Codi: 05.001

Assignatura: Estructura de la Informació

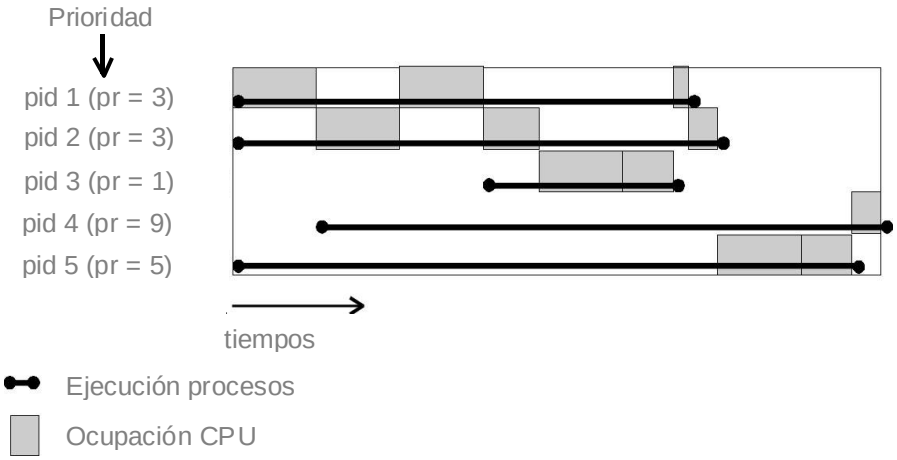
Curso: 2009/2010 1r semestre

PAC1

Exercici 1 [3 punts]

Els sistemes operatius multiprocés poden incorporar en qualsevol moment, nous processos al conjunt de processos que ja s'estigui executant i competint per tant, per la CPU. Disposen, d'un mecanisme que els permet escollir quin procés podrà usar la CPU i durant quant de temps. El mecanisme té en compte el nivell de prioritat que s'assigna a cada procés en el moment que s'inicia l'execució. Suposem que estem treballant amb un sistema operatiu que treballa amb 10 nivells de prioritat per procés, de 0 a 9, sent 0 la més alta i 9 la més baixa.

El sistema operatiu assignarà la CPU de la manera següent: seleccionarà un procés en execució (a continuació veurem com) i l'executarà durant un interval de temps limitat. Si el procés acaba, es passarà a seleccionar un altre procés. Si no acaba, es desassignarà de la CPU, es tornarà a posar a la seqüència de processos que esperen i es seleccionarà un altre procés a ser servit (és a dir, ser executat).



Quan el sistema operatiu hagi de seleccionar un procés a servir, seleccionarà aquell que té la prioritat més alta (més propera a 0). Si n'hi ha més d'un, podrà seleccionar qualsevol d'ells (advertiu que això és una simplificació de la realitat, on normalment les polítiques per seleccionar el procés solen ser més complexes).

D'aquesta manera, veiem que un procés amb una certa prioritat no podrà ser servit a no ser que hagin finalitzat els processos que tenen més prioritat que ell. D'altra banda, també és possible eliminar un procés de la cua d'espera.

Es volen tenir les següent funcionalitats:

- Crear la seqüència buida.
- Executar un procés, és a dir, afegir-lo a la seqüència donada, a més de la informació del procés (per exemple, la direcció de memòria a partir de la qual resideix el codi a

executar), la seva prioritat i el temps de CPU que necessita. S'ha de retornar un identificador únic, anomenat PID - Process Identifier -, que identifica el procés en la seqüència.

- Enviar un procés a la CPU per a què es serveixi.
- Tornar un procés a espera quan ha acabat el temps màxim d'ocupació de la CPU.
- Eliminar un procés quan acaba la seva execució.
- Eliminar un procés identificat per PID de la seqüència.
- Permetre fer un llistat per pantalla dels processos que estan en espera ordenats de més a menys prioritat (a igual prioritat no importa l'ordre).

El TAD no decideix durant quant de temps es serveix un procés ni si un procés acaba la seva execució o es torna a la seqüència de processos. Això es decideix de manera externa al nostre TAD i per tant no ha de ser tractat.

Atès que el nombre de processos en espera no serà més gran que N, es demana:

Apartat (1.1) Defineix la signatura (operacions i sintaxi) del TAD i situa'l en la jerarquia de classes de la biblioteca de TADs de l'assignatura (d'on s'hauria d'estendre, relacions amb altres, etc.). Especialitza (substitueix) els paràmetres de les classes parametritzades de la jerarquia, sempre que et sigui possible, pels tipus que consideris més adients.
Raona la teva resposta.

Signatura del TAD:

TAD WaitingQueue{

WaitingQueue();

int runProcess(long codeAddress, long totalCpuTime, byte priority);

boolean cpuIsBusy();

int processInCpu();

void toCpu();

void toWaitingQueue();

void finishProcess();

void deleteProcess(int pid);

Iterador waitingProcessList();

}

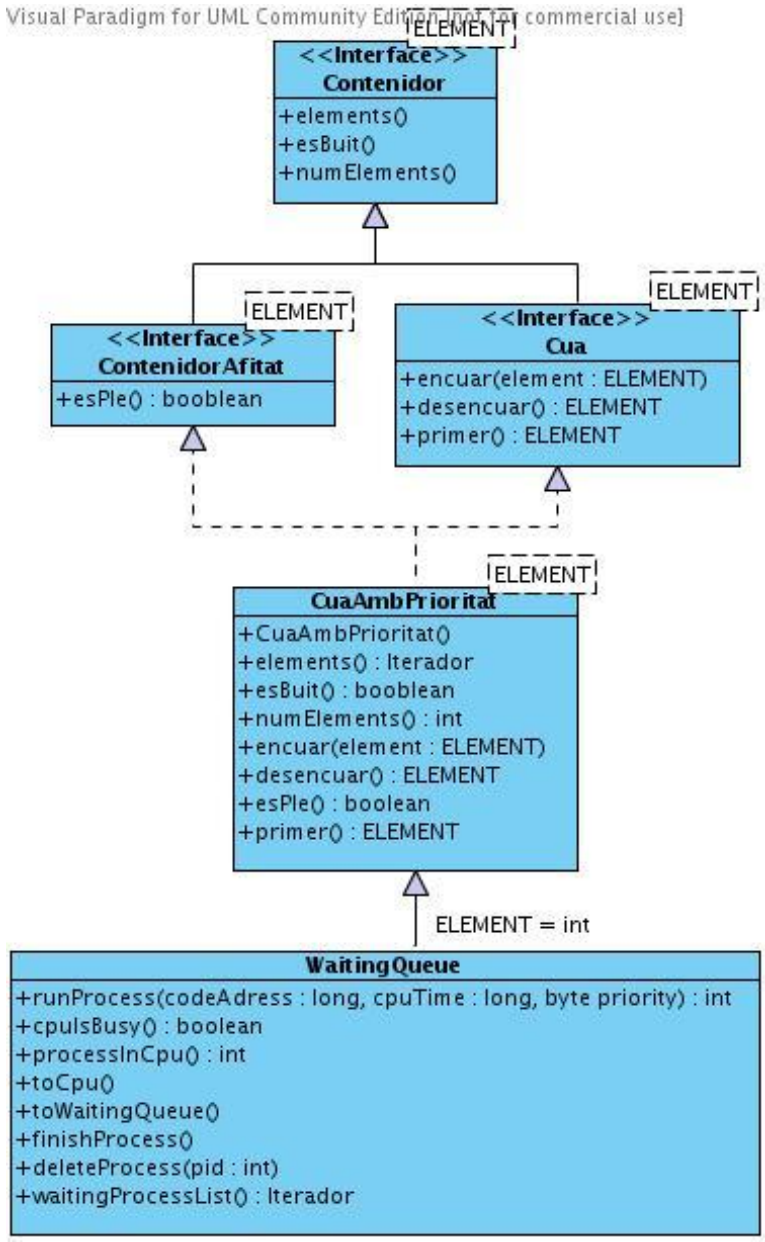
Noteu que s'han afegit algunes operacions necessàries per identificar el procés que es troba a la CPU.

Aquest TAD comparteix trets essencials amb la CuaAmbPrioritat en el sentit que conté elements que poden ordenar sota cert criteri i que sempre manté seleccionat un element que pugui situar-se en la primera posició atenent al criteri d'ordenació.

Però presenta diferències respecte a la CuaAmbPrioritat. Les operacions són molt específiques pel que fa als paràmetres i els elements usats. A més cal afegir la funcionalitat referent a l'ús de la CPU. Per això s'ha optat per situar aquest TAD com una especialització d'una CuaAmbPrioritat a la qual se li ha afegit tota la funcionalitat específica del nostre TAD.

S'ha suposat que el TAD treballa amb identificadors de processos (PIDS) i disposa de mètodes per a consultar informació del procés a partir del PID.

Per això WaitingQueue a més d'heretar de CuaAmbPrioritat especialitza el paràmetre ELEMENT establint-lo com a tipus sencer.



Apartat 1.2) Fes l'especificació del TAD. Utilitza una especificació com la de l'apartat 1.2.3. Es valorarà especialment la concisió (absència d'elements redundants o innecessaris), precisió (definició correcta del resultat de les operacions), completesa (consideració de tots els casos possibles en què es pot executar cada operació) i manca d'ambigüitats (coneixement exacte de com es comporta cada operació en tots els casos possibles) de la solució. És important respondre aquest apartat usant una descripció condicional i no procedimental. L'experiència ens demostra que no sempre resulta fàcil distingir entre les dues descripcions, és per això que fem especial èmfasi insistint que poseu molta atenció a les vostres definicions. A títol d'exemple indicarem que la descripció condicional (la correcta a utilitzar en el contracte) d'omplir un got buit amb aigua seria:

@ pre el got es troba buit.
@ post el got és ple d'aigua.

En canvi una descripció procedimental (incorrecta per utilitzar en el contracte) tindria una forma semblant a:

@ pre el got hauria de trobar-se buit, si no es trobés buit s'hauria de buidar.
@ post s'acosta el got a l'aixeta i s'hi tira aigua fins que estigui ple.

Cal també tenir en compte que un contracte hauria de disposar d'invariant sempre que aquesta fos necessària per descriure el TAD.

TAD WaitingQueue{

@invariant tots els elements continguts tenen un valor associat que anomenem prioritat comprés entre 0 i 9 (*$all(pid:waitingProcessList(), 0≤priority(pid)≤9).*

@pre cert

@post el contenidor creat està buit *($this.elementos()=0).*

WaitingQueue();

@pre *codeAddress* és l'adreça de memòria on comença el codi del procés, *totalCpuTime* el temps que el procés necessitarà usar la CPU i *priority* es troba dins de l'interval [0, 9]

@post El sistema ha reservat els recursos necessaris per a l'execució del procés i manté guardada la informació identificada amb un identificador numèric (PID) el qual passa a formar part de *$this.* El procés es troba en espera d'usar la CPU.

$return l'identificador numèric (PID) amb el qual el sistema ha identificat la informació necessària per executar el procés.

int runProcess(long codeAddress, long totalCpuTime, byte priority);

@pre cert

@post $return hi ha algun procés executant-se en la CPU.

boolean cpuIsBusy()

@pre hi ha un procés executant-se en la CPU *(cpuIsBusy())*.

@post *$return* l'identificador numèric (PID) del procés que s'està executant en la CPU

int processInCpu();

@pre el contenidor no està buit (*elements ()> 0*) i no hi ha cap procés executant-se en la CPU *(!cpuIsBusy())*

@post s'està executant un procés a la CPU (*cpuIsBusy ()*). El procés que s'està executant a la CPU té una prioritat menor o igual a qualsevol altre procés contingut al contenidor *($all(pid:waitingProcessList(), priority(pid) <= priority(processInCpu()))*. Hi ha un procés menys esperant *($old(elementos()) = elementos()+1)*. El procés que s'està executant en la CPU no està esperant *(!$exists(pid:waitingProcessList(), pid=processInCpu()))*.

void toCpu();

@pre hi ha un procés executant-se en la CPU *(cpuIsBusy())*.

@post no hi ha cap procés executant-se a la CPU *(!cpuIsBusy())*. El procés que estava a la CPU es troba esperant *($exists(pid:waitingProcessList(), pid=$old(processInCpu())))*.

void toWaitingQueue();

@pre hi ha un procés executant-se en la CPU *(cpuIsBusy())*.

@post no hi ha cap procés executant-se a la CPU *(!cpuIsBusy())*. El procés que estava a la CPU ha finalitzat i per tant no es troba esperant *(!$exists(pid:waitingProcessList(), pid=$old(processInCpu())))*. S'han alliberat els recursos associats al procés acabat.

void finishProcess();

@pre pid és un identificador que es troba a la cua d'espera *($exists(id:waitingProcessList(), id=pid))*

@post Hi ha un procés menys esperant ($old(elementos()) = elementos()+1). El procés identificat por pid no està esperant (!$exists(id:waitingProcessList(), id=pid)). S'han alliberat els recursos associats al procés identificat per pid.

void deleteProcess(int pid);

@pre cert

@post $return Iterador contenint tots els processos que es troben esperant l'ús de la CPU.

Iterador waitingProcessList();

}

Exercici 2 [2,5 punts]

Es vol implementar un algorisme per avaluar un polinomi de grau n. Els coeficients del polinomi es guarden en un vector de n +1 elements. Per exemple, un polinomi de grau 2 com $2x^2 + 3x - 5$ es guardaria en un vector de tres posicions V = [2, 3, -5]. A continuació es mostra una solució al problema:

```
(1) long evalua(int[] vector, int x, int grau){
(2)       long s, pot;
(3)       int i, j;
(4)       s=0;
(5)       i=0;
(6)       while(i<=grau){
(7)            j=0;
(8)            pot=1;
(9)            while(j< i){
(10)                pot *= x;
(11)                j++;
(12)           }
(13)           s+=pot*vector[i];
(14)           i++;
(15)      }
(16)      return s;
(17)}
```

Es demana:

Apartat 2.1) Indicar detalladament la complexitat asimptòtica de la solució proposada.

Els costos de les instruccions (4), (5), (7), (8), (10), (11), (13), (14) i (16) són constants. També ho són el cost de les avaluacions dels bucles. Per tant:

Cost del bucle de la instrucció (12) = $T(_{12}) = 1 + \sum_{j=0}^{i-1}(+1+1) = 1+3i$

Cost del bucle de la instrucció (6) serà:

$$T(_6) = 1 + \sum_{i=1}^{n+1}\left(1+1+1+1+\sum_{j=0}^{i-1}(+1+1)\right) = 1 + \sum_{i=1}^{n+1}(+1+1+1+3i)$$

$$T(_6) = 1 + n + n + n + n + \frac{3(1+n+1)(n+1)}{2} = 1 + 4n + \frac{3n^2+9n+6}{2} = \frac{3n^2+17n+8}{2}$$

I el cost total del bucle serà:

$$T(n) = 1+1+1+\sum_{i=1}^{n+1}\left(1+1+1+1+\sum_{j=0}^{i-1}(+1+1)\right) = 2+\frac{3n^2+17n+8}{2} = \frac{3n^2+17n+12}{2} \in O(n^2)$$

El cost asimptòtic és doncs $O(n^2)$

Apartat 2.2) Proposar un algoritme que millori l'eficiència asimptòtica temporal de la solució proposada, indicant la eficiència (no cal fer el càlcul detallat).

```
(1)   long evaluaEf(int[] vector, int x, int grado){
(2)       long s, pot;
(3)       int i;
(4)       pot=1;
(5)       s=0;
(6)       i=0;
(7)       while(i<=grado){
(8)           s+=pot*vector[i];
(9)           pot*=x;
(10)              i++;
(11)          }
(12)          return s;
(13)      }
```

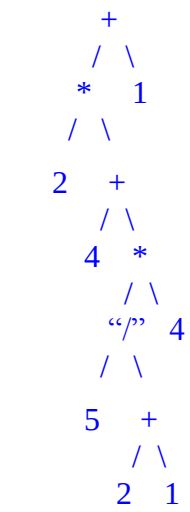
Es pot apreciar que en aquest cas el cost és lineal, O (n).

Exercici 3 [1 punt]

Quina estructura de dades utilitzaries per representar una operació aritmètica amb parèntesis, per exemple 2 * (4 +5 / (2 +1) * 4) +1. Raoneu-ho i feu un esquema de com quedaria l'expressió anterior. Quin seria el recorregut més adequat per avaluar el resultat de l'operació?

El podem representar com un arbre binari on les fulles serien els operands i els nodes intermedis els operadors. Per avaluar l'operació hauríem de fer un recorregut en postordre i efectuar l'operació cada vegada que ens trobéssim un operador.

Els parèntesis no cal guardar-los, però s'utilitzaran per a generar l'arbre.



El recorregut per avaluar el resultat de l'operació estaria en postordre.

Exercici 4 [1,5 punts]

Ordena les següents cotes de complexitat de menor a major, indicant també les que són iguals: $O(0.5 \cdot 2^n)$, $O(n+1)$, $O(2^{n/2})$, $O(\log_{10} n)$, $O((n+1)^2)$, $O(n \cdot \log\ n)$, $O(2\log_6 n)$, $O(n^n)$, $O(3n^2)$, $O(n!)$, $O(n(n+\log\ n))$, $O(1)$, $O(1+1/n)$, $O(n^{n-2})$. Justifica breument les relacions de l'ordre obtingut.

COMPLEXITAT		EQUIVALENT ASSIMPTÒTIC
a) $O(1)$	≡	$O(1)$
b) $O(1+1/n)$	≡	$O(1)$
c) $O(\log_{10} n)$	≡	$O(\log n)$
d) $O(2\log_6 n)$	≡	$O(\log n)$
e) $O(n+1)$	≡	$O(n)$
f) $O(n \cdot \log n)$	≡	$O(n \cdot \log n)$
g) $O((n+1)^2)$	≡	$O(n^2)$
h) $O(3n^2)$	≡	$O(n^2)$
i) $O(n(n+\log n))$	≡	$O(n^2)$
j) $O(2^{n/2})$	≡	$O(2^n)$
k) $O(0.5 \cdot 2^n)$	≡	$O(2^n)$
l) $O(n!)$	≡	$O(n!)$
m) $O(n^{n-2})$	≡	$O(n^n)$
n) $O(n^n)$	≡	$O(n^n)$

Per ordenar les complexitats proposades caldrà reduir-les als seus equivalents asimptòtics usant al notació O. Així doncs direm que a) i b) són equivalents i els de menor complexitat. c) i d) són també equivalents reduint els logaritmes de diferents bases a logaritmes naturals i representen el següent esglaó asimptòtic. A continuació i) representa una complexitat lineal. f) una complexitat més gran que e) però menor que g). Des de g) fins a i) són equivalents i representen una complexitat quadràtica. Les exponencials de base constant (*j* i *k*) suposen el següent esglaó asimptòtic, seguit de la complexitat amb càlcul factorial (*l*) i per últim les exponencials en base n (*m* i *n*).

Exercici 5 [2 punts]

Apartat 5.1) Ja coneixeu la representació de les cues mitjançant vectors i amb una estructura seqüencial, i també s'han estudiat una sèrie de millores a través de la utilització d'una gestió circular de les mateixes. Enumereu breument els avantatges i/o inconvenients que representaria una gestió circular en la implementació de les piles.

No representa cap avantatge ja que el primer element entrat en una pila no es desempila mai mentre quedin altres elements. És a dir, el primer element sempre es troba en la posició 0 i per tant les posicions relatives (ordre d'entrada) coincideixen sempre amb la absolutes (posició dins el vector).

Apartat 5.2) En les llistes ordenades implementades mitjançant vectors podem millorar l'eficiència de la consulta mitjançant una recerca dicotòmica. En la implementació encadenada no es pot millorar l'eficiència d'aquesta manera, és necessari realitzar una cerca començant pel primer element. Comentar la millora en l'eficiència que obtindríem si per fer la cerca disposéssim d'un punter a l'element que ocupa la posició mitjana de la llista.

Per millorar l'eficiència hauríem de tenir també els punters mitjans de totes les subllistes obtingudes dividint per la meitat la llista i subllistes obtingudes d'aquesta manera mentre es puguin dividir. Aquests punters mitjans haurien d'estar connectats entre ells de manera que a partir de qualsevol d'ells es pugui obtenir el punter corresponent a la posició mitjana de la subllista esquerra, així com el punter de la posició mitjana de la subllista dreta. És a dir hauríem de tenir els punters intermedis connectats en forma d'arbre.

Això suposa d'una banda un cost espacial doble i d'un altre un cost temporal important durant la inserció doncs una modificació en la llista suposaria una reorganització de l'arbre de punters intermedis penalitzant el cost a O (n) per cada inserció que fem. És a dir, un consti quadràtic per al conjunt d'entrades inserides.

La complexitat temporal i espacial és tan gran que no val la pena mantenir aquesta informació.

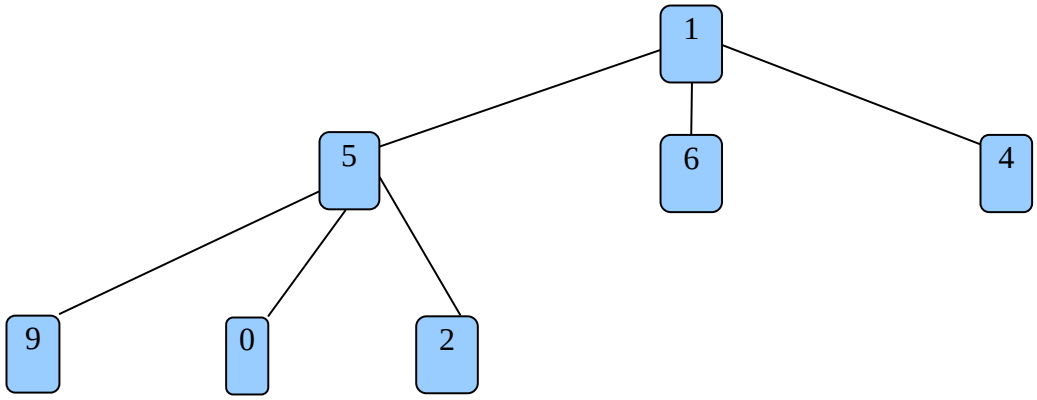
Apartat 5.3) Hem vist el funcionament de la implementació amb pilons dels arbres binaris. Creieu possible una implementació per piló d'arbres ternaris? Si ho creieu possible, de quina dimensió s'ha de crear el vector si volem emmagatzemar un arbre quasicomplert de X elements?. Proposeu un exemple.

La representació per piló d'arbres ternaris és possible usant el càlcul adequat que permet passar de pares a fills i de fills a pares. Així doncs, des de la posició de qualsevol node (N), excepte l'arrel, calcularem la posició del pare (P) fent P = (N +1) / 3. De la mateixa manera la posició dels fills es farà calculant

- N(iz) = 3P - 1
- N(cent) = 3P
- N(der) = 3P + 1

En tractar-se d'un arbre quasicomplert, es pot assegurar que no hi haurà posicions buides en el vector. Per tant per a albergar X elements hauríem dimensionar un vector de X posicions.

Vegem un exemple. Per representar l'arbre ternari i quasicomplert



necessitarem un vector com el que segueix:

1	5	6	4	9	0	2

Apartat 5.4) Ordenar el vector mitjançant l'algorisme de heap-sort utilitzant la versió que treballa amb un sol vector. Mostreu tots els estats intermedis del vector i la situació dels elements en cada pas:

4	1	7	2	6	3

Formació de la cua prioritària

4	1	7	2	6	3
4	1	7	2	6	3
4	6	7	2	1	3
7	6	4	2	1	3

Ordenació

7	6	4	2	1	3
3	6	4	2	1	7
6	3	4	2	1	7
1	3	4	2	6	7
4	3	1	2	6	7
2	3	1	4	6	7

3	2	1	4	6	7
1	2	3	4	6	7
2	1	3	4	6	7
1	2	3	4	6	7