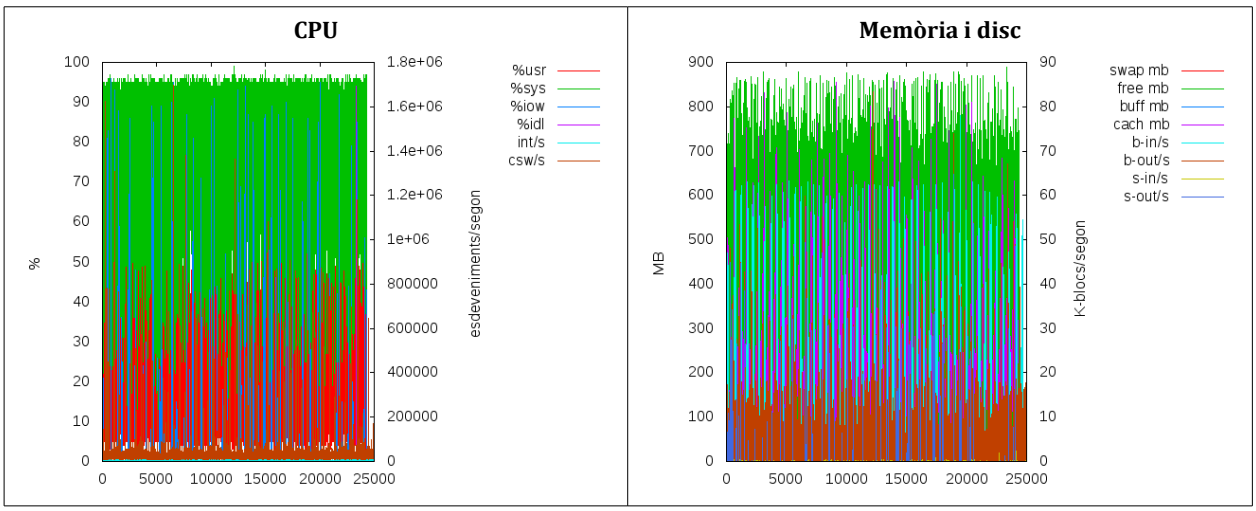
8.7.2 Voluntary



8.7.3 Preempt



8.7.4 RT



8.8 Formules utilitzades en la fulla de càlcul per als estadístics

Les formules que a continuació és mostre s'han d'aplicar sobre tota la columna de dades de les mostres registrades durant l'experiment. D'aquestes columnes hi han dades que són les pròpies registrades mentre d'altres són càlculs intermedis realitzats a partir d'aquestes. Aquestes fòrmules són compatibles amb Open Office Calc tot i Microsoft Excel, versió en anglès.

8.8.1 Latència

Valors utilitzats:

- X_i : registre de valor de latència
- F_i : registre de freqüència d'aparició de X_i
- Log(X_i)*Fi) : =LOG(X_i)*F_i
- X^2 : =POWER(X_i, 2)
- $FACUM_i$: = F_i + F_{i-1}
- Percentil : variable d'entrada per el càlcul del percentil donada en l'interval [0..1].

Estadístic	Fórmula
Pitjor cas	=MAX(Latència)
Mitjana geomètrica	=POWER(10, SUM(Log(Xi)*Fi) / SUM(Fi)
Mitjana aritmètica	=SUMPRODUCT(Xi, Fi) / SUM(Fi)
Desviació estàndard	=SUMPRODUCT(Xi^2, Fi) / SUM(Fi) – POWER(Log(Xi)*Fi, 2)
Percentils[113]	=INDIRECT(ADDRESS(MATCH((SUM(Fi) + 1) * Percentil, FACUMi), 1))

8.8.2 Rendiment

En el cas del rendiment al no tenir les dades agrupades per freqüència de repetició s'han utilitzat simplement les pròpies formules estadístiques que proporciona la fulla de càlcul.

Estadístic	Fórmula
Màxim	=MAX(COLUMNA_VAR)
Mínim	=MIN(COLUMNA_VAR)
Mediana	=MEDIAN(COLUMNA_VAR)
Mitjana geomètrica	=GEOMEAN(COLUMNA_VAR)
Mitjana aritmètica (μ)	=AVERAGE(COLUMNA_VAR)
Desviació estàndard (σ)	=STDEV(COLUMNA_VAR)
Percentil	=PERCENTILE(COLUMNA_VAR, Percentil)
Coeficient de Pearson	=PEARSON(COLUMNA_VAR_A, COLUMNA_VAR_B)

113 Inclòs mediana que és percentil 50%.

8.9 Estadístiques

En aquest apartat de l'annex es presenten les estadístiques completes recollides però no presentades en la memòria.

8.9.1 Latència

NO PREEMPT

	Mitges		Dispersió	Percentils							Mostreig		
	Mitja Geomètrica	Mitja Aritmètica	Desviació Estàndard	Mediana (50%)	2 Sigma (68,27%)	3 Sigma (95,45%)	1 de 10^5 99,999%	1 de 10^7 99,99999	Pitjor cas	Nombre De mostres	# Fallades (>75% *rate)* en 8h	1 Fallo cada (segons)	1 Fallo cada (minuts)
500RT	19,198	31,944	772,327	16,000	20,000	55,000	123453,000	168793,000	169676,000	57679312	283357,000	0,102	0,002
1100RT	19,563	25,655	84,686	17,000	20,000	52,000	14713,000	23460,000	24352,000	26217884	6843,000	4,209	0,070
2300RT	19,954	25,917	53,527	17,000	20,000	53,000	3886,000	21962,000	23349,000	12538988	196,000	146,939	2,449
5300RT	19,668	25,841	54,529	16,000	19,000	55,000	3627,000	21322,000	21322,000	5441449	55,000	523,636	8,727
111000RT	19,792	25,740	58,068	17,000	19,000	52,000	982,000	17270,000	17270,000	259817	1,000	28800,000	480,000
270000RT	19,961	25,799	51,422	17,000	20,000	53,000	1293,000	7616,000	7616,000	106813	0,000	0,000	0,000
550000NORT	96,472	2868,859	43542,491	71,000	78,000	384,000	999787,000	999787,000	999787,000	51039	174,000	165,517	2,759

VOLUNTARY

	Mitges		Dispersió	Percentils							Mostreig		
	Mitja Geomètrica	Mitja Aritmètica	Desviació Estàndard	Mediana (50%)	2 Sigma (68,27%)	3 Sigma (95,45%)	1 de 10^5 99,999%	1 de 10^7 99,99999	Pitjor cas	Nombre De mostres	# Fallades (>75% *rate)* en 8h	1 Fallo cada (segons)	1 Fallo cada (minuts)
500RT	17,752	30,906	758,507	15,000	19,000	41,000	104840,000	121985,000	122721,000	63969013	329531,000	0,087	0,001
1100RT	17,525	22,975	86,760	15,000	18,000	38,000	13582,000	36282,000	36891,000	29076828	12246,000	2,352	0,039
2300RT	19,111	24,483	57,063	16,000	20,000	41,000	2897,000	35020,000	36831,000	13906310	219,000	131,507	2,192
5300RT	18,068	23,573	58,081	15,000	18,000	39,000	2705,000	36372,000	36372,000	6034814	49,000	587,755	9,796
111000RT	18,507	23,706	54,092	16,000	18,000	38,000	6268,000	10721,000	10721,000	288149	4,000	7200,000	120,000
270000RT	18,933	24,041	49,395	16,000	19,000	39,000	884,000	7192,000	7192,000	118461	0,000	0,000	0,000
550000NORT	91,198	3049,125	44769,735	68,000	73,000	404,000	992231,000	992231,000	992231,000	56570	215,000	133,953	2,233

PREEMPT

	Mitges		Dispersió	Percentils							Mostreig		
	Mitja Geomètrica	Mitja Aritmètica	Desviació Estàndard	Mediana (50%)	2 Sigma (68,27%)	3 Sigma (95,45%)	1 de 10^5 99,999%	1 de 10^7 99,99999	Pitjor cas	Nombre De mostres	# Fallades (>75% *rate)* en 8h	1 Fallo cada (segons)	1 Fallo cada (minuts)
500RT	18,691	41,927	1219,234	16,000	19,000	33,000	134468,000	183245,000	183908,000	63363850	398194,000	0,072	0,001
1100RT	18,164	23,607	105,120	16,000	18,000	32,000	16451,000	27957,000	28479,000	28801759	26936,000	1,069	0,018
2300RT	20,725	26,034	49,243	18,000	21,000	41,000	1120,000	2938,000	3350,000	13774754	20,000	1440,000	24,000
5300RT	18,959	23,997	49,922	17,000	19,000	32,000	1127,000	4666,000	4666,000	5977724	1,000	28800,000	480,000
111000RT	24,953	29,082	43,921	22,000	25,000	38,000	1113,000	1202,000	1202,000	285423	0,000	0,000	0,000
270000RT	20,839	25,882	45,451	17,000	20,000	40,000	1072,000	1153,000	1153,000	117340	0,000	0,000	0,000
550000NORT	87,754	2027,877	35398,780	70,000	74,000	373,000	998117,000	998117,000	998117,000	56604	137,000	210,219	3,504

RT

	Mitges		Dispersió	Percentils							Mostreig		
	Mitja Geomètrica	Mitja Aritmètica	Desviació Estàndard	Mediana (50%)	2 Sigma (68,27%)	3 Sigma (95,45%)	1 de 10^5 99,999%	1 de 10^7 99,99999	Pitjor cas	Nombre De mostres	# Fallades (>75% *rate)* en 8h	1 Fallo cada (segons)	1 Fallo cada (minuts)
500RT	17,895	18,666	6,581	16,000	18,000	30,000	211,000	245,000	295,000	62838228	0,000	0,000	0,000
1100RT	18,010	18,595	6,033	16,000	18,000	27,000	212,000	247,000	257,000	28562826	0,000	0,000	0,000
2300RT	19,158	19,838	6,612	17,000	19,000	30,000	213,000	245,000	247,000	13660490	0,000	0,000	0,000
5300RT	19,823	20,448	6,276	18,000	20,000	30,000	213,000	247,000	247,000	5928138	0,000	0,000	0,000
111000RT	18,586	18,934	4,953	17,000	18,000	25,000	212,000	218,000	218,000	283055	0,000	0,000	0,000
270000RT	17,985	18,319	5,768	17,000	17,000	22,000	212,000	217,000	217,000	116367	0,000	0,000	0,000
550000NORT	102,252	113,416	102,368	96,000	100,000	200,000	8081,000	8081,000	8081,000	57126	0,000	0,000	0,000

8.9.2 Rendiment descompressió a disc on resideix swap

ESTADISTICS NO_PREEMPT HD													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow-Down	
10,000	807,399	72,618	7,459	81,470	514,448	235,000	370432,000	1064,000	2,000	164,923	8,493	1,204	1,587	MILLOR
7,000	1131,113	76,635	9,239	84,887	699,618	343,000	406393,000	1087,000	9,000	366,813	11,898	1,324	1,693	PITJOR
8,000	1005,963	75,258	8,553	83,676	632,556	289,500	394606,000	1076,500	4,000	288,561	10,582	1,294	1,626	MEDIANA
7,941	997,296	75,099	8,414	83,540	627,441	288,830	393789,700	1073,872	3,489	284,259	10,490	1,286	1,629	MIT.GEO.
7,964	999,634	75,105	8,432	83,544	629,139	289,714	393885,607	1073,893	3,929	286,951	10,515	1,286	1,630	MIT.ARIT.
0,637	68,195	0,937	0,562	0,808	46,310	22,816	8800,958	6,822	1,923	37,612	0,717	0,028	0,026	DESV.EST.
8,000	1041,126	75,692	8,809	83,895	653,415	297,299	399853,117	1078,000	5,000	307,515	10,951	1,304	1,643	2 SIGMA = 68,27
9,000	1085,635	76,200	9,165	84,591	690,946	321,629	405215,684	1081,000	6,772	331,910	11,420	1,319	1,673	3 SIGMA = 95,45
-0,849	1,000	0,564	0,174	0,775	0,849	0,245	0,823	-0,055	0,063	0,751	1,000	0,347	-0,161	PEARSON

ESTADISTICS VOLUNTARY HD													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
8,000	955,696	74,821	7,992	83,704	547,917	248,000	392950,000	1064,000	2,000	243,679	10,057	1,237	1,540	MILLOR
7,000	1158,819	77,074	9,436	86,118	718,904	1565,000	414208,000	1247,000	10,000	382,177	12,189	1,377	1,682	PITJOR
8,000	1026,958	76,135	8,794	84,964	659,456	281,500	402449,000	1076,000	5,000	296,683	10,802	1,289	1,626	MEDIANA
7,755	1031,878	76,156	8,725	84,895	650,815	332,031	402018,912	1079,271	3,906	294,124	10,854	1,286	1,631	MIT.GEO.
7,767	1033,134	76,158	8,732	84,898	652,186	378,567	402061,033	1079,700	4,467	296,051	10,867	1,286	1,631	MIT.ARIT.
0,430	52,097	0,601	0,360	0,624	42,329	278,827	5919,558	32,367	2,285	35,028	0,548	0,028	0,028	DESV.EST.
8,000	1056,205	76,534	8,887	85,191	677,979	297,395	405841,811	1078,000	5,798	307,823	11,110	1,298	1,645	2 SIGMA = 68,27
8,000	1122,479	77,072	9,200	85,835	704,842	921,245	411016,489	1086,764	9,042	368,994	11,807	1,325	1,672	3 SIGMA = 95,45
-0,753	1,000	0,602	0,239	0,718	0,742	0,106	0,777	0,052	0,017	0,578	1,000	0,163	-0,034	PEARSON

ESTADISTICS PREEMPT HD													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
8,000	1007,592	75,588	9,669	86,000	609,807	394,000	403468,000	1064,000	2,000	266,021	10,599	1,258	1,580	MILLOR
6,000	1300,519	78,208	11,302	88,820	756,886	681,000	427159,000	1090,000	11,000	455,238	13,680	1,350	1,660	PITJOR
8,000	1086,152	76,814	10,399	87,193	692,124	474,000	416186,000	1077,000	6,000	325,744	11,425	1,296	1,626	MEDIANA
7,374	1110,868	76,838	10,352	87,205	692,232	480,671	413975,310	1077,584	5,936	329,333	11,685	1,297	1,623	MIT.GEO.
7,407	1113,439	76,841	10,360	87,207	693,352	483,815	414026,074	1077,593	6,296	332,880	11,712	1,298	1,624	MIT.ARIT.
0,694	78,560	0,654	0,416	0,708	40,012	58,405	6611,810	4,352	2,109	51,746	0,826	0,028	0,025	DESV.EST.
8,000	1127,953	77,195	10,542	87,395	719,334	496,753	417509,266	1078,000	7,000	339,047	11,865	1,309	1,643	2 SIGMA = 68,27
8,000	1273,887	78,015	11,087	88,393	753,977	559,889	424431,746	1083,817	9,817	446,012	13,400	1,350	1,657	3 SIGMA = 95,45
-0,924	1,000	0,643	0,398	0,827	0,803	0,519	0,874	-0,114	0,449	0,886	1,000	0,655	-0,542	PEARSON

ESTADISTICS RT HD													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
7,000	1230,384	76,105	10,629	87,278	773,696	24290,000	431926,000	1071,000	2,000	369,410	12,942	1,274	1,533	MILLOR
5,000	1595,688	77,718	12,365	90,087	927,016	32031,000	467631,000	1085,000	10,000	597,588	16,785	1,407	1,661	PITJOR
6,000	1380,966	77,144	11,382	88,530	832,038	28865,000	446883,000	1076,000	6,000	462,873	14,526	1,335	1,601	MEDIANA
5,737	1396,870	77,094	11,345	88,453	842,744	28418,277	446855,139	1076,948	5,879	462,684	14,693	1,332	1,604	MIT.GEO.
5,762	1399,448	77,095	11,353	88,456	844,268	28485,238	446919,667	1076,952	6,333	466,724	14,720	1,333	1,604	MIT.ARIT.
0,539	87,381	0,455	0,450	0,715	52,193	1986,889	7796,642	3,106	2,331	63,498	0,919	0,031	0,029	DESV.EST.
6,000	1420,647	77,315	11,600	88,798	870,921	29522,462	451201,828	1077,654	7,654	484,757	14,943	1,342	1,616	2 SIGMA = 68,27
6,090	1528,702	77,678	11,907	89,506	922,970	31329,390	456585,420	1083,180	10,000	584,341	16,080	1,377	1,645	3 SIGMA = 95,45
-0,881	1,000	0,586	0,573	0,733	0,691	0,634	0,699	0,455	0,170	0,800	1,000	0,494	-0,403	PEARSON

8.9.3 Rendiment descompressió a disc extern

ESTADISTICS NO_PREEMPT USB													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
12,000	601,667	71,118	3,453	75,702	390,167	240,000	350262,000	1069,000	2,000	120,698	2,525	1,173	1,617	MILLOR
10,000	749,284	74,451	5,049	78,120	530,148	464,000	371501,000	1086,000	8,000	187,370	3,145	1,266	1,721	PITJOR
11,000	670,832	72,571	3,879	76,722	450,057	309,000	359950,000	1072,000	2,000	137,454	2,816	1,207	1,676	MEDIANA
10,824	674,672	72,656	3,983	76,671	454,913	313,101	360686,634	1076,263	3,006	141,481	2,832	1,211	1,675	MIT.GEO.
10,846	675,782	72,662	4,006	76,674	456,624	314,436	360738,256	1076,282	3,410	142,483	2,836	1,211	1,675	MIT.ARIT.
0,709	39,290	0,909	0,440	0,699	40,281	31,226	6182,549	6,557	1,874	17,513	0,165	0,027	0,029	DESV.EST.
11,000	699,665	72,995	4,203	76,862	481,003	319,713	364266,750	1083,000	3,943	149,773	2,937	1,218	1,685	2 SIGMA = 68,27
12,000	743,168	74,284	4,830	77,929	520,521	348,084	370062,444	1085,271	7,000	171,327	3,119	1,258	1,722	3 SIGMA = 95,45
-0,915	1,000	0,648	-0,013	0,834	0,904	-0,067	0,771	0,125	0,272	0,130	1,000	-0,324	0,498	PEARSON

ESTADISTICS VOLUNTARY USB													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
12,000	617,084	71,272	3,556	75,724	397,111	213,000	349286,000	1069,000	2,000	114,559	2,590	1,160	1,560	MILLOR
1,000	747,833	74,405	4,449	78,217	538,652	469,000	370428,000	1312,000	11,000	235,534	3,126	1,332	1,734	PITJOR
11,000	687,496	72,932	4,035	76,867	465,176	299,000	361826,000	1080,500	4,000	137,549	2,886	1,205	1,681	MEDIANA
10,191	686,865	72,867	4,014	76,902	466,703	298,102	361254,554	1082,483	3,541	140,666	2,883	1,207	1,680	MIT.GEO.
10,571	687,813	72,871	4,026	76,905	468,367	301,095	361308,310	1083,024	4,167	142,542	2,887	1,207	1,680	MIT.ARIT.
1,655	36,308	0,799	0,315	0,653	40,065	44,093	6298,141	36,822	2,439	24,886	0,152	0,035	0,036	DESV.EST.
11,000	708,039	73,237	4,232	77,353	491,129	311,000	364240,205	1082,000	5,000	151,824	2,972	1,218	1,694	2 SIGMA = 68,27
12,000	742,402	74,274	4,455	77,964	533,053	366,076	370517,806	1087,211	8,135	179,821	3,116	1,261	1,731	3 SIGMA = 95,45
-0,456	1,000	0,699	-0,087	0,813	0,795	0,273	0,735	0,117	0,193	0,159	1,000	-0,136	0,279	PEARSON

ESTADISTICS PREEMPT USB													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
12,000	609,746	71,378	4,296	76,907	420,971	229,000	360113,000	1070,000	2,000	109,777	2,708	1,143	1,623	MILLOR
9,000	815,361	74,915	5,936	79,641	579,447	440,000	381804,000	1095,000	12,000	203,635	3,422	1,272	1,754	PITJOR
11,000	706,678	73,238	4,933	78,159	474,028	337,000	368736,000	1083,000	6,000	156,570	2,966	1,222	1,670	MEDIANA
10,542	708,705	73,239	4,946	78,215	475,140	329,741	368315,427	1099,049	5,105	153,057	2,975	1,218	1,671	MIT.GEO.
10,561	710,074	73,244	4,967	78,218	477,038	332,902	368375,780	1106,049	5,634	154,818	2,980	1,218	1,671	MIT.ARIT.
0,634	44,745	0,839	0,462	0,758	43,318	46,521	6751,681	156,336	2,343	23,233	0,188	0,031	0,032	DESV.EST.
11,000	737,158	73,530	5,284	78,596	498,022	353,616	371400,800	1084,000	6,000	172,529	3,094	1,231	1,686	2 SIGMA = 68,27
11,180	782,856	74,513	5,649	79,374	551,888	417,040	380903,800	1090,080	9,180	186,473	3,286	1,272	1,725	3 SIGMA = 95,45
-0,911	1,000	0,621	0,184	0,799	0,861	-0,044	0,792	-0,059	0,197	0,294	1,000	-0,125	0,308	PEARSON

ESTADISTICS RT USB													
% CPU	Ellapsed (s)	Sys(s)	User (s)	Run (s)	Sched Wait (s)	CS V	CS I	Min PF	Maj PF	No Sched Wait(s)	Penalized	No Sched Slow Down	Sched Slow Down	
10,000	772,459	71,362	5,229	78,028	522,482	6601,000	386840,000	1070,000	2,000	141,312	3,242	1,159	1,659	MILLOR
7,000	1122,717	75,268	7,192	81,624	801,338	11592,000	421172,000	1148,000	10,000	295,013	4,712	1,264	1,759	PITJOR
8,000	905,994	73,708	5,874	79,470	629,600	8633,500	401785,000	1082,000	6,000	193,182	3,803	1,220	1,693	MEDIANA
8,189	916,126	73,663	5,836	79,526	640,034	8878,582	402418,004	1083,798	4,628	192,988	3,845	1,213	1,699	MIT.GEO.
8,219	919,898	73,670	5,854	79,531	644,598	8964,625	402510,188	1083,906	5,156	195,768	3,861	1,213	1,700	MIT.ARIT.
0,706	85,701	1,031	0,472	0,957	78,847	1272,758	8762,730	15,781	2,201	34,312	0,360	0,034	0,036	DESV.EST.
8,164	955,448	74,055	6,042	79,781	665,706	9446,473	406536,318	1083,164	6,000	212,033	4,010	1,238	1,716	2 SIGMA = 68,27
9,000	1092,398	75,022	6,555	81,225	780,245	11082,769	416807,861	1115,222	8,590	247,347	4,585	1,256	1,759	3 SIGMA = 95,45
-0,888	1,000	0,763	0,176	0,909	0,916	0,866	0,804	0,299	0,246	0,368	1,000	-0,192	0,375	PEARSON