Universitat Oberta de Catalunya-UOC
TFC- Xarxes de Computadors

ANALITZADOR GRÀFIC DE XARXA EN ENTORN GNU

Pilar ARBONÈS i ARBONÈS
Enginyeria Tècnica en Informàtica de Sistemes

Consultora: María Isabel MARCH i HERMO
Juny 2010

En memòria del meu pare Romualdo i de David Duaigües (bomber mort a l'incendi d'Horta de Sant Joan el 21 de juliol de 2009).

Estimats en vida i cada dia recordats.

El temps que he passat estudiant, esforçant-me, fent anar de bòlit i posant dels nervis a tots els que m'envolten ha estat important, però tot això i aquest projecte no hagués estat possible sense el recolzament, la paciència, l'ajuda i sobretot l'estimació que m'han demostrat sempre el meu marit Dídac i els meus dos fills, Dídac i Xavier.

Moltes gràcies.

Resum del Treball

Aquest treball es desenvolupa dins de l'àrea de Xarxes de Computadors, consisteix bàsicament en la realització d'una aplicació gràfica en entorn GNU, que permeti realitzar un anàlisi del trànsit d'una xarxa informàtica.

Això es coneix amb el nom de "*Sniffer*", la seva funció és "escoltar" el trànsit que circula per una determinada xarxa. Amb finalitats de seguretat i funcionalitat, pot ésser de gran utilitat en l'administració d'una xarxa; però també es pot utilitzar-la amb finalitats fraudulentes. Per això, és importat conèixer el funcionament d'aquestes aplicacions i poder-les utilitzar adequadament.

Dins el software lliure, els sniffers, s'han desenvolupat majoritàriament per a tasques educatives (es fan servir per a implementar atacs amb els que es puguin explorar els serveis de la xarxa), però la majoria d'usuaris els fan servis per a implementar atacs amb els que puguin explorar serveis de la xarxa. En el món empresarial, els programes d'aquest tipus estan dedicats als administradors de xarxa; les empreses que els executen ho fan sota l'entorn windows, i són de pagament.

En aquest treball es pretén desenvolupar una eina que mostri a l'usuari informació de cadascun dels protocols capturats per l'aplicació i orientada principalment a finalitats educatives.

Es podrà executar des de qualsevol màquina per a la que s'hagi implementat un compilador del llenguatge utilitzat, C++, i que tingui disponibles i instal·lades les llibreries 'pcap', que són les encarregades de proveir les dades de la xarxa i amb sistema operatiu basat en Linux.

ÍNDEX DE CONTINGUTS

ÍNDEX DE FIGURES

CAPÍTOL 1

INTRODUCCIÓ al TFC

Capítol 1. Introducció al TFC

1.1. Justificació del TFC i context en el qual es desenvolupa: punt de partida i aportació del TFC.

Aquest Treball Final de Carrera s'inclou dins de l'àrea de Xarxes de Computadors. Consisteix en la realització d'una aplicació gràfica en entorn GNU que faci un anàlisi del tràfic d'una xarxa informàtica; això es coneix amb el nom de *Sniffer*.

Per a poder portar a terme el treball, s'han hagut de tenir presents els coneixements adquirits al llarg de la carrera d'Enginyeria Tècnica de Sistemes Informàtics i sobretot a les assignatures de "Fonaments de Programació I i II", "Fonaments de Computadors I i II", "Xarxes de Computadors" i "Enginyeria del Programari".

La primera dificultat que sorgeix és determinar-concretar amb detall què ha de fer l'aplicació, quines característiques ha de tenir i com s'ha d'implementar. Una segona dificultat és el temps de que es disposa per la seva realització. La tercera i última dificultat és saber amb quines eines i recursos es compta; i sobretot si es tindran els suficients coneixements per a realitzar-lo.

La realització d'aquest TFC significa un repte a nivell personal, perquè requereix una gran capacitat i organització de treball que s'ha de compaginar amb aspectes familiars i professionals actuals. Tècnicament, suposa un esforç per l'aplicació correcta dels coneixements obtinguts en programació, xarxes, protocols…

Per iniciar el treball i com es podria dir "entrar en matèria", s'ha tingut en compte la informació de quines característiques i funcionalitats tenen altres productes existents en el mercat i així s'ha pogut determinar que l'aplicació és capaç de realitzar la captura de qualsevol classe de paquet, però només en fa l'anàlisi dels següents protocols:

- Trames Ethernet
- Paquets IP, ICMP i ARP
- Connexions TCP
- Datagrames UDP

Per a poder interactuar amb l'aplicació, s'ha creat una interfície de comandes molt intuïtiva, àgil i amigable per l'usuari, de forma que la seva utilització

requereix només els coneixements bàsics de xarxes i protocols. De totes maneres, el manual de l'usuari permet realitzar la consulta del funcionament.

El món de la informàtica es massa ampli i molt complex com per a ésser tractat en un treball que només pretén ser una mostra simple de les aplicacions que es poden aconseguir, i arribat aquest punt es planteja la qüestió de si es tenen suficients coneixements per a la seva realització o bé és necessita fer una ampliació de continguts, i en quins cal aprofundir.

Fent un anàlisi de l'amplitud que pot arribar a tenir el projecte es considera necessària l'ampliació de coneixements de programació en C++, aprofundir en la creació dels diagrames UML i l'aplicació dels coneixements adquirits en xarxes i protocols de comunicació.

En el primer nivell d'investigació s'han trobat gran varietat d'analitzadors de la xarxa que depenent de les necessitats dels usuaris poden tenir gran ventall d'aplicacions, en general positives; tot i que ens podem trobar amb usuaris (minoritaris) que en fan un ús fraudulent.

Tot seguit, en fem un petit resum:

1.1.1. Què és un *Sniffer*?

El terme *Sniffer* prové de l'anglès i significa "olfateador" (ensumador). En informàtica s'utilitza sobretot en l'apartat de l'anàlisi de xarxes per a capturar informació que es transmet per la xarxa.

Un Sniffer és un programa informàtic que registra la informació que envien els perifèrics, així com l'activitat realitzada en un ordinador determinat. Suposa un risc de la seguretat, no només per la màquina si no també per la xarxa

És molt habitual que donada la gran quantitat de paquets que viatgen per la xarxa en molt poc temps, els sniffers tinguin capacitats avançades de filtrat que permeten capturar el o els tipus de paquets desitjats. Alguns sniffers (gràfics) tenen la possibilitat de realitzar un anàlisi posterior de la informació que s'ha obtingut.

La utilització que es faci d'aquesta aplicació dependrà de les necessitats que es tinguin, però sobretot cal seguir un "codi ètic " per no fer-ne un mal ús.

Alguns usos que se li poden donar són:

- Captura automàtica de contrasenyes enviades en clar i noms d'usuari. Aquest és un ús malèfic que se li pot donar, perquè moltes vegades el "crackers" poden atacar el sistema després d'haver-ne obtingut la informació.
- Detecció d'intrusos, amb l'objectiu de descobrir "hackers". (Els programes específics que existeixen – IDS, Intrusion Detection System- , són Sniffers amb funcionalitats específiques).
- Mesura del trànsit, per a descobrir els punts de la xarxa que pateixen "colls d'ampolla".
- Anàlisi d'errades per a descobrir problemes de xarxa.
- Anàlisi d'aplicacions client-servidor que els hi permet obtenir informació real del que es transmet per la xarxa.
- Creació de registres de xarxa, de forma que els hackers no puguin detectar que estan essent rastrejats o investigats.
- Conversió del trànsit de la xarxa en format intel·ligible.

1.1.2. Anàlisi d'aplicacions

Els sniffers han estat i estan unes eines molt utilitzades en el món de les xarxes de computadors , per aquest motiu n'existeixen una gran varietat, tant comercials com gratuïts.

Per a xarxes amb connexió física (Ethernet/LAN) tenim: TCPdump, WinDump, Snort, Wireshark (abans Ethereal), Ettercap, WinSniffer, Hunt, Darkstat, traffic-vis, KSniffer;

Per xarxes sense fils tenim: Kismet o NetworkSumbler, encara que els de xarxes cablejades estan desenvolupant mòduls per suportar aquest tipus de xarxa.

Degut que el treball té com a punt d'interès els sniffers de la xarxa física, les funcionalitats que s'han tingut en compte han estat els del primer grup.

Per a copsar una mostra de la informació existent i degut que aquesta és molt àmplia, tot seguit es presenten les característiques d'alguns sniffers abans nomenats.

TCPDump (en Windows s'anomena **WinDump**)**:**

És una eina de diagnòstic per xarxes TCP/IP en temps real, monitoritza els paquets que entren i surten d'una interfície de xarxa i els presenta forma llegible per l'usuari.

Els filtres es poden crear per a mostrar només la informació que ens interessa, intrínsecament no és perillós i és de gran utilitat pels administradors de xarxa que necessiten veure el trànsit per a poder cercar problemes que pugui tenir la xarxa.

Permet utilitzar filtres mitjançant una expressió de cerca i solament mostrarà els paquets que tinguin la capçalera coincident amb aquesta. Permet també captura tot el trànsit que passa per una xarxa Ethernet basada en hubs o qualsevol altra xarxa que no disposi d'enrutaments intel·ligents com switch o router.

Malgrat és una eina molt útil, un dels inconvenients d'aquest sniffer és que no es considera un veritable IDS[1], funciona en línia de comandes, sense entorn gràfic ja que no analitza ni assenyala paquets anòmals, imprimeix tota la informació per pantalla o en un arxiu de registre per la seu posterior anàlisi.

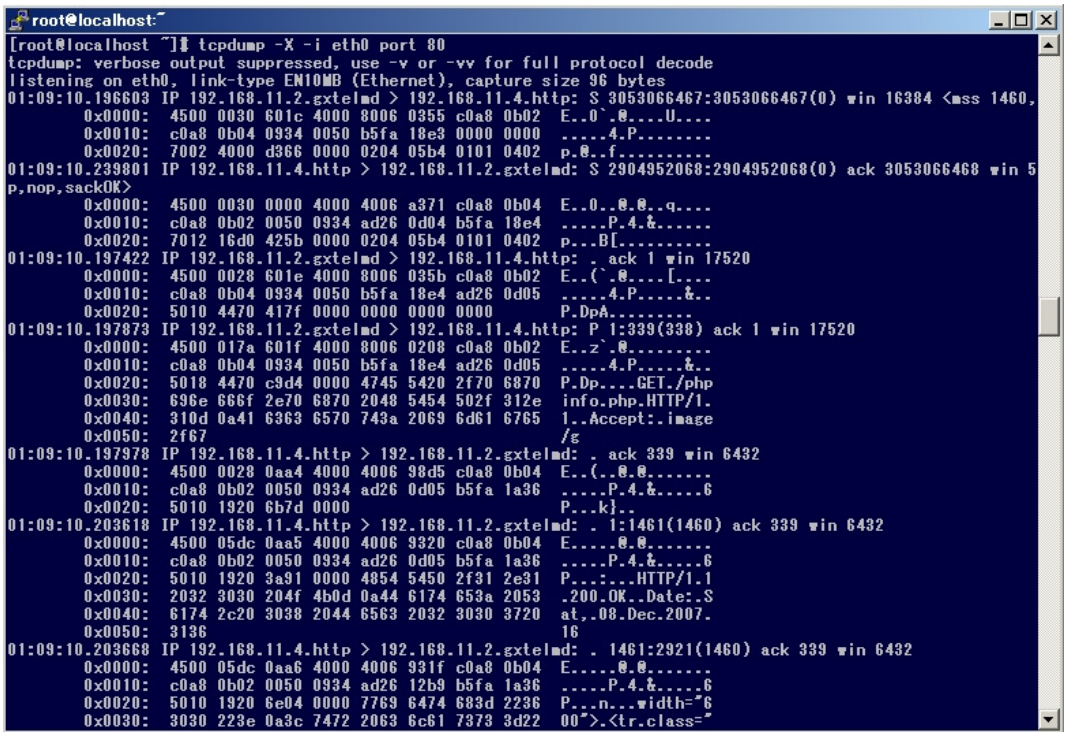


Figura1. Exemple de captura amb Tcpdump

[1] IDS : Sistema de detecció d'intrusos (*Intrusion Detection System)*

Es pot baixar des de l'adreça http://www.tcpdump.org i els tutorials des de l'adreça: http://www.arrakis.es/~terron/tcpdump.html

Snort:

És un IDS, sniffer de paquets i detector d'intrusions basat en xarxa (es monitoritza tot un domini de col·lisió).

És un software molt flexible, potent i senzill que ofereix capacitat d'emmagatzematge tant en arxius de text com en base de dades. Implementa un motor de detecció d'atacs i escàner de ports que permet enregistrat, alertar i respondre davant de qualsevol anomalia prèviament definida. També existeixen eines de tercers per a mostrar informes a temps real (ACID) o per a convertir-lo en un sistema per detectar i previndre intrusions.

Pot funcionar com *sniffer* (podem veure en consola i a temps real, tot el trànsit que passa per la xarxa), *registre de paquets* (permet guardar en un arxiu els logs per el seu posterior anàlisi) o *com un IDS normal* (en aquest cas NIDS).

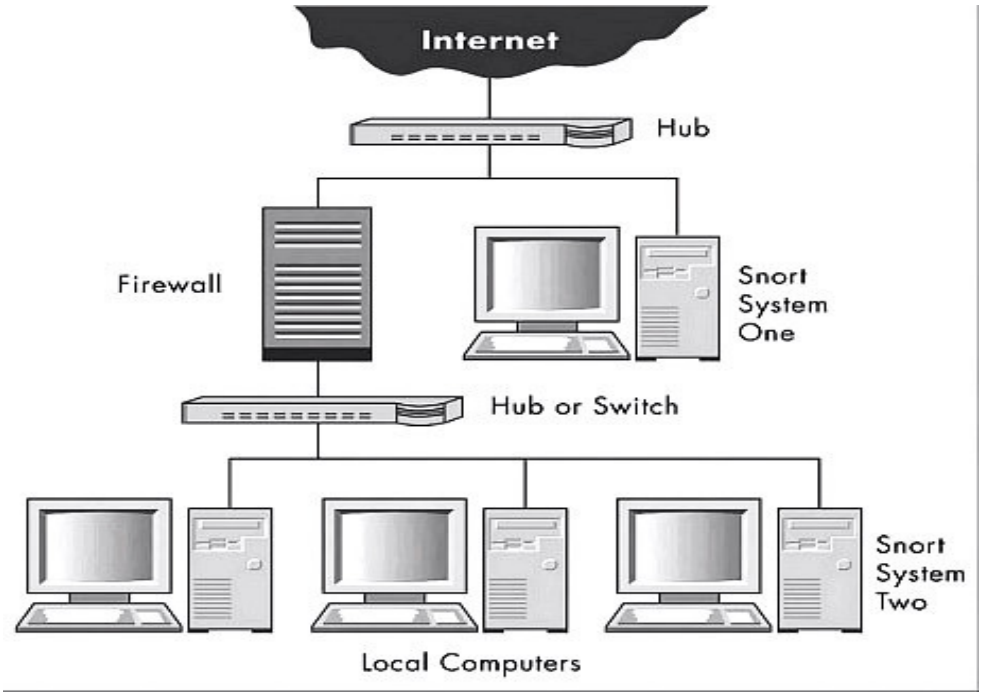


Figura 2 : Xarxa amb Snort

Quan un paquet coincideix amb alguna regla establerta en la configuració, es logea; així se sap "ON" i "QUAN" s'ha produït l'atac. Snort utilitza la llibreria estàndard "libcap" i "tcpdump" . Està disponible sota la llicència de GPL, gratuït i funciona tant en plataforma Windows com UNIX/Linux.

Disposa d'una gran quantitat de filtres predefinits i actualitzacions constants, mitjançant els diferents butlletins de seguretat, dels casos d'atacs o vulnerabilitats que s'hagin detectat.

Els usuaris poden crear "signatures" basades en les característiques dels nous atacs de xarxa i enviar-les a les llista de correu se signatures de Snort, d'aquesta forma tots els altres usuaris se'n poden beneficiar.

Wireshark (abans conegut com **Ethereal**):

És un analitzador de protocols utilitzat per realitzar anàlisi i solucionar problemes en xarxes de comunicacions pel desenvolupament de software i protocols, també com una eina didàctica per ensenyament.

La seva funcionalitat és similar a la de tcpdump, però afegeix una interfície gràfica i moltes opcions (més de 300 protocols) d'organització i filtrat d'informació. Permet veure tot el trànsit que passa per la xarxa establert com a mode promiscuo. Inclou una versió basada en text anomenada "tshark". Ens permet fer la captura de dades directament de la xarxa o bé obtenir informació a partir d'una captura d'un disc (pot llegir més de 20 tipus de formats diferents).

Pot capturar diàlegs, llistar trames Ethernet, visualitzar les capçaleres de cada trama i accedir a les dades adjuntes. Les seves eines ens permeten fixar trames per direcció física, per protocol…

És de software lliure i s'executa tant en sistemes UNIX/Linux y compatibles (Solaris, OpenBSD, MAC) com en Windows.

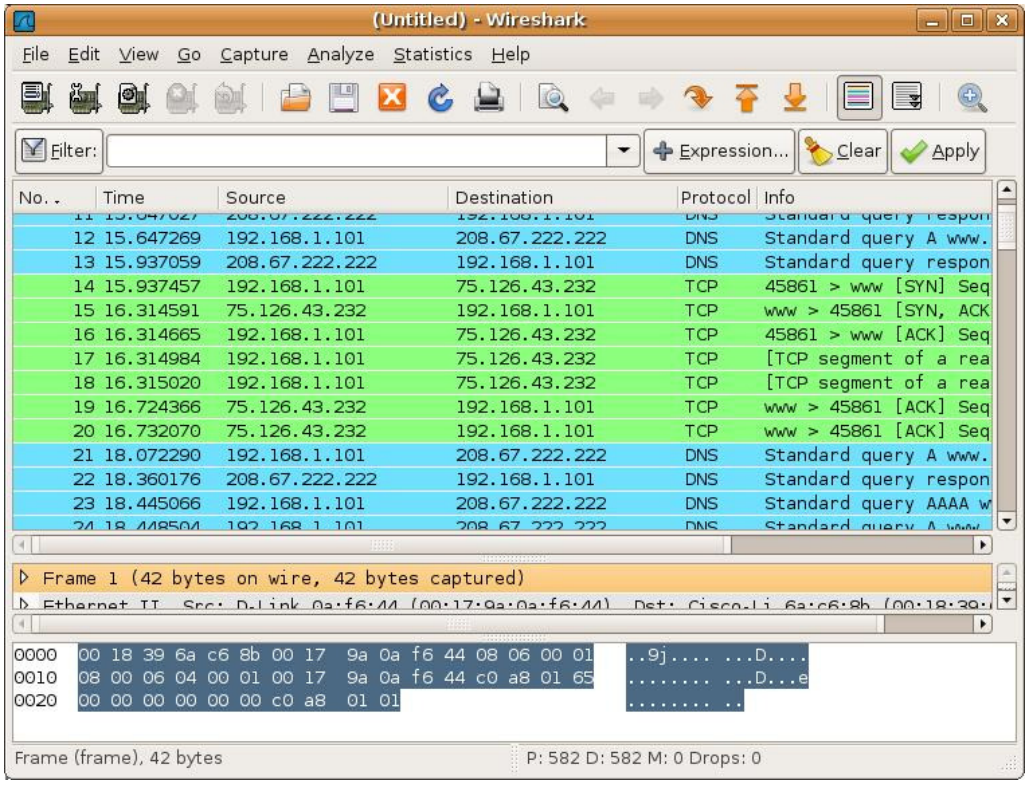


Figura 3. Exemple de captura amb Wireshark

Kismet:

És un sniffer per a xarxes sense fils 802.11, es diferencia de la majoria dels altres sniffers d'aquest tipus en el seu funcionament passiu, realitza la seva funció sense enviar cap paquet detectable, permetent detectar la presència de varis punts d'accés i clients sense fils, associant uns amb altres. També inclou característiques bàsiques de sistemes de detecció d'intrusos, com programes de rastreig incloent NetStumbler, així com certs atacs de xarxa, tot sense fils. Kismet té tres parts diferenciades:

Una "sonda" que pot utilitzar-se per a recollir paquets, que son enviats a un *servidor* per la seva interpretació.

Un "servidor" que pot ser utilitzat junt amb la sonda, o bé ell per si sol, interpretant les dades dels paquets, extrapolant la informació sense fils i organitzant-la.

I per últim, el "client" que es comunica amb el servidor i mostra la informació que el servidor recull.

Funciona amb qualsevol tarja sense fils que suporti monitorització "raw" i pot rastrejar trànsit 802.11b. 802.11a i 802.11g.

El programa pot aplicar-se sota Linux, FreeBSD, NetBSD, OpenBSD i Mac OS X. També pot funcionar amb Windows, malgrat la única Font entrant de paquets compatible és una altra sonda.

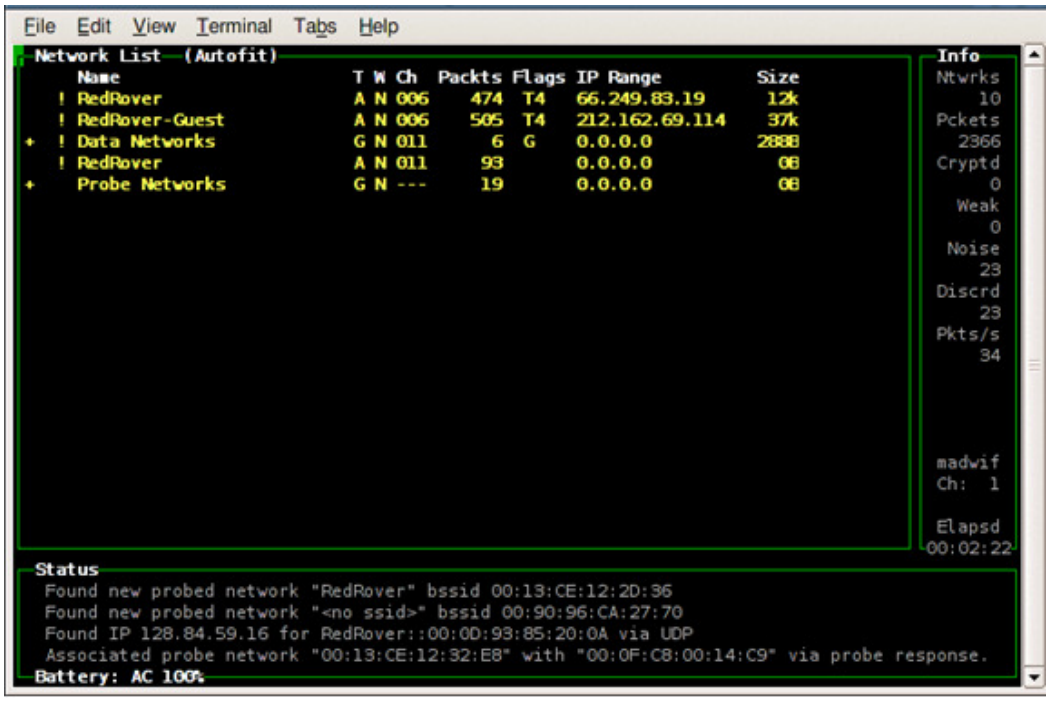


Figura 4. Exemple de captura amb Kismet

Ettercap:

És un sniffer/interceptor/registrador per LANs amb switch. Suporta adreces actives i passives de varies protocols (incloent els xifrats, com SSH i HTTPS),. També fa possible la injecció de dades en una connexió establerta i filtrat al vol mantenint la connexió sincronitzada gràcies al seu poder per establir un atac “man-in-the-middle”.

Per a més informació: http://ettercap.sourceforge.net/

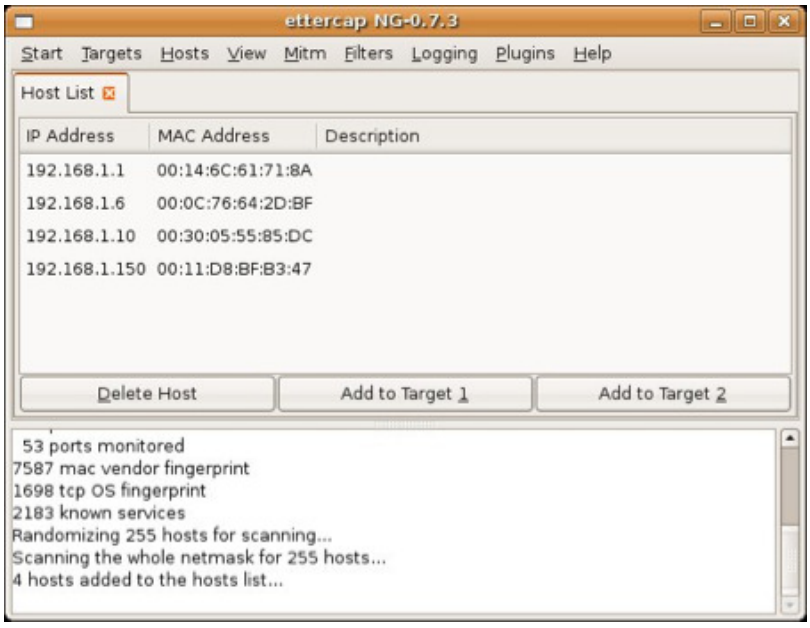


Figura 5. Exemple de captura amb Ettercap

1.2. Objectius del TFC

L'objectiu final del TFC és la creació d'una eina per a la captura i posterior anàlisi del trànsit d'una xarxa determinada, en entorn GNU; tot aplicant els coneixements adquirits al llarg dels estudis d'Enginyeria Tècnica Informàtica de Sistemes dins les diferents apartats de disseny, programació i implementació. Per aconseguir l'objectiu es proposa dotar a l'aplicació de les següents funcionalitats:

- Facilitat en la utilització per part de l'usuari i visualment senzilla.
- Poca càrrega en el consum de recursos per part de l'ordinador.
- Eficiència en la captura dels paquets.
- Realització de captures dels paquets a temps real
- Dotar a l'aplicació d'opcions de filtrat: per adreces IP, MAC's, ports i protocols.
- Poder emmagatzemar les dades d'una sessió en un fitxer.
- Mostrar dades estadístiques de les captures.

1.3. Enfocament i mètode seguit

1.3.1. Llenguatge de programació

Pel que fa referència al llenguatge de programació, com que es vol treballar amb un **llenguatge orientat a objectes**, aquest ha de ser en JAVA o bé C++.

L'elecció ha estat fer-ho en C++ principalment perquè aquest ha estat i és el llenguatge més utilitzat en la majoria de les empreses per fer programari nou, i en el que està escrit una gran part del programari lliure a Internet a més de la seva gran potència i suport.

1.3.2. Metodologia del desenvolupament

Per al desenvolupament del projecte, basat en la programació orientada a objectes, es farà servir el mètode UML (Unified Modeling Language). Aquest model ha esta proposat com estàndard ISO per l'OMG (Object Management Group); és un llenguatge gràfic i flexible de modelat que permet descriure models que representen sistemes basant-se amb els conceptes de la programació d'Orientació a Objectes.
En la producció de l'aplicació s'ha seguit un cicle de vida iteratiu i incremental de manera que s'anava consolidant a mesura que s'aconseguien els objectius marcats.

En el desenvolupament del projecte s'han seguit els següents apartats:

1- Cerca, investigació i valoració dels productes semblants existents en el mercat.
Dins la xarxa de comunicació d'Internet, s'ha fet una cerca d'informació dels sniffers; que són, quins són el més coneguts, els més utilitzats; quines característiques tenien cadascun d'ells, i després, com a mostra, s'ha realitzat una tria i s'ha fet un resum.

2- Anàlisi i disseny: Arquitectura general del programa, planificació del projecte, casos d'ús.
S'estudia el domini del problema, s'estableix l'arquitectura general del programa i es realitza una primera planificació de tot el projecte.

3- Implementació: Desenvolupament de les fase anterior.
Es realitza la implementació de forma iterativa i incremental, es tenen en compte les necessitats que ha de complir la informació

obtinguda i es desenvolupa l'arquitectura de l'aplicació. S'han adquirit i repassat coneixements propis del llenguatge de programació en C++ per poder-los utilitzar en el projecte.

4- Fase de proves, retocs de l'estructura establerta i creació de la memòria.
Aquesta fase, igual que amb la fase 3 s'aplica la forma iterativa i incremental per tal d'anar consolidant el projecte.

1.4. Planificació del projecte

El projecte està pensat per a ésser desenvolupat al llarg d'un quadrimestre (primavera del curs 2009-2010) i amb una càrrega lectiva de 7,5 crèdits que implica un treball (com a mínim) de 6 hores setmanals.

La planificació del projecte és la següent:

Pas 1:

Temporització: 2 setmanes (de l'1 al 14 de març)

Descripció: Recollida i classificació de tota la informació que pugui ser d'utilitat per la realització del projecte:
- Informació sobre altres analitzadors de xarxa existents
- Informació sobre llenguatges C++ , Java i Visual C
- Informació sobre protocols TCP/IP
- Informació de disseny, anàlisi i implementació

Objectius:

- Tenir una visió general de altres analitzadors
- Decidir el llenguatge més adient i còmode per l'autora.
- Aprofundir el coneixement sobre els protocols TCP/IP
- Aprofundir el coneixement d'anàlisi, disseny i implementació dels sniffers.

Fites:
- Document on es reflecteixi el llenguatge de programació que s'utilitzarà
- Document resum dels altres analitzadors existents.

Pas 2:

Temporització: 4 setmanes (del 15 de març al 11 d'abril)

Descripció: Anàlisi i disseny de l'aplicació a implementar

Objectius:

- Establir les opcions i requisits que ha de tenir l'aplicació
- Determinar el llenguatge de programació
- Obtenir l'anàlisi del programa segons les opcions que se li vulguin donar
- Obtenir el disseny de l'aplicació segons l'anàlisi feta anteriorment.

Fites:

- Obtenir l'anàlisi i disseny de l'aplicació
- Documentació de l'anàlisi i disseny de l'aplicació.

Pas 3:

Temporització: 6 setmanes (del 12 d'abril al 23 de maig)

Descripció: Implementació de l'aplicació

Objectius:

- Implementació de l'aplicació.
- Creació de l'aplicació. Codi

Fites:

- Documentació sobre la implementació.

Pas 4:

Temporització: 1 setmana (del 24 al 30 de maig)

Descripció: Documentació del producte obtingut

Objectius:

- Determinar l'ús de les opcions de configuració de l'aplicació
- Documentar la configuració de l'aplicació

Fites:

- Document on es reflecteixi les opcions de configuració

Pas 5:

Temporització: 2 setmanes (del 31 de maig al 13 de juny)

Descripció: Revisió final de la memòria i preparar la presentació de l'aplicació creada

Objectius:

- Síntesi de la memòria realitzada durant el projecte
- Realització de la presentació de l'aplicació.

Fites:

- Memòria del projecte
- Presentació del projecte

Desviació temporal del pla de treball inicial

El pla de treball ha estat modificat respecte com va estar planificat en un principi. Per circumstàncies familiars, problemes de hardware i haver de realitzar un petit reciclatge dels coneixements de programació, van fer que la planificació portes unes tres setmanes retard que calia recuperar. Els punts més afectats han estat els que corresponen als apartats de *l'Enfocament i mètodes a seguir*, *Llenguatge de programació, Metodologia del desenvolupament i Planificació del projecte*. També s'ha vist afectat el punt de la interfície gràfica. En aquets apartat tenia previst realitzar una interfície visualment molt atractiva però a mesura que realitzava codi i que el temps m'apremiava, he optat per obtenir més eficiència de l'aplicació en detriment de la visualització; fixant-se com a exemple amb l'Snort que és una referència en aquest camp i que es fa servir amb línia de comandes. A mesura que s'anaven esvaint les dificultats, s'anava augmentant el rendiment per l'execució del projecte que tenim en reflectit en aquest document.

1.5. Productes obtinguts

Una vegada acabat el projecte, s'haurà obtingut:

- Analitzador de xarxa en entorn GNU, amb les característiques abans anomenades(apartat 1.2. del present document).
- Memòria del TFC (el present document).
- Manual d'usuari, instal·lació i requeriments (capítol 6).
- Presentació amb diapositives per explicar tot el treball desenvolupat (com a material complementari).

1.6. Breu descripció dels següents capítols

Els següents capítols estan dedicats a cadascuna de les etapes de creació de l'aplicació i corresponen a:

- Anàlisi del projecte de TFC (capítol 2): *Definir que és el que el programari necessita, els requisits, a un llenguatge més formal, fàcil i entenedor.*
- Disseny i Implementació del projecte (capítol 3): *Elaboració de la proposta de treball segons unes pautes i processos sistemàtics. Execució i posada en marxa del projecte. Realització de codi programador.*
- Joc de proves (capítol 4):
- Manual d'usuari i d'instal·lació (capítol 5): *Guia per la instal·lació de l'aplicació i indicacions bàsiques per la seva utilització.*

CAPÍTOL 2

ANÀLISI del TFC

2.1. Introducció

L'anàlisi és la primera part del programari i potser la més important, ja que una bona anàlisi influencia un bon desenvolupament d'aquest.

L'anàlisi consisteix a traduir el que el programari necessita, els requisits, a un llenguatge més formal, fàcil i entenedor, per aquest treball utilitzarem el model i diagrames UML (per a realitzar els gràfics es farà servir l'eina MagicDraw).

Per obtenir un bon producte, és important que posem cura en tres aspectes d'aquest apartat.

- Primer una bona anàlisi per identificar els requisits
- Segon, identificació de les classes fonamentals que intervindran i que seran la base per la implementació del programari.
- Tercer, identificació dels diferents casos d'ús en termes de les classes que han estat identificades en el segon pas i que intervindran en el procés.

2.2. Model de domini

De la recollida i documentació dels requisits, s'ha extret el model de domini que queda de la següent forma:

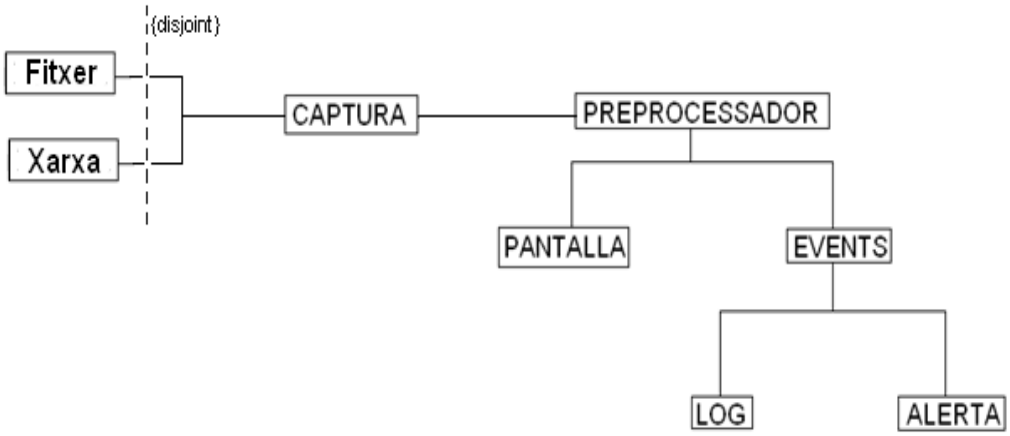


Figura 6. Model de domini

I en l'aplicació es tracten els paquets que es detecten seguint aquest arbre de protocols (Figura 7)

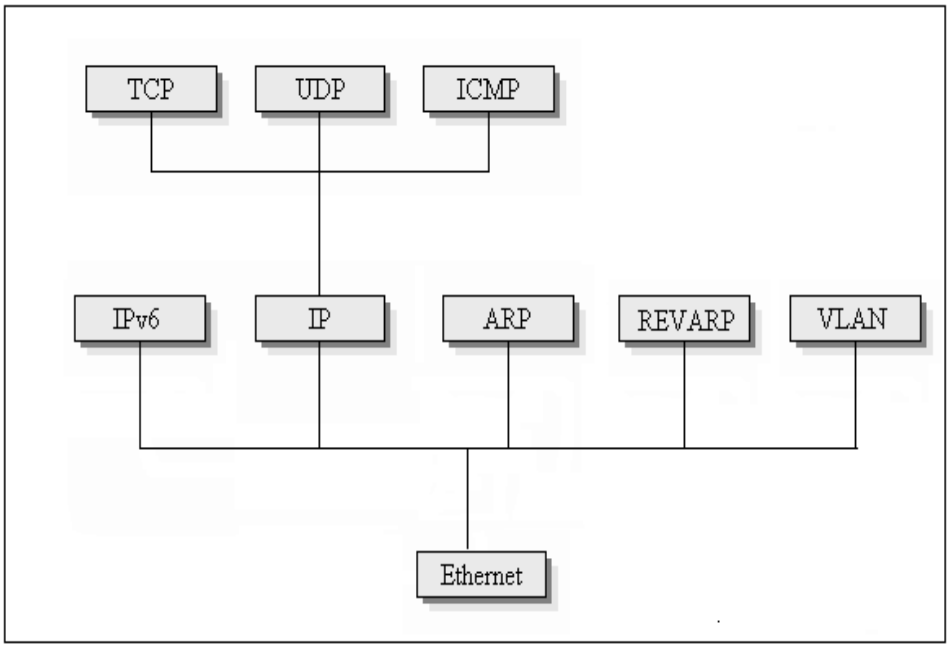


Figura 7. Relació de protocols per nivells

2.3. Casos d'ús: Diagrama i definició de cada cas

Els diagrames de casos d'ús (en anglès, *use case*) serveixen per a mostrar les funcions d'un sistema de programari des del punt de vista de les seves interaccions amb l'exterior i sense entrar ni en la descripció detallada ni en la implementació d'aquestes funcions.

Els casos d'ús es fan servir tant a nivell de recollida i documentació de requisits com d'anàlisi.

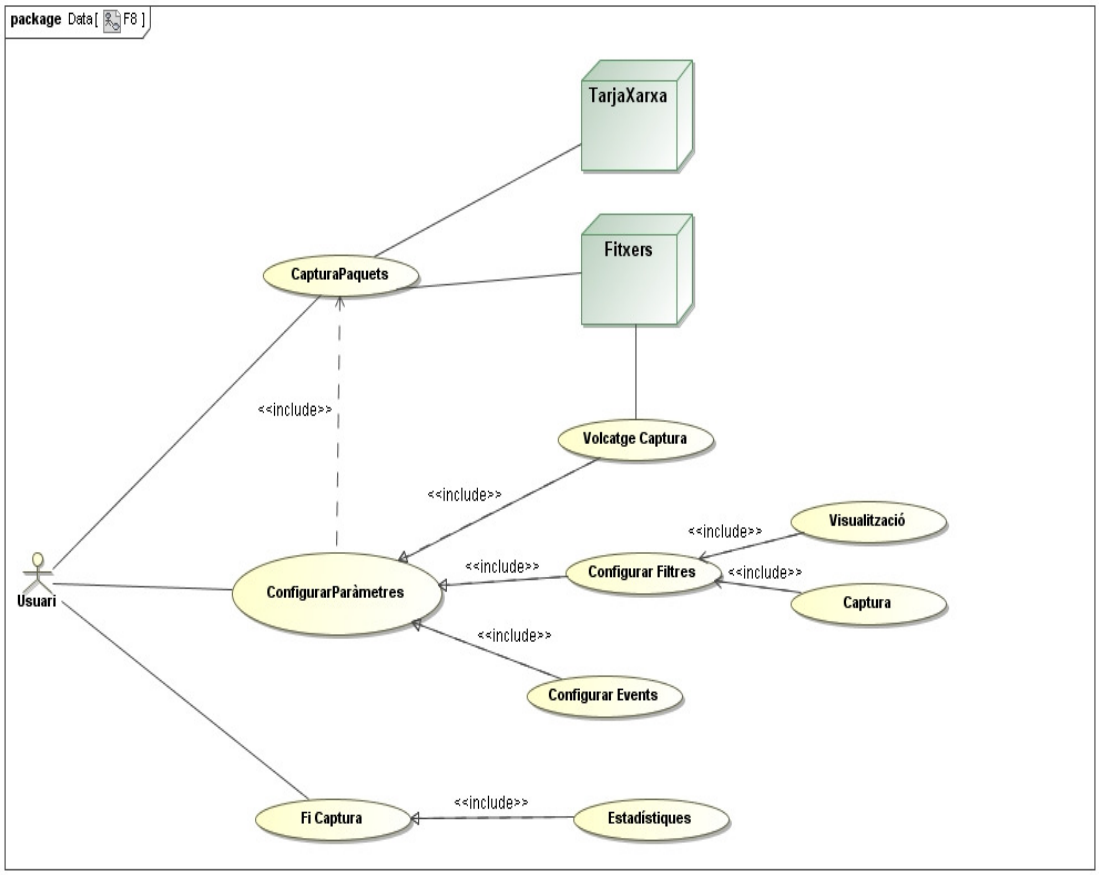


Figura 8. Diagrama de casos d'ús de l'aplicació.

La figura 8 mostra l'esquema general de l'aplicació utilitzant el diagrama de cas d'ús. El diagrama ens mostra com interacciona l'actor USUARI amb el sistema. L'usuari és qui demana al sistema realitzar la "captura" de paquets", qui configurarà els paràmetres d'aquesta captura i qui posarà fi a la captura. Tot això relacionat amb la visualització i la creació d'events que està vinculat amb els casos d'ús abans dits. Per últim i per a poder realitzar l'aplicació és necessari que el sistema físic – hardware- tingui la tarja de xarxa i la de l''emmagatzematge en fitxers (al ser elements aliens a l'usuari), s'han representat gràficament de forma diferent.

Tot seguit es desenvolupen cadascun dels casos d'ús establerts per l'aplicació.

Cas d'ús: CapturaPaquets

Resum de la funcionalitat: S'inicia el procés de captura de paquets.

Actors: Usuari

Casos d'ús relacionats: ----

Precondició: Existeix el fitxer que guarda les dades si s'escau

Postcondició: La pantalla mostra les dades de la captura

Procés normal principal:

1. L'usuari inicia la sessió de captura.
2. Les dades es mostren per pantalla.

Alternatives de procés i excepcions:

1.1. Problema en executar-se en algun punt del codi per falta de privilegis o permisos.

1.2. El procés finalitza.

Interconnectivitat- Relació amb Tarja de Xarxa i Fitxers

Cas d'ús: ConfigurarParàmetres

Resum de la funcionalitat: L'usuari selecciona els paràmetres de captura de paquets i el sistema configura la seva manera de treballar i els recursos que necessiti per satisfer les necessitats de l'usuari amb aquests requeriments.

Actors: Usuari

Casos d'ús relacionats: CapturaPaquets,BolcatCaptura, ConfigurarFiltres, ConfigurarEvents

Precondició: L'usuari ha seleccionat d'interfície amb la que vol realitzar la captura.

Postcondició: El sistema arrenca la captura amb tots els paràmetres introduïts.

Procés normal principal:

1. L'usuari selecciona des d'on vol realitzar la captura.
2. L'usuari selecciona opcions de filtrat.
3. L'usuari escull guardar o no les dades i a on.
4. Els paquets obtinguts de la captura es mostren per pantalla

Alternatives de procés i excepcions:

1.1. La interfície seleccionada no és vàlida per captura o no s'hi té accés.

2.1. Els filtres introduïts no són vàlids o contenen errors.

3.1. No es poden guardar les dades, el fitxer no es pot crear o sobreescriure.

4.1. Problema en la captura de paquets. La informació dels paquets no es mostra correctament.

* En tots els casos el programa acaba la seva execució.

Cas d'ús: FiCaptura

Resum de la funcionalitat: Permet que la captura de paquets es pugui finalitzar en qualsevol moment que l'usuari vulgui.

Actors: Usuari

Casos d'ús relacionats: Estadístiques

Precondició: S'està realitzant la captura de paquets de xarxa.

Postcondició: El procés s'atura.

Procés normal principal:

1. L'usuari prem una combinació de tecles (normalment Ctrl+C).
2. El sistema posa fi a la captura.

Alternatives de procés i excepcions:

2.1. El sistema no respon i continua l'execució

Cas d'ús: BolcatCaptura

Resum de la funcionalitat: Es guarda en un fitxer la captura de dades.

Actors: Usuari.

Casos d'ús relacionats: CapturaPaquets, ConfigurarParàmetres

Precondició: L'usuari inicia una captura especificant una ruta de fitxer a on bolcar les dades.

Postcondició: S'ha creat un fitxer amb la captura.

Procés normal principal:

1. L'usuari introdueix les dades d'una sessió de captura i la ruta del fitxer a on bolcar les dades.
2. L'usuari finalitza la captura quan ho desitgi.
3. El fitxer ha sigut creat en el sistema de fitxers.

Alternatives de procés i excepcions:

1.1. El nom del fitxer conté caràcters erronis o la ruta no és vàlida.

1.2. No es troba la ruta per la creació del fitxer o no es tenen permisos.

1.3. El fitxer ja existeix i l'usuari no ha especificat que es sobreescrigui.

* El programa acaba en tots els casos.

Cas d'ús: ConfigurarFiltres

Resum de la funcionalitat: L'usuari pot triar el tipus de filtrat que li és més útil tant a nivell de captura com a nivell de visualització.

Actors: Usuari

Casos d'ús relacionats: CapturaPaquets, ConfigurarParàmetres,

Precondició: L'usuari inicia una captura especificant fitxers contenint filtres.

Postcondició: L'aplicació realitza el filtrat i ens va mostrant el resultat de la captura.

Procés normal principal:

1. L'usuari introdueix les dades d'una sessió de captura i el tipus de filtre que vol aplicar amb la ruta del fitxer que el conté.
2. L'aplicació engega una captura.
3. L'aplicació el mostra per pantalla els resultats que compleixen els filtres.

Alternatives de procés i excepcions:

1.1. L'usuari no dóna correctament les ordres de filtratge

1.2. L'usuari no escriu correctament la ruta dels fitxers dels filtres.

1.3. Els fitxers contenen filtres que estan mal formats.

* El sistema mostra per pantalla les opcions que té l'usuari i surt.

Cas d'ús: Visualització

Resum de la funcionalitat: Es mostra per pantalla la captura de dades

Actors: Usuari

Casos d'ús relacionats: CapturaPaquets, ConfigurarParàmetres, ConfigurarFiltres

Precondició: La captura s'ha iniciat sense errors.

Postcondició: La pantalla ens va mostrant els resultats de la captura.

Procés normal principal:

1. L'usuari inicia una captura.
2. Es va veient per pantalla una llista dels paquets capturats, detallada per camps.

Cas d'ús: Captura

Resum de la funcionalitat: El sistema realitza la captura de paquets

Actors: -

Casos d'ús relacionats: CapturaPaquets, ConfigurarParàmetres,

Precondició: L'usuari ha indicat els paràmetres de la captura i aquestos no contenen errors.

Postcondició: L'aplicació realitza els filtrats els resultats es mostren per pantalla.

Procés normal principal:

1. L'usuari ha donat tots els paràmetres per una captura.
2. L'aplicació captura els paquets segons els paràmetres demanats.

Alternatives de procés i excepcions:

2.1. L'aplicació detecta un error des de el sistema operatiu que no el permet continuar.
2.2. L'aplicació surt.

Cas d'ús: ConfigurarEvents

Resum de la funcionalitat: Es mostra la possibilitat d'emmagatzemar els paquets capturats i realitzar alertes o logs.

Actors: Usuari

Casos d'ús relacionats: ConfigurarParàmetres

Precondició: L'usuari ha iniciat una captura especificant un fitxer de configuració.

Postcondició: Es creen les alertes o logs i es graben en un fitxer si s'escau.

Procés normal principal:

1. L'usuari inicia una captura.
2. Es va veient per pantalla una llista dels paquets capturats, detallada per camps.
3. Si es produeix una concordança positiva entre el paquet capturat i la informació del fitxer de configuració el sistema actua.
   3.1. Si està configurat com a alarma es mostra per pantalla i es grava al fitxer.
   3.2. Si és un log només es grava al fitxer.
4. L'aplicació continua fins que l'usuari la para.

Alternatives de procés i excepcions:

1.1. L'usuari no dóna correctament les ordres de filtratge

1.2. L'usuari no escriu correctament la ruta dels fitxers dels filtres.

1.3. Els fitxers contenen filtres que estan mal formats.

* Surt de l'aplicació.

2.4. Diagrama de seqüència

El diagrama de seqüència mostra la interacció d'un conjunt d'objectes en una aplicació a través del temps i es modela per a cada mètode de la classe. El diagrama de seqüència conté detalls de implementació de l'escenari, incloent objectes i classes que s'utilitzen per a implementar

aquest escenari, i missatges que intercanvien dos objectes entre ells. Mostra els objectes que intervenen amb línies discontinues verticals (que simbolitzen la vida dels objectes, la durada en el temps d'execució) i els missatges entre objectes, amb fletxes horitzontals.

Existeixen dos tipus de missatges: *síncrons i asíncrons.*

Els missatges síncrons es corresponen a les crides als mètodes de l'objecte que rep el missatge. L'objecte que envia el missatge queda bloquejat fins que acaba la crida. Aquest tipus de missatge es representa amb fletxes que tenen el cap ple. Els missatges asíncrons acaben immediatament, i creen un nou fil d'execució dins la seqüència. Es representen amb el cap obert. També es representen les respostes amb un missatge de fletxa discontínua.

Estructura del diagrama: Els missatges es dibuixen cronològicament des de la part superior fins la inferior del diagrama; la distribució horitzontal dels objectes, és arbitrària (no necessàriament han d'estar en aquest ordre). Durant l'anàlisi inicial, el modelador, col·loca el nom d'un missatge A en la línea del missatge. Més endavant, durant el disseny, el nom A es canvia pel nom del mètode que està essent cridat o invocat i que pertany a la definició de la classe instanciada per l'objecte en la recepció final del missatge.

El gràfic que es mostra en la Figura 9, s'han representat quatre objectes Pantalla, Preprocessador, Captura i Events; que es comuniquen de la següent forma:

Una vegada l'usuari ha iniciat el procés, es processen els paràmetres inicials i els filtres de visualització. Després s'inicialitza un Preprocessador per cada interfície on es vol realitzar la captura. El preprocessador inicia la Captura i els Events. Les Captures comencen a enviar dades als Preprocessadors que les tracten mitjançant els filtres introduïts i generen alertes o logs si es necessari. Es mostra la captura per pantalla realitzant el filtratge de visualització fins que l'usuari decideix (s'inicia el bucle de captura-loop), mentre no es premi CTRL+C. Quan l'usuari prem CTRL+C, el "loop" de la Captura es para i s'executa el "break", mostrant les estadístiques dels paquets capturats i filtrats i eliminant els Preprocessadors i Captures.

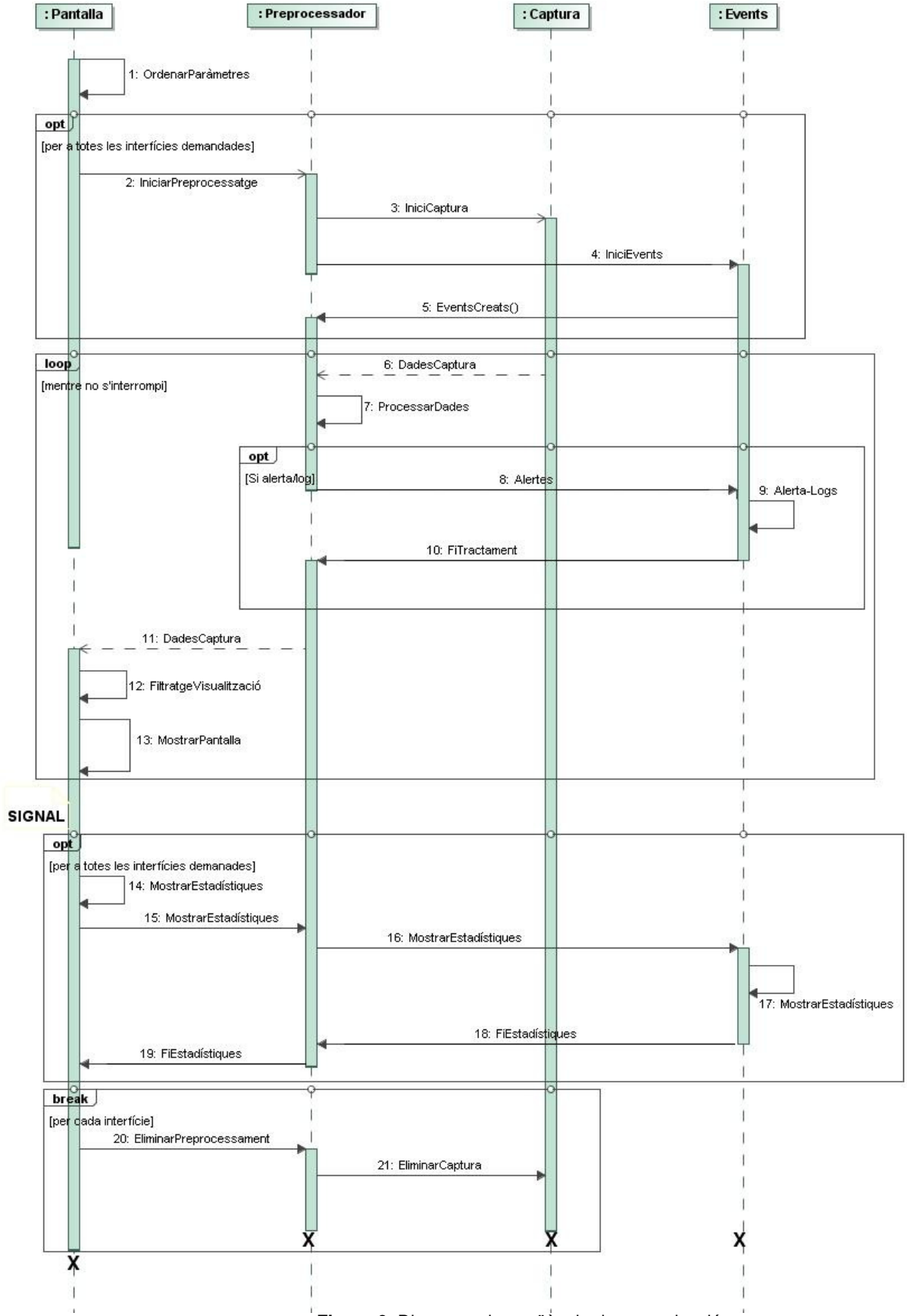


Figura 9. Diagrama de seqüència de comunicació

2.5. Diagrama de col·laboració

El diagrama de col·laboració en les versions de UML és essencialment diagrama estàtic que mostra interaccions organitzades al voltant dels rols establerts anteriorment, damunt del qual es representen els missatges de la interacció.

No ens mostra el temps com una dimensió a part, per la qual cosa és necessari etiquetar amb nombres la seqüència dels missatges i els fils concurrents.

Es poden representar tres tipus de missatges:

a) *Missatges simples:*corresponen a la simple progressió dins un fil d'execució. Es representen amb una fletxa amb la punta oberta, amb la qual s'indica el sentit del missatge.

b) *Missatges síncrons:* només es donen quan el client envia un missatge al subministrador i aquest accepta el missatge; la classe emissora executa el codi fins que envia el missatge i, després, s'espera a rebre el resultat de l'operació associada al missatge. Es representen amb una fletxa amb la punta plena que n'indica el sentit.

c) *Missatges asíncrons:* la comunicació asíncrona es produeix quan la classe emissora envia un missatge al subministrador i es continua executant sense esperar que arribi el resultat; la classe receptora, per la seva banda, no executa l'operació immediatament, sinó que desa la petició en una cua. Es representa amb una fletxa amb la punta oberta i tallada horitzontalment.

La Figura 10 mostra els rols establerts (igual que el diagrama de seqüència), la numeració correspon als missatges que cadascun dels rols efectua i en quin sentit, així tenim que el rol "Events" (etiquetat com rol número 4) efectua 2 comunicacions: 4.1 i 4.2 ; el 4.1. procés amb ell mateix i 4.2 comunicació cap a "Preprocessador" indicant el Fi del tractament.

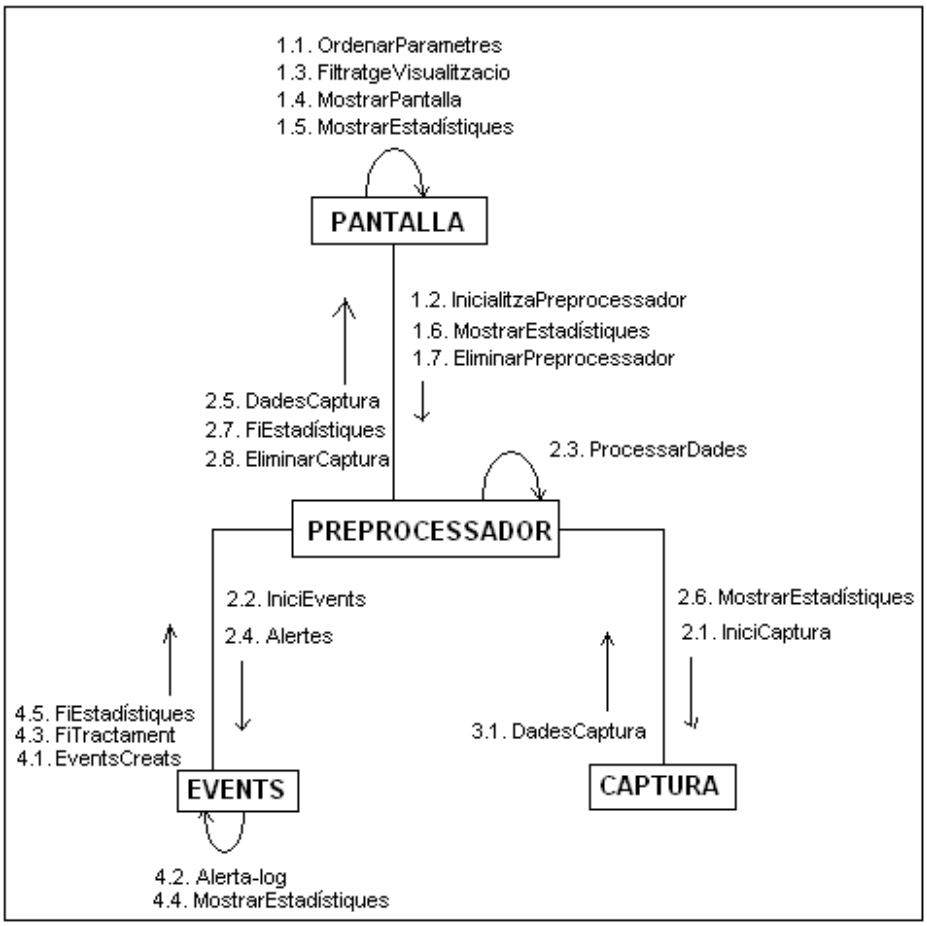


Figura 10. Diagrama de col·laboració

2.6. Diagrama estàtic de disseny. Identificació de les classes d'entitats

La identificació de les classes d'entitats consisteix a establir les classes que intervindran en la nostra aplicació, i definir els atributs i operacions que les definiran.

Partint dels casos d'ús definits en l'apartat 2.3, s'han trobat les següents classes d'entitats:

a) Diagrama estàtic de disseny: Esquema general per establir les relacions existents entre les classes que intervenen.

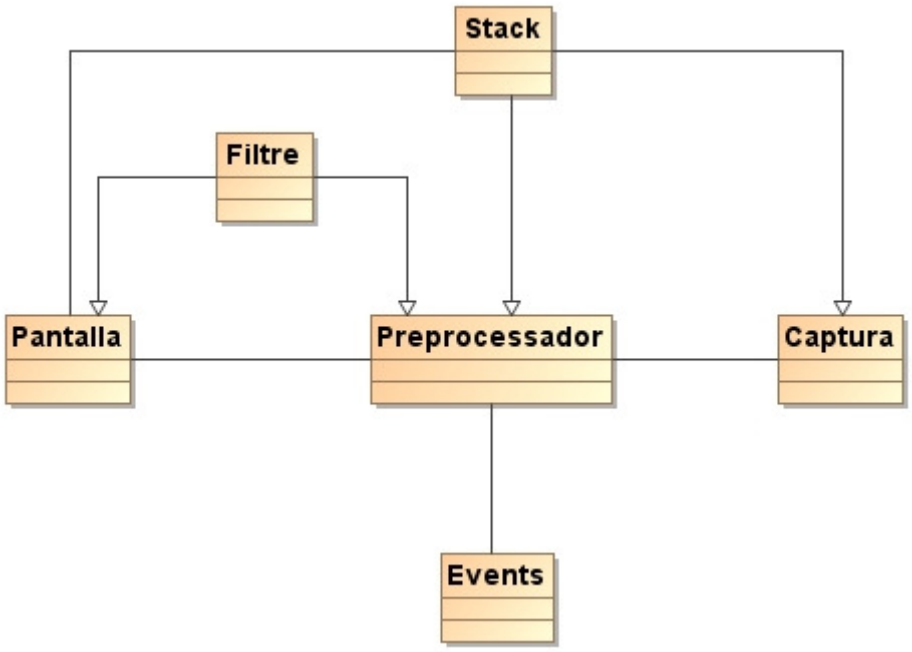
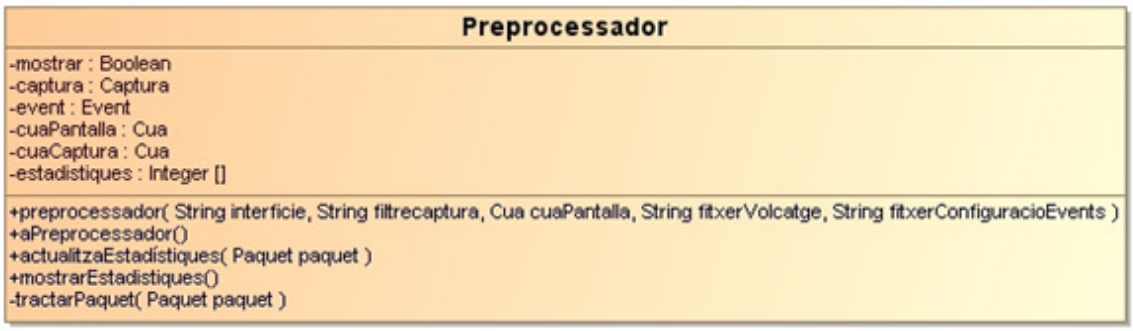


Figura 11: Diagrama estàtic de disseny: Relacions entre classes

b) Diagrames d'objectes: Atributs i operacions de cada classe.

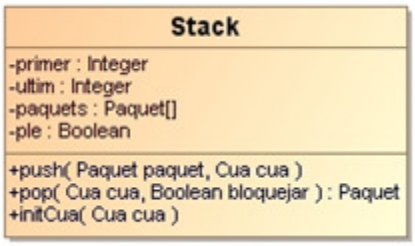
Classe Preprocessador



Classe Captura

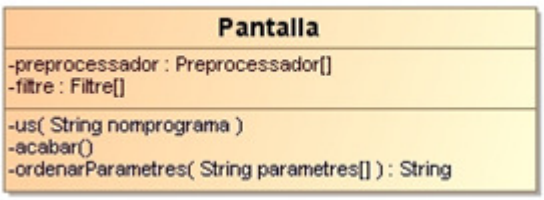
Classe Events

Classe Stack (Pila)



Classe Filtre

Classe Pantalla (Visualització)



CAPÍTOL 3.

DISSENY i IMPLEMENTACIÓ

3.1. Introducció

L'etapa de disseny és el procés d'elaboració de la proposta de treball segons les pautes i processos sistemàtics abans mencionats (capítol 1), fa de pont entre l'anàlisi i la implementació.

Un bon disseny ha de identificar per:

- establir un diagnòstic de la situació
- justificació dels objectius del projecte
- les funcions i actors clau que intervenen
- definir estratègies possibles per arribar a la solució
- resultats i/o productes esperats.

En aquest apartat es descriuen les funcions que ha de realitzar l'aplicació, també s'especifica els requisits no funcionals, es a dir problemes que ha de resoldre.

La implementació és la realització d'una aplicació, l'execució d'un pla, idea, disseny o posada en marxa de la proposta del projecte.

En el món informàtic, la implementació és l'etapa on es programa el sistema i/o l'aplicació.

Per a la realització d'aquest projecte, es necessita algun tipus d'eina que faciliti la implementació. S'ha cercat informació, de totes les solucions adequades al llenguatge emprat (C++) s'ha trobat que la més idònia és la utilització de "pcap" .

3.2. Llibreria pcap

El **pcap** és una interfície d'una aplicació de programació per la captura de paquets. La implementació del pcap per a sistemes basats en Unix es coneix amb el nom de "libpcap"[2]; el port per Windows del libpcap rep el nom de WinPcap.

Aquestes llibreries són el nucli de captura de paquets i els motors de filtració de moltes eines de codi obert i comercials de la xarxa, incloent analitzadors de

[2] Libpcap és una llibrería open source escrita en C. Abans de començar a programar amb Libpcat s'ha de tenir instal·lat una copia de les font oficials. (Es poden trobar a www.tcpdum.org)

protocol, monitors de xarxa, sistemes de detecció d'intrusos, programes de captura de trames de xarxa, generadors de trànsit i posta a punt de la xarxa.

Alguns programes que utilitzen libpcap/WinPcap: tcpdump, Snort, Wireshark, Nmap, Cain&Abel (eina per a recuperar contrasenyes per Windows, inclou també funcions de sniffer i detector de xarxes wi-fi).

Per la instal·lació del Libpcap només s'han de seguir tres passos habituals: configurar (./configure), construir(make) i instal·lar com root (make install). No importa la complicació del programa que vulguem fer am Libcap, aquest sempre ha de seguir un esquema bàsic com es mostra a la següent figura

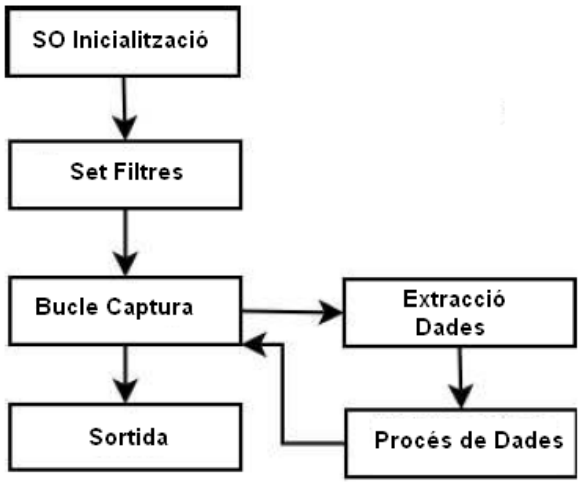


Figura 12. Esquema d'un programa amb Libpcap

Aquestes llibreries s'estructuren de manera el més simple possible per al desenvolupador, de manera que tenim una sèrie de funcions especifiques que duen a terme cadascun dels passos anteriorment esmentats.

3.2.1. Obtenció d'informació

La informació necessària tant de les interfícies de xarxa com de la seva configuració s'extreuen mitjançant les funcions seguidament descrites, i fent servir els tipus de dades propis de 'pcap'.

char *pcap_lookupdev(char *errbuf) ;

Funció que ens retorna en un String (char *) en C++, contenint el nom del primer dispositiu de xarxa vàlid del nostre sistema, en cas d'error torna NULL i una descripció d'aquest en el buffer.

int pcap_lookupnet(char *device, bpf_u_int32 *netp, bpf_u_int32 *maskp, char *errbuf) ;

Funció que ens retorna la direcció de xarxa i la màscara del dispositiu que li passem per paràmetres. En cas d'error retorna -1.

int pcap_findalldevs(pcap_if_t **alldevsp, char *errbuf) ;

Aquesta funció fa el mateix que 'pcap_lookupdev()' però ens proporciona la informació de totes les interfícies de xarxa vàlides per captura. En cas d'error torna -1.

int pcap_datalink(pcap t *p)

Aquesta funció ens retorna el tipus de xarxa, que en la majoria de casos serà Ethernet o Wireless.

3.2.2. Filtratge

Pcap ens proporciona la possibilitat de crear i aplicar filtres en el mateix mòdul de captura que ens serviran per descarregar de feina tant el sistema operatiu, com el nostre programa en els casos que es pugui fer.

Aquestos filtres tenen un format específic TCPDUMP, i permeten filtrar, entre d'altres, per adreces MAC, IP, ports, protocols i xarxes.

Quan es vol aplicar un filtre de captura primer s'ha de compilar, i un cop compilat s'ha d'enllaçar a un dispositiu de captura. Per fer-ho disposem de les següents funcions:

int pcap_compile(pcap_t *p,struct_bpf_program *fp,char *str,int optimize,bpf_u_int32 ne t);

Aquesta funció ens compila el filtre en format text i ens la retorna en un format compatible amb les funcions de captura. Hi ha la possibilitat d'optimitzar-lo.

int pcap_setfilter(pcap_t *p, struct_bpf_program *fp) ;

Per aplicar el filtre compilat a un dispositiu apliquem aquesta funció.

3.2.3.Captura de paquets

Aquí es on es mostren les diferents funcions que proporciona 'pcap' directament relacionades amb capturar dades:

pcap_t *pcap_open_live(char *device, int snaplen,int promisc, int to_ms, char *errbuf) ;

Abans de començar a capturar en un dispositiu s'ha d'obtenir el descriptor de tipus 'pcap_t', entre d'altres coses, per posar la interfície en mode promiscu si s'escau, un cop fet se'n pot enllaçar un filtre o obtenir-ne informació varia. Un paràmetre interessant es el nombre de mili-segons que volem que s'esperi per passar-nos les dades, ja que aquestes es capturen en zona del Kernel, però el nostre codi s'executa en zona d'usuari, i aquestos canvis tenen molt cost de computació.

int pcap_dispatch(pcap_t *p, int cnt,pcap handler_callback, u_char *user) ;

Aquesta funció s'utilitza per capturar els paquets, des de la interfície especificada, quan es capturen dades es crida la funció que li haguem especificat com a 'handler_callback i se li passa com a paràmetre els paquets corresponents. Es capturen com a màxim 'cnt' paquets. L'últim paràmetre es pel desenvolupador, per passar-li a la funció que manega els paquets algun paràmetre específic.

int pcap_loop(pcap_t *p, int cnt, pcap handler_callback, u_char *user) ;

Aquesta funció es molt semblant a l'anterior amb la diferencia que no acaba quan hi ha un error per *timeout*. És la que s'ha fet servir en el codi.

u_char *pcap_next(pcap_t *p, struct pcap_pkthdr *h) ;

Captura un paquet i en retorna el contingut.

3.3. Decisions preses en la implementació

S'ha enfocat el desenvolupament de l'aplicació cap a l'eficiència en la utilització dels recursos, la simplicitat d'ús i el rendiment, per poder-la fer servir com a part d'un sistema més complex com podria ser un IDS o un firewall auto-adaptable.
Es va decidir dotar l'aplicació de la capacitat d'obtenir informació de varies fonts alhora i poder aplicar una configuració per a cadascuna d'elles. L'esquema de classes implementat es basa en el que utilitza el famós *sniffer snort*, exemple a seguir per aquest projecte. Analitzant la funció que fa cadascuna de les classes podem concloure que:

- La classe Captura és l'encarregada d'inicialitzar el dispositiu i obtenir els paquets capturats, i immediatament després passar-los al Preprocessador per no perdre temps que podria provocar la pèrdua d'algun d'ells.
- La classe Preprocessador és l'encarregada d'agafar els paquets proporcionats per Captura i tractar-los segons els filtres i configuracions establerts per l'usuari. Immediatament després els ha de passar a la classe Pantalla per ser mostrats per pantalla si s'escau.
- La classe Events és l'encarregada de manipular els fitxers del sistema, tant a l'hora de guardar-hi informació com a l'hora de recuperar-la per ser tractada de nou.
- La classe Pantalla, com he dit, s'encarrega d'obtenir els paquets ja processats per mostrar-los per pantalla.

Primer de tot es va decidir utilitzar **threads** com a eina per a aconseguir tenir un codi adaptable i eficient, i es va dissenyar un esquema que pugues du a terme tota la càrrega de treball demanada, prioritzant que no hi haguessin threads corrent si no hi havia dades a processar, i obtenir un resultat fiable i reutilitzable. Es va decidir tenir, a part del thread principal, un thread per cada objecte Captura que anés obtenint dades i un altre thread per cada objecte Preprocessador que les processés. D'aquesta manera es crea un subsistema composat per un thread capturant dades i un thread processant-les per a cada font de dades demanada, que ens assegura que ni la captura de dades ni el procés d'aquestes quedaran

sense atendre, fet que podria provocar una pèrdua de dades o un resultat inesperat o no adient amb la realitat. Tots aquestos subsistemes proveeixen d'informació al thread principal el qual s'ha assignat una tasca ben simple per no provocar cap mena d'interferència entre els subsistemes abans descrits, que són els que realment fan la feina, que és la de mostrar la informació ja tractada per pantalla si s'escau (un cop el sistema ja ha començat a funcionar a ple rendiment). Cada objecte Preprocessador té associat un objecte Events per tal de du a terme les accions que l'usuari hagi descrit pel que fa a fitxers. Aquestos dos objectes (Preprocessador i Events) utilitzen el mateix thread quan s'executen, ja que es considera que formen part del mateix mòdul del programa.

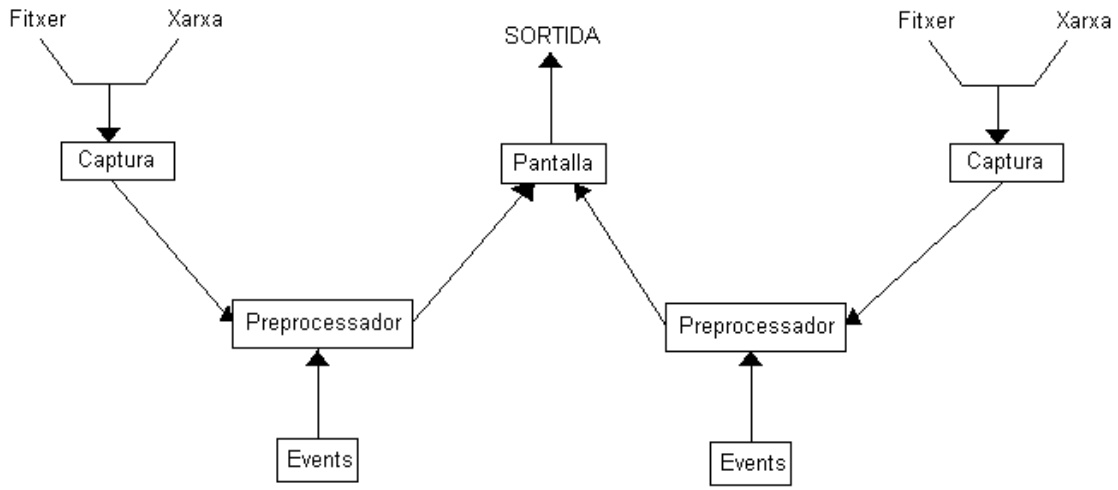


Figura 13. Esquema de threads d'execució

Un cop dissenyat l'esquema esmentat apareix la dificultat de treballar amb les dades de manera segura entre threads, que poden provocar malformació de la informació entre lectures i escriptures en zones de memòria comunes. Es tria implementar aquesta seguretat mitjançant mutex, que són semàfors que impedeixen l'accés a més d'un fil alhora a una determinada regió de memòria.
Un cop implementats els mutex sorgeix l'escenari en que cada fil d'execució passa les dades a l'altre de manera individual, es a dir, cada objecte Captura

passa cada paquet que rep al seu objecte Preprocessador, que quan ha fet la seva funció el passa a l'objecte Pantalla, tot “paquet a paquet”, provocant que el mètode d'implementació mitjançant threads no resulti tant òptim com podria. Com a solució per millorar l'efectivitat del sistema triat es creen piles de paquets FIFO en que cadascun dels objectes omple la pila del receptor de les seves dades, en un esquema productor-consumidor, i l'altre agafa aquestos paquets. D'aquesta manera s'evita que si un fil d'execució tarda més a realitzar la seva tasca amb un determinat paquet, aquest temps influeixi en que el thread que l'ha de proveir amb el següent paquet s'hagi d'esperar, fet que podria provocar un desbordament de buffer si es donen les condicions idònies.

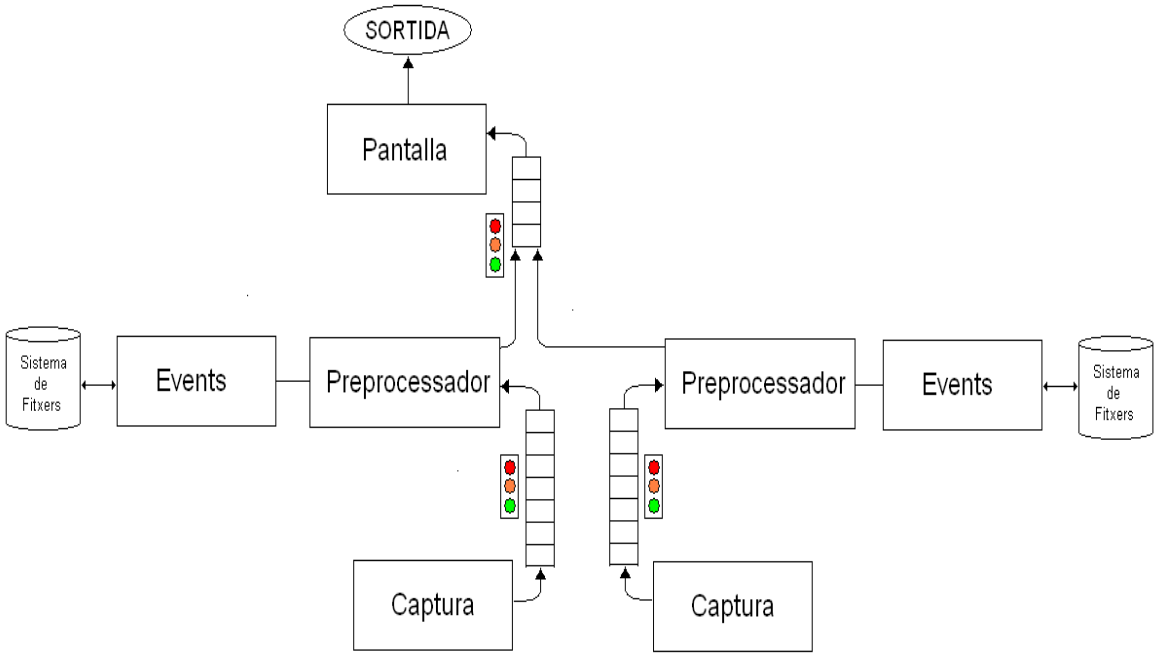


Figura 14. Esquema de piles i mutex

3.4. Anàlisi del codi

Hem creat fitxers de capçalera (.h) i de codi (.cpp) per cadascuna de les classes que hem planificat, i a més, a mode de llibreria i per simplificar-nos el desenvolupament, hem creat altres fitxers inclosos en les classes del projecte, per manegar en primer lloc piles FIFO, i per filtres que fa servir el programa. No hem volgut crear-ne classes perquè no constitueixen una part genèrica del programa, si no un mecanisme que fa servir el projecte de manera externa.

3.4.1.Stack

Realitza la funcionalitat d'una pila FIFO, i consta de quatre funcions amb tres objectius concrets, i de la definició de dues estructures de dades, paquet (capt) i cua:

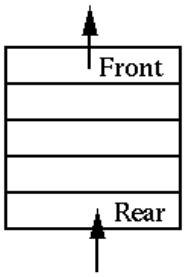


Figura 15. Esquema de pila

L'estructura 'capt' consta dels atributs que ens proporciona 'pcap' a l'hora de passar-nos paquets, que són el moment en que s'ha capturat (ts), la longitud que s'ha capturat (caplen), la longitud del paquet (len) i el propi paquet en el format de String.

```
struct capt
{
        struct timeval ts;
        bpf_u_int32 caplen;
        bpf_u_int32 len;
        u_char* packet;
};
```

L'estructura 'cua' conté els atributs necessaris per al bon funcionament d'una pila, que són els índex del primer i l'últim elements continguts, un array de paquets que fa de contenidor dels punters a les estructures de dades de paquets, un booleà que ens indica si la cua està completament plena o buida en el cas que els dos índex siguin iguals, i uns semàfors que s'han incorporat per evitar que hi hagi

errors en la lectura/escriptura de dades a la cua.

```
#define BUFFER_PAQUETS 15

struct cua
{
        int primer;
        int ultim;
        struct capt paquets[BUFFER_PAQUETS];
        bool ple;
        pthread_mutex_t mutex;
        pthread_cond_t cond;
};
```

La funció 'init_cua()' ens inicialitza la cua que li passem, en el cas que estigues plena ens la torna buida. Posa a 0 els punters i inicialitza el semàfor.

```
void init_cua (struct cua *cua);
```

Tenim dues versions de la funció 'push', les dues fan el mateix ja que una crida a l'altra. S'ha dissenyat d'aquesta manera per poder passar-li directament a la funció 'pcap_loop()' com a paràmetre. Aquesta funció ens posa el paquet a la cua que li passem.

```
int push (struct capt paq, struct cua *cua);
void push (u_char *c, const struct pcap_pkthdr* pkthdr,const u_char* packet);
```

L'última funció es la 'pop()' que ens retorna el paquet que correspon de la cua. La particularitat que té aquesta funció es que li podem passar l'argument 'block' que farà que el procés o thread que crida aquesta funció es bloquegi si no hi han paquets a la cua i continuï quan n'arribi un de nou.

```
#define BLOCK 1
#define NONBLOCK 1

struct capt pop (struct cua *cua, int block);
```

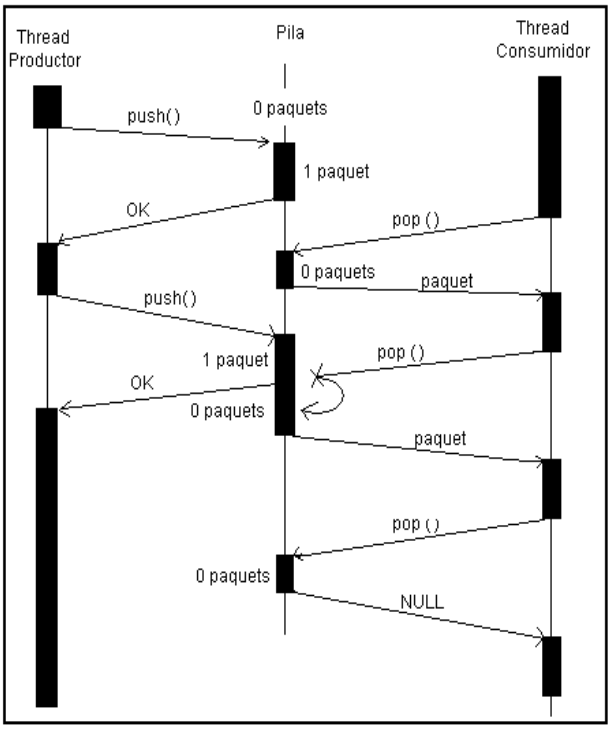


Figura 16. Funcionament de pila sense bloqueig

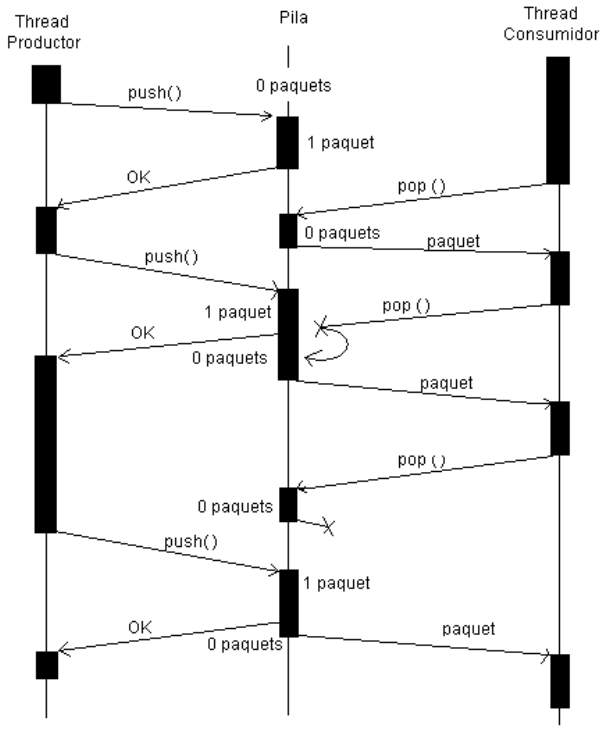


Figura 17. Funcionament de pila amb bloqueig

3.4.2. Filtratge

Aquest fitxer conté diferents funcions per manegar filtres que puguem tenir en el nostre programa, per comparar-hi paquets i per a mostrar-los per pantalla..
L'estructura 'filtre' conté els atributs que poden tenir qualsevol paquet, com són les adreces físiques, les IP's, els ports i el protocol, que després seran comparats amb les dades capturades per saber l'acció a dur a terme, i el tipus d'acció que comportarà que la comparació amb el filtre sigui positiva, alarma o log. Algun d'aquests camps pot ser '0', aleshores aquest camp no tindria influencia a l'hora de filtrar un paquet, tots passarien.

```
struct filtre
{
        struct ether_addr MAC[2];
        struct in_addr ip[2];
        u_int16_t port[2];
        char proto[6];
        int tipus;
};
```

La funció 'us()' ens imprimeix per pantalla les instruccions de format dels filtres en el cas que es produeixi algun error en el format d'aquests. En acabar finalitza l'execució del programa.

```
void us(bool visual);
```

La funció 'obrir_filtre()' llegeix el fitxer que se li passa per paràmetre i en transforma la informació que hi troba en filtres que després puguin ser aplicats. Alhora n'analitza els errors i els comunica a l'usuari fent servir la funció 'us()'. Ens retorna el descriptor del fitxer que conté el filtre per guardar-hi, en el cas que sigui un fitxer de configuració d'events, les possibles alarmes o logs, en altre cas retorna NULL.

```
FILE *obrir_filtre (char *f, struct filtre **filtres, bool filtre_visual);
```

La funció 'passa_filtre()' ens compara un determinat paquet amb el set de filtres que haguem configurat, i en el cas que hi hagi concordança completa ens retorna un atribut filtre amb totes les dades del paquet en qüestió, en cas contrari retorna NULL. S'ha dissenyat així per retornar totes les dades d'un paquet que concordi amb un filtre, ja que podem tenir algun camp d'aquest que sigui '0', per tant, a l'hora de registrar les dades del paquet ho farem amb la informació d'aquest, i no del filtre.

```
struct filtre *passa_filtre(struct capt paq, struct filtre **filtres);
```

La funció 'mostrar()' ens imprimeix per pantalla totes les dades interessants del paquet que li passem per paràmetre.

```
void mostrar(struct capt paq);
```

La funció 'us()' ens imprimeix per pantalla les instruccions de format dels filtres en el cas que es produeixi algun error en l'escriptura d'aquest. En acabar finalitza l'execució del programa.

```
void us(bool visual);
```

La funció 'mostrar()' ens imprimeix per pantalla totes les dades interessants del paquet que li passem per paràmetre.

```
void mostrar(struct capt paq);
```

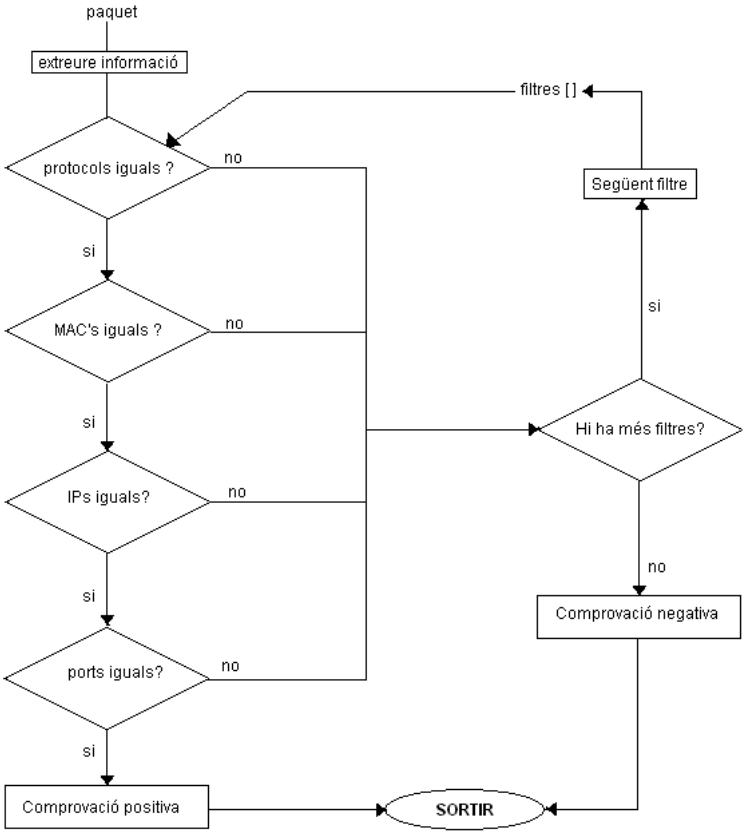


Figura 18. Esquema de filtratge d'un paquet

3.4.2.1. Format del filtres

Els filtres de visualització i de configuració d'events tenen un format específic que es descriu a continuació:

```
protocol MAC_origen|IP_origen|port_origen > MAC_destí|IP_destí|port_destí
```

Amb protocols possibles:

eth ip tcp udp icmp ipv6 arp revarp vlan

Format d'adreça MAC:

00:01:02:03:04:05

Format d'adreça IP:

192.168.0.1

En el cas de ser un filtre al fitxer de configuració d'events s'ha d'especificar l'acció a dur a terme com a prefix al filtre, per tant el format serà el següent:

```
alm/log protocol MAC_origen|IP_origen|port_origen > MAC_destí|IP_destí|port_destí
```

Si es volen registrar els events en un fitxer existeix la possibilitat d'especificar la ruta al final del filtre mitjançant '-w', i en el cas de que el fitxer existeixi es pot sobreescriure automàticament afegint l'opció '-wo'. Si es tracta d'un 'log' especificar aquest fitxer és obligatori, ja que l'únic objectiu del loggatge és registrar les comparacions positives en un fitxer.

```
alm/log protocol MAC_origen|IP_origen|port_origen > MAC_destí|IP_destí|port_destí -w[o] fitxer_events
```

Un exemple de fitxer de configuració d'events és el següent:

```
alm tcp any|192.168.0.1|any > any|any|80 -wo /home/usuari/Desktop/alarma.txt
```

Aquest filtre generaria alarmes per tots els paquets que vinguessin de la IP 192.168.0.1 amb port destí 80, i les gravaria al fitxer descrit.
En el cas que s'especifiquin protocols de nivell 2 o 3 existeixen uns requeriments, els quals es basen en que aquests tipus de paquets no fan servir o bé adreces IP (arp,revarp,vlan) o ports (icmp), per tant és obligatori posar '0' o 'any' en aquests camps.

3.4.3. Captura

La classe captura es l'encarregada d'obrir la interfície, de configurar-ne el filtre de captura i d'engegar el *thread* que omplirà la pila corresponent amb els paquets que capturi. Tenim varis atributs i funcions publiques:

El constructor 'Captura()' inicialitza el dispositiu i n'obté la informació, i en el cas que la sessió de captura sigui des d'un fitxer en guarda el descriptor per fer-lo servir quan comenci la sessió.

```
public Captura(char *disp, char *fil, pcap_t *descr=NULL);
```

El destructor '~Captura()' [3] es crida quan el programa finalitza, i finalitza el thread que està capturant les dades.

```
public ~Captura();
```

[3] A les definicions de classe, (punt 2.6 , apartat b) el signe ~ s'ha representat per "a", perquè el software utilitzat no permetia escriure el signe ~.

La funció 'init()' inicia el thread de captura i guarda el punter de la pila que li passa el Preprocessador.

```
public void init (struct cua *c);
```

La funció 'get_descr()' es fa servir per proporcionar el descriptor de la interfície on es captura per poder extreure'n les estadístiques.

```
public pcap_t *get_descr ();
```

Els atributs 'dispositiu', 'filtre', 'mask' i 'netaddr' són els atributs de la interfície enllaçada a l'objecte captura en qüestió, contenint el nom, el filtre de captura la màscara i l'adreça de xarxa.

```
public:
        char *dispositiu;
        char *filtre;
        bpf_u_int32 mask;
        bpf_u_int32 netaddr;
```

Les funcions i atributs privats es fan servir en les altres funcions accessibles públicament però no es necessari o segur que siguin accessibles des de fora.
La funció 'sniff()' es la que es fa servir com a funció del thread de captura, que crida a 'pcap_loop()' i s'hi queda bloquejat. Val a dir que la funció que es passa a 'pcap_loop()' per ser executada cada vegada que es capturen dades es la 'push()' continguda a la llibreria 'stack.h'.

```
public static void *sniff (void *stream);
```

El descriptor es el punter al tipus de dades 'pcap_t' que conté l'accés cap a la sessió de captura d'aquest objecte. No convé que sigui public, encara que s'hagi de fer servir externament, perquè un canvi en el seu valor faria que no fos accessible la interfície, i, per tant, que es bloqueges el programa. L'atribut 'filtrec' conté el filtre de captura ja compilat. El thread conté l'identificador del thread que està capturant, en el cas que el programa s'acabi es necessita per poder acabar la seva execució. La cua es un punter cap a la pila creada des de la classe preprocessador.

```
private:
        static void *sniff (void *stream);
        pcap_t* descriptor;
        struct bpf_program filtrec;
        pthread_t thread;
        struct cua *cua;
```

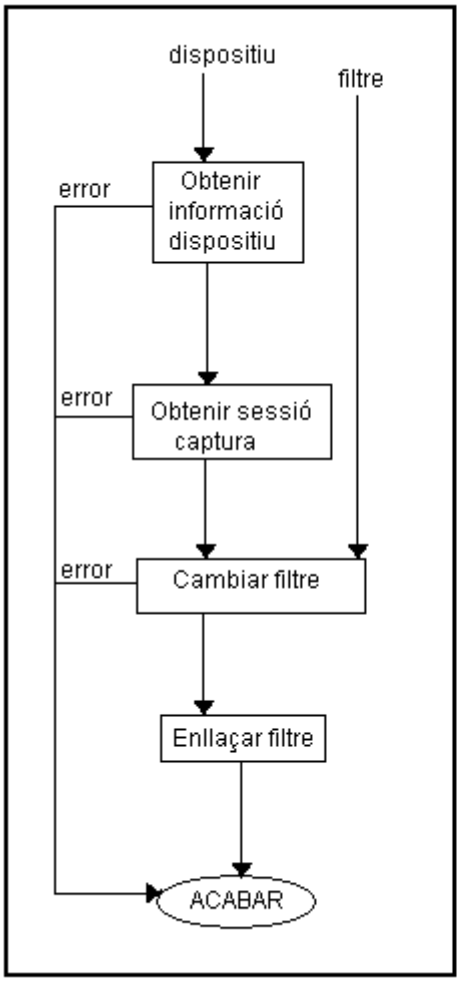


Figura 19

Construcció objecte Captura

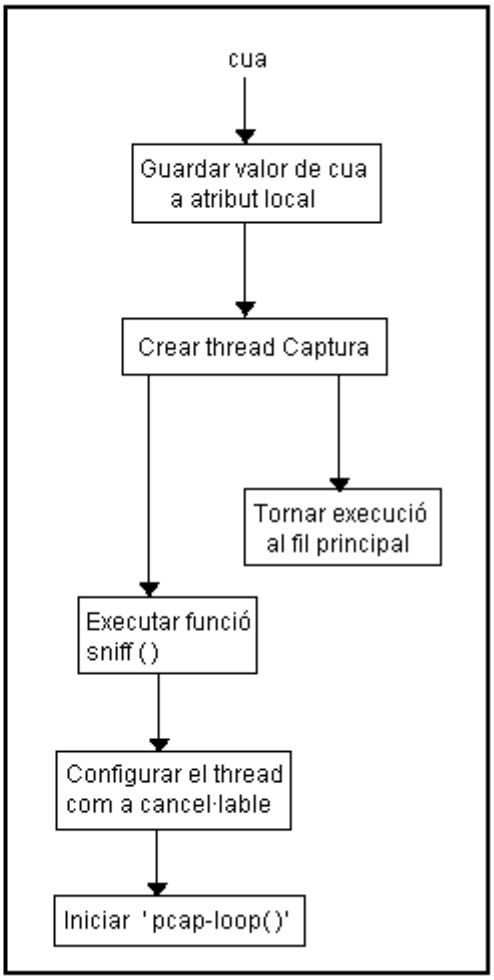


Figura 20.

Inicialització sessió Captura (init())

3.4.4. Preprocessador

La classe Preprocessador es l'encarregada de fer de pont entre la de Captura i Pantalla, per analitzar cadascun dels paquets que passen. També crea un thread que enllaça cada objecte Captura amb el fil principal Pantalla. La seva funció principal es anar agafant els paquets deixats a la pila per l'objecte Captura corresponent, processar-los, i tornar-los a posar a la pila de pantalla.

Tenim diferents funcions i atributs públics:

El constructor 'Preprocessador' que agafa com a paràmetres tots els necessaris per a la seva vegada crear una Captura, a més de la cua del fil principal, els fitxers de bolcat i de configuració d'events, i tres booleans que ens indiquen si aquest objecte Preprocessador ha de passar algun paquet a la pila de pantalla (mostrar), si l'objecte pertany a una sessió dump o no, que es diferencia per l'origen de les dades, siguin la interfície de xarxa o un fitxer de captura, i si en cas que el fitxer de bolcat especificat existeixi es pot sobreescriure. Durant l'execució d'aquest constructor es creen els objectes Captura i Events que farà servir, s'inicialitzen totes les estructures de dades necessàries i s'inicien la captura de dades (init()) i el thread de procés de paquets.

```
public Preprocessador(char *disp, char *fil, struct cua *c, char *fitxer, bool sobreescriure, bool
d, char *alarma, bool m);
```

El destructor '~Preprocessador()' [4]que es crida quan el programa finalitza i que destrueix l'objecte Captura al que està enllaçat i finalitza l'execució del thread que processa els paquets.

```
public ~Preprocessador();
```

La funció 'actualitzar_estadistiques()' es crida cada vegada que passa un paquet per l'objecte per tenir un recompte de tots els paquets que fins aquell moment han passat.

```
public void actualitzar_estadistiques(struct capt *pac);
```

[4] A les definicions de classe, (punt 2.6 , apartat b) el signe ~ s'ha representat per "a", perquè el software utilitzat no permetia escriure el signe ~.

La funció 'estadístiques()' es crida des del fil principal quan finalitza el programa i mostra les estadístiques fins aleshores recollides i crida la funció del mateix nom continguda a la classe 'Events' per obtenir les dades de la interfície oferides per 'pcap'.

```
public void estadistiques();
```

Els atributs privats són especialment els objectes amb els que treballa la classe, com poden ser la Captura, l'Event i les cues, després podem apreciar que tenim d'altres com l'identificador del thread que es crea (per poder-lo parar quan finalitza el programa), un array de filtres per comparar-hi els paquets i un array d'enters que representen els protocols dels paquets tractats fins al moment. Les funcions privades que conté són la que utilitza el thread creat per processar els paquets, funcio(), i la que s'utilitza per tractar els paquets que arriben per generar un event si s'escau.

```
private:
        static void *funcio(void *capt);
        void tractar(struct capt pac, int ev);
        Captura *captura;
        pthread_t thread;
        Events *event;
        FILE *fitxer_events;
        int tipus_event;
        struct cua *cua;
        struct cua *cua_pantalla;
        struct filtre *filtres[MAX_FILTRES];
        int estadist[N_PROTOCOLS];
        bool mostrar;
        bool dump;
```

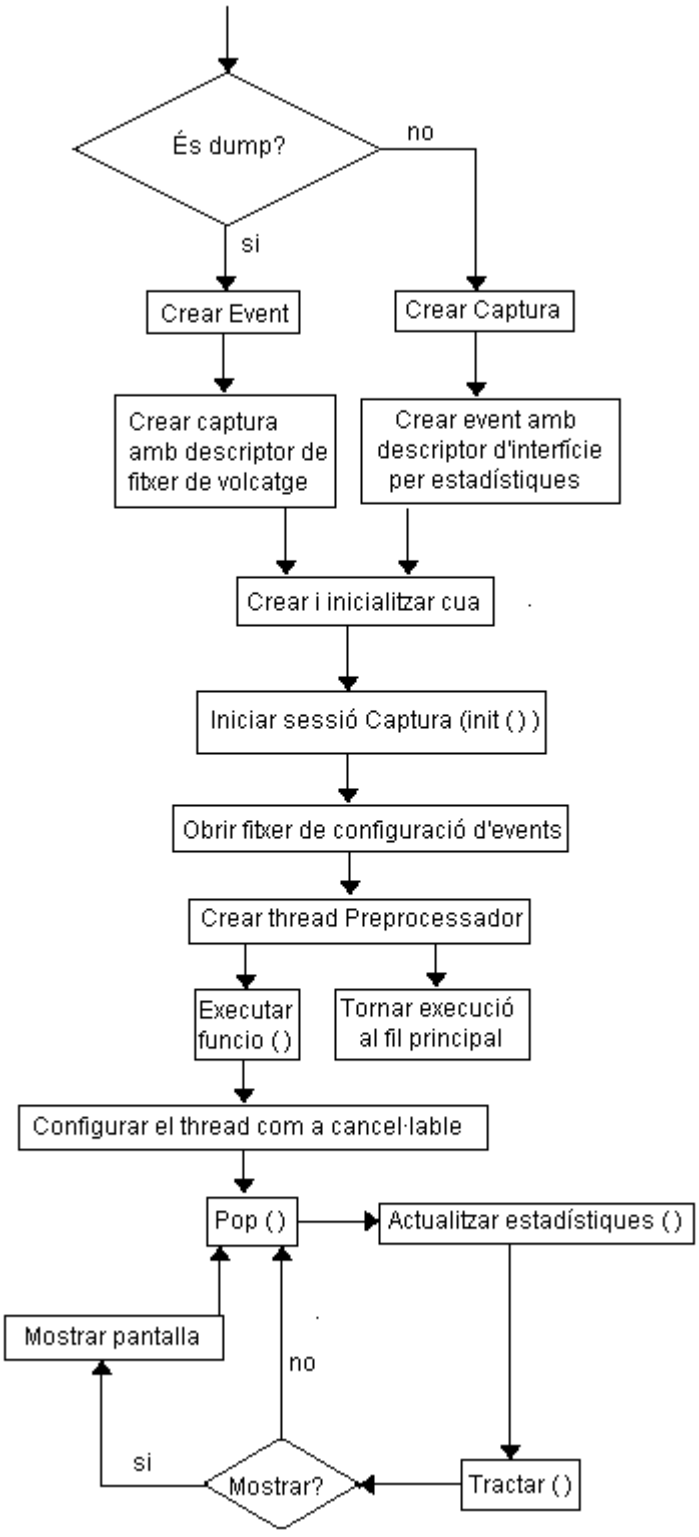


Figura 21: Esquema construcció Preprocessador

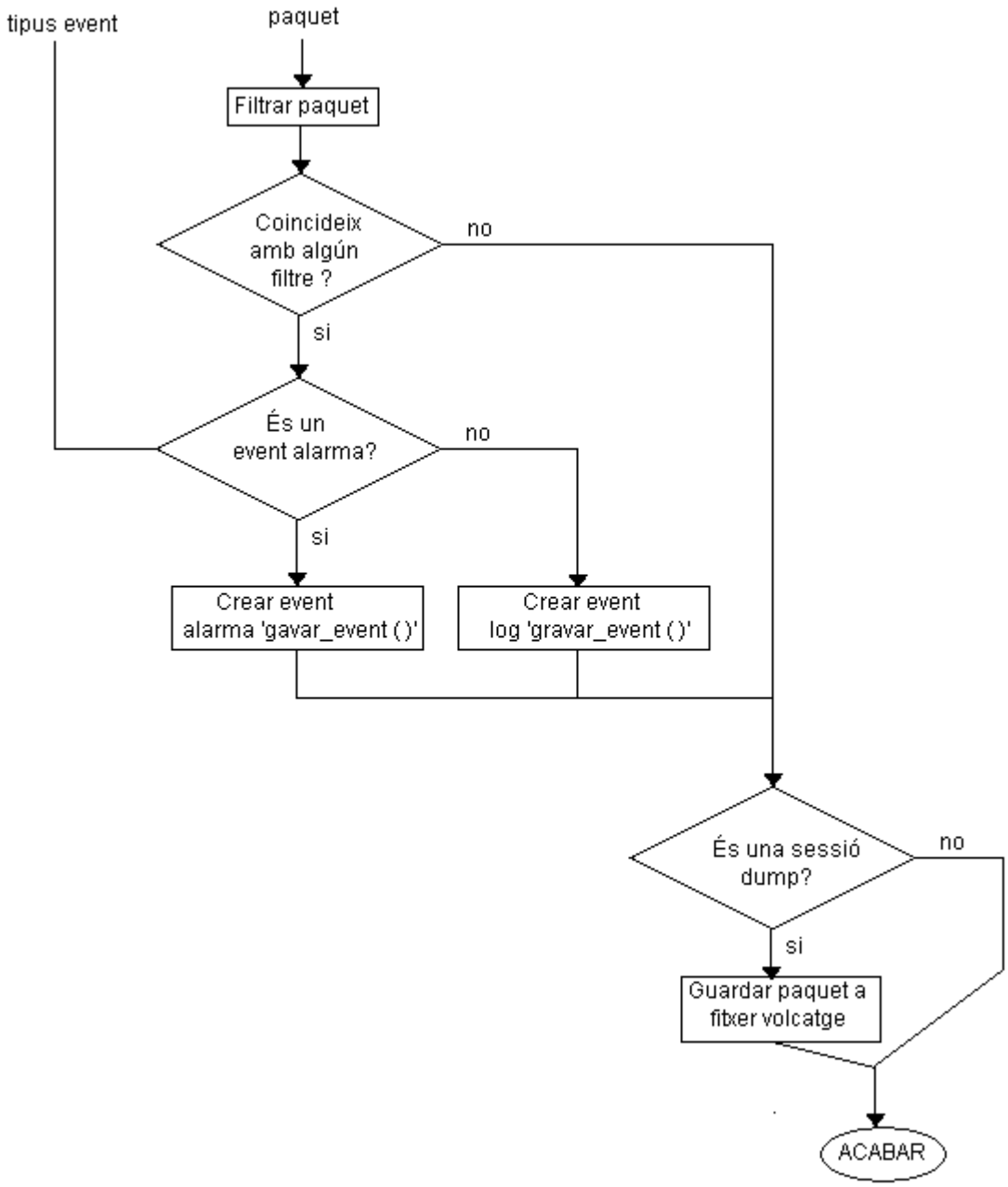


Figura 22: Esquema de la funció Tractar ( )

3.4.5. Events

La classe Events s'encarrega de manegar els fitxers de bolcat i on es gravaran les alarmes i logs, i les funcions que s'encarreguen de fer-ho. Aquesta classe no crea cap thread, les accions que es fan des d'aquí utilitzen el thread del Preprocessador, ja que les funcions es criden des d'aquell objecte.
Les funcions i atributs públics són:

El constructor 'Events()' que agafa per paràmetre el la ruta del fitxer, ja sigui per bolcar-hi les dades de la captura o per construir la sessió des d'un fitxer ja existent, un booleà que ens indica si el podem sobreescriure en el cas que ja existeixi, un descriptor de captura en el cas que el fitxer indicat sigui per guardar-hi paquets, i un altre booleà que ens indica quin dels dos tipus de sessió tenim, de captura des de fitxer o des de xarxa.

```
public Events(char *fit, bool sobreescriure, pcap_t* descriptor, bool dump);
```

El destructor '~Events()' que es crida quan es finalitza el programa i que tanca qualsevol fitxer que tinguem obert.

```
public ~Events();
```

La funció 'gravar_event()' té l'objectiu de rebre unes dades d'un paquet en forma de filtre, estampa de temps i protocol, per gravar-les al fitxer d'events, i mitjançant un booleà sap si es tracta d'una alarma o d'un log, la diferencia dels quals es que l'alarma ens mostra les dades per pantalla en el moment que es detecta.

```
public bool gravar_event (struct filtre *filtre, bool alarma, struct timeval *ts, char *protocol);
```

La funció 'estadistiques()' proporciona les dades de la sessió de captura que ofereix pcap mitjançant la funció 'pcap_stats()'.

```
public struct pcap_stat estadistiques (pcap_t *p);
```

La funció 'guardar_paquet()' emmagatzema el paquet que li passem al fitxer de bolcat que hem obert mitjançant el constructor. Es du a terme amb la funció de pcap 'pcap_dump()'.

```
public void guardar_paquet (struct capt pac);
```

La funció 'gravar_fitxer_alarma()' actualitza el descriptor del fitxer d'events que hi ha a l'objecte amb el que se li passa per paràmetre. Es dissenya així perquè la ruta del fitxer d'events està dintre del fitxer de configuració d'events, i per tant es competència de la llibreria 'filtratge.h'.

```
public void gravar_fitxer_alarma (FILE *f);
```

La funció 'get_descr_offline()' ens retorna el descriptor per a l'obtenció de dades des de fitxer. Aquesta funció es crida al moment de crear l'objecte captura per indicar-li que la font de dades es aquest fitxer.

```
public pcap_t *get_descr_offline ();
```

Els atributs privats de la classe tenen l'objectiu de contenir els descriptors dels diferents fitxers que es poden fer servir, i les funcions es fan servir per obrir i tancar aquestos fitxers en el constructor i destructor respectivament.

```
private:
        pcap_dumper_t *dumper;
        FILE *fitxer_alarma;
        pcap_t *dump_off;
        char *fitxer_ruta;
        pcap_t* descriptor;
        pcap_dumper_t *obrir_fitxer_dump (char *f, pcap_t* descr, bool sobreescriure);
        pcap_t *obrir_fitxer_offline (char *f);
        void tancar_fitxer (pcap_dumper_t *fitxer_bolcat);
        void tancar_fitxer (pcap_t *fitxer_guardat);
        void tancar_fitxer (FILE *fitxer_text);
```

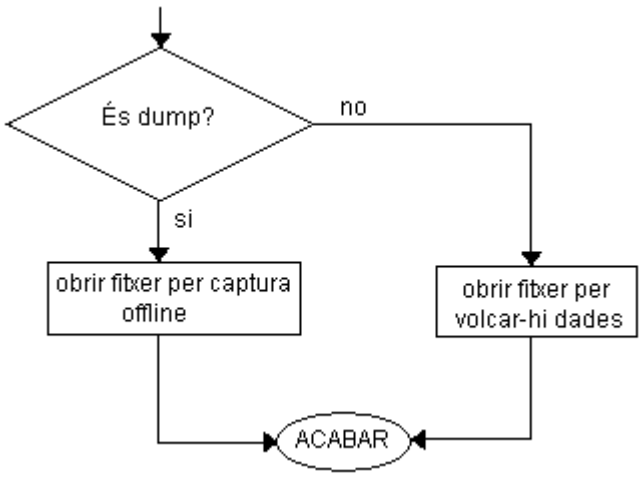


Figura 23: Esquema construcció objecte Events

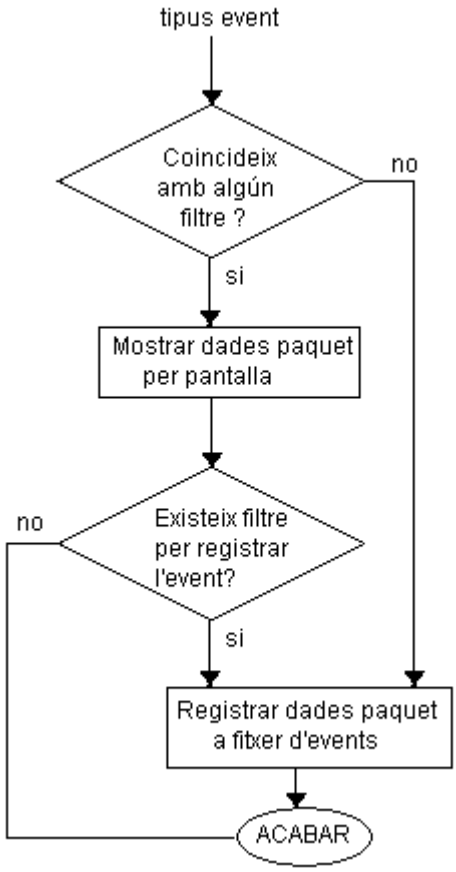


Figura 24: Esquema de funcionament de la funció 'gravar_event( )'

3.4.6.Pantalla

La classe Pantalla es la que conté el 'main' i la que s'ocupa d'obtenir i ordenar els paràmetres, crear un objecte Preprocessador per cada interfície que es demana i mostrar els paquets per pantalla segons arribin a la pila. S'han definit unes constants per limitar la quantitat d'interfícies a les que es pot escoltar simultàniament (5) i el nombre de filtres que s'han de tenir en compte, tant per visualització com per generar events(15).

També es aquí on es defineix la combinació de tecles que aturaran el funcionament del programa. S'ha escollit el senyal SIGINT que el provoca la combinació Ctrl+C, que executarà la funció 'acabar()'.

Conté dos atributs que són arrays, un d'objectes Preprocessador que es creen per cada interfície que es demana, i l'altre de filtres, on s'hi posaran els filtres de visualització demanats per l'usuari.

```
#define MAX_CAPTURES_SIMULTANIES 5
#define MAX_FILTRES 15

Preprocessador *prep[MAX_CAPTURES_SIMULTANIES];
struct filtre *filtres[MAX_FILTRES];
int main( int argc, const char* argv[] );
void us(const char *parametres);
void acabar(int param);
void ordenar_parametres(int argc, const char* argv[], char
*parametres[MAX_CAPTURES_SIMULTANIES][8]);
```

Les funcions que en hi trobem són:

El 'main()', que es la que es crida quan s'executa el programa i que manega tota la resta de funcions i atributs del codi. Es dedica a mostrar per pantalla tots els paquets que els Preprocessadors posen a la cua.

```
int main( int argc, const char* argv[] );
```

La funció 'us()' que ens imprimeix per pantalla les instruccions d'ús de l'aplicació en el cas que hi hagi algun error en els paràmetres, i seguidament surt.

```
void us(const char *parametres);
```

La funció 'acabar()' que s'executa quan rebem el senyal de finalitzar el programa i que s'ocupa de mostrar les estadístiques de tots els objectes Preprocessador creats i eliminar-los.

```
void acabar(int param);
```

La funció 'ordenar_parametres()' que rep com a paràmetres els que l'usuari passa al programa i els ordena de manera que puguin ser fàcilment interpretats pel codi.

```
void ordenar_parametres(int argc, const char* argv[], char *parametres[MAX_CAPTURES][8]);
```

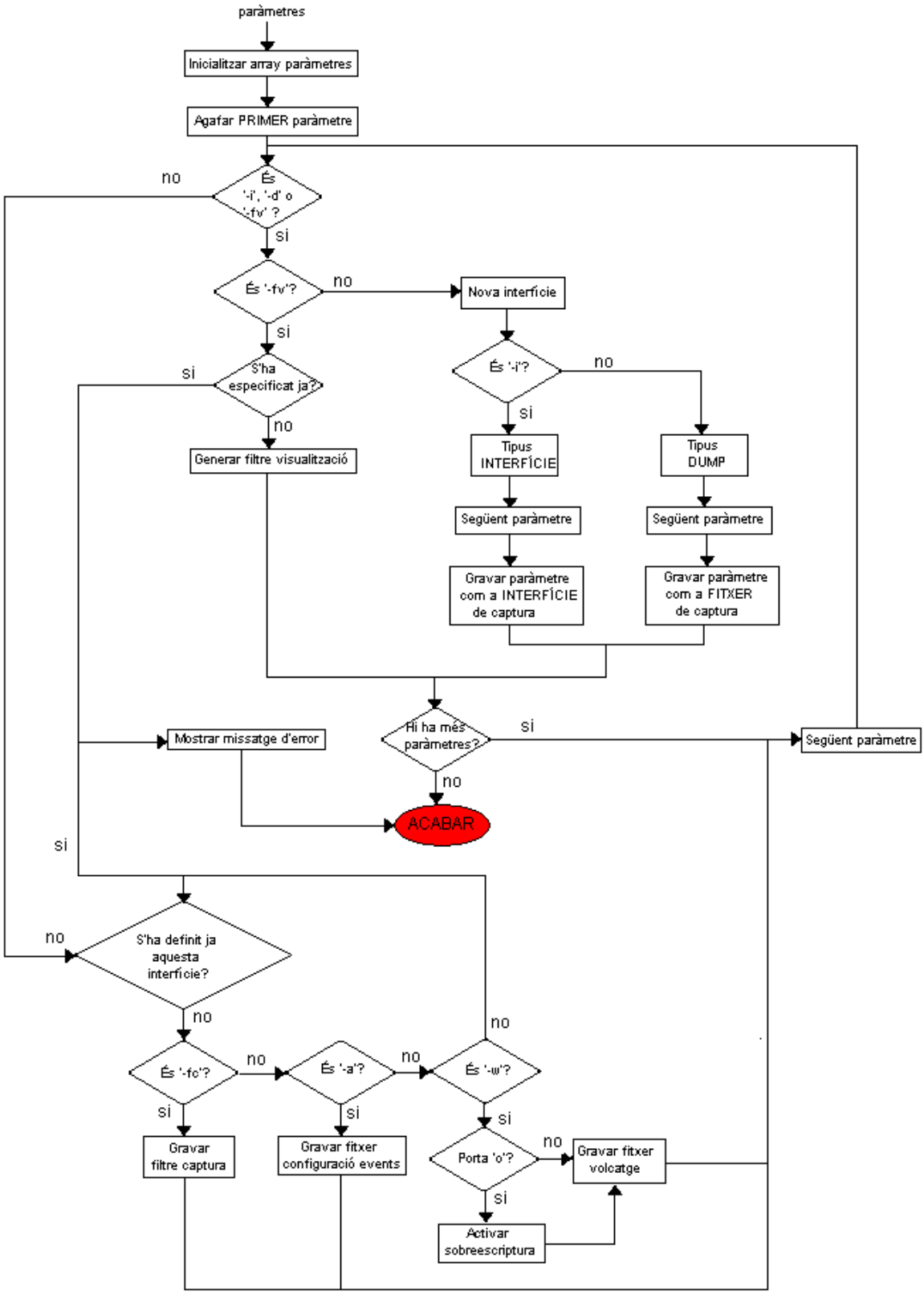


Figura 25: Esquema de funcionament de la funció 'ordenar_parametres( )'

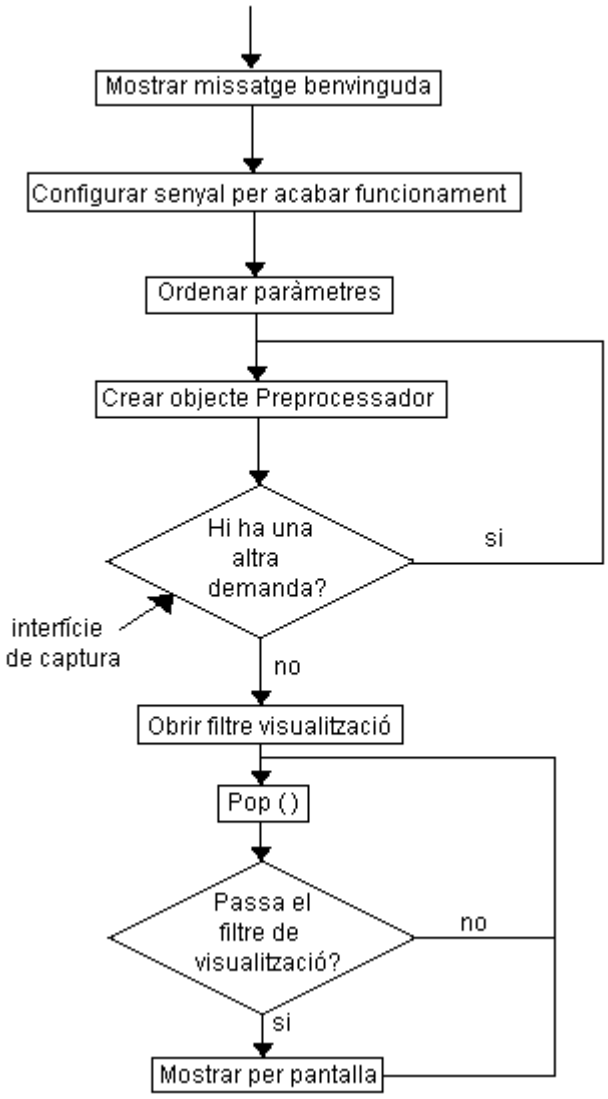


Figura 26: Esquema de funcionament de la funció 'main( )'

3.5. Visualització dels paquets

La visualització dels paquets es fa segons el protocol a que pertanyen i imprimint per pantalla tot un seguit d'informació del mateix. S'ha optat per una visualització resumida ja que l'aplicació està orientada a poder funcionar sota escenaris amb molt de trànsit. Un exemple de paquets capturats per la nostra aplicació en una sessió configurada perquè surtin per pantalla són els següents:

```
Timestamp=      1276195232,815129
MAC origen=     0:10:ff:c3:12:d4
MAC destí=      0:10:dd:12:f:00
Tipus=          IP
IP origen=      195.55.80.255
IP destí=192.168.0.1
```

```
Protocol=       TCP
Port origen=    80
Port destí=     46235
Flags=          ACK,
Timestamp=      1276195421,85781
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=127.0.0.1
Protocol=       ICMP Echo reply

Timestamp=      1276195503,831261
MAC origen=     0:10:ff:c3:12:d4
MAC destí=      ff:ff:ff:ff:ff:ff
Tipus=          ARP
```

A part dels paquets que captura l'aplicació el que també es visualitza són les alarmes que es produeixen, que mostren la mateixa informació però en un altre format per a la seva ràpida identificació. El que es visualitza per pantalla són les mateixes dades que es graven al fitxer si s'escau. Seguidament podem observar el format de visualització d'un paquet que genera una alarma:

```
Alarma!!
MACo:  0:10:ff:c3:12:d4        MACd:  0:10:dd:12:f:00
IPo:   192.168.0.1             IPd:   195.55.80.255
Porto: 46235                   Portd: 80
Protocol: TCP
Temps:  1276195825,443257
```

CAPÍTOL 4.

MANUALS

INSTAL·LACIÓ i ÚS

4.1. Manual d'instal·lació

Es proporcionen els fitxers de les diferents classes i llibreries de l'aplicació, a més d'un fitxer 'Makefile' per a fer la compilació i linkatge del codi més fàcil. Al ser una aplicació per a sistemes Linux, es requereix diferent software addicional compatible amb la plataforma per obtenir el codi executable final.

Els paquets principals requerits per obtenir el codi executable són 'make', 'libpcap', 'g++', 'binutils' i 'linux-headers', a més de paquets secundaris que pugin tenir dependències amb aquestos esmentats i llibreries que no es tinguin instal·lades. Es descriu el mètode a seguir en sistemes Debian o derivats, en els quals la manera més senzilla per instal·lar aquestos paquets és mitjançant el gestor de paquets Synaptic, o si no es disposa, mitjançant l'aplicació 'apt-get' o 'aptitude' (totes les comandes s'han d'executar com a superusuari):

```
apt-get install make libpcap0.8 g++ binutils linux-headers-$(uname -r)
```

Un cop instal·lades tots els paquets i les seves dependències s'ha de procedir a compilar i linkar l'aplicació mitjançant la següent comanda:

```
make install
```

Un cop fet això ja es pot executar el programa amb el nom 'capturador' seguit dels paràmetres addients.

4.2. Manual de l'usuari

L'aplicació s'ha d'executar des d'una consola de comandes com a superusuari amb el nom 'capturador'.
Si executem l'aplicació sense paràmetres ens sortirà una petita ajuda mostrant els paràmetres necessaris per a iniciar una sessió de captura:

```
Ús:

Per captura de dades des de xarxa:      capturador -i <interficie> [-w[o] <fitxer_bolcat>]

Per captura de dades des de fitxer:     capturador -d <fitxer>

Per definir filtre de visualització:    -fv <filtre_visualitzacio>
```

Per definir filtre de captura:	-fc <filtre_captura>
Per definir fitxer de configuració d'events:	-a <fitxer_conf_events>
Per activar la visualització de paquets en la interfície:	-v

Podem iniciar per defecte fins a cinc sessions simultànies de captura de qualsevol dels dos tipus, i especificar fins a quinze línies de filtratge en cadascun dels fitxers proporcionats.

4.2.1. Captura des de xarxa

Per iniciar una captura de dades des de xarxa hem d'especificar el paràmetre '-i' i com a mínim la interfície des d'on volem obtenir les dades, i si s'escau un fitxer a on bolcar-hi els paquets que capturem. Un exemple de captura de dades simple és:

```
capturador -i eth0
```

En aquest cas no aprofitem les capacitats de l'aplicació, ja que l'única funció que realitza és la de capturar, sense tractar les dades ni oferir-les a l'usuari. En el cas que vulguem imprimir tots els paquets capturats per pantalla hi hem d'afegir el paràmetre '-v':

```
capturador -i eth0 -v
```

Existeix l'opció en el cas de captura des de xarxa de bolcar la informació en un fitxer per analitzar-la a posteriori. El fitxer que especifiquem no té cap restricció de nom ni extensió que no siguin les del sistema operatiu fet servir, per tant no estem obligats a fer servir un format específic. En el cas de que el fitxer existeixi l'aplicació ens donarà error, però si volem que ens sobreescrigui el fitxer sense sortir ho podrem fer afegint-hi l'opció 'o':

```
capturador -i eth0 -v -w /home/usuari/Desktop/bolcat.dmp

capturador -i eth0 -v -wo /home/usuari/Desktop/bolcat.dmp
```

4.2.2. Captura des de fitxer

Per iniciar una captura de paquets des d'un fitxer de bolcat hem d'especificar el paràmetre '-d' i com a mínim la ruta del fitxer de font de dades. Igualment que en el cas de captura des de xarxa, si volem que se'ns mostrin les dades hem d'escriure el paràmetre '-v':

```
capturador -d /home/usuari/Desktop/bolcat.dmp -v
```

4.2.3. Filtre de visualització

Es possible especificar un filtre de visualització, que en cap cas sobrepassarà al paràmetre '-v'. El filtre ha d'estar contingut en un fitxer de text i s'ha de proporcionar amb el prefix '-fv'. Nomes es pot proporcionar un filtre de visualització per cada execució de l'aplicació, i es el que s'aplicarà a totes les sessions especificades. El format dels filtres esta descrit en l'apartat 3.5.2.1. Format de filtres, d'aquest document, un exemple del qual seria:

```
udp 00:01:02:03:04:05|192.168.0.1|any > A0:B1:C2:D3:E4:F5|192.168.0.2|80
```

Aquest filtre mostraria per pantalla tots els paquets 'udp' procedents de l'adreça MAC '00:01:02:03:04:05' amb IP ' 192.168.0.1', i destinació l'adreça MAC ' A0:B1:C2:D3:E4:F5' amb IP ' 192.168.0.2' i port 80. En el cas que haguem guardat el corresponent filtre en un arxiu anomenar 'visual.txt' a la carpeta actual, hauríem d'executar l'aplicació de la següent manera:

```
capturador -fv ./visual.txt -i eth0 -v
```

En el cas que haguem indicat més d'una sessió de captura el filtre només s'aplicarà a aquelles que continguin el paràmetre '-v'. Per exemple:

```
capturador -fv ./visual.txt -i eth0 -v -i lo
```

Aquesta comanda nomes mostrarà per pantalla els paquets coincidents amb el filtre provinents de la interfície 'eth0'. En el cas que vulguem que s'apliqui també a l'altra interfície haurem de posar:

```
capturador -fv ./visual.txt -i eth0 -v -i lo -v
```

4.2.4. Filtre de captura

Es possible especificar un filtre de captura, les principals diferencies que hi han amb el filtre de visualització són que se'n pot aplicar un per cada interfície o fitxer que s'introdueixi, i que el filtre de captura es passa directament a les llibreries 'pcap', per tant els paquets que no coincideixin amb aquestos no arribaran a sortir del Kernel amb el conseqüent estalvi de recursos del sistema. El format del filtre de captura ve definit per el manuals de 'pcap' inclosos en els annexos d'aquest document, i s'ha de proporcionar bé en un fitxer, o directament com a String des de la consola de comandes. Un exemple de filtre de captura és:

```
capturador -i eth0 -v -fc 'udp' -i lo -fc 'tcp' -v
```

En aquesta crida al programa nomes arribaran a l'aplicació els paquets udp procedents de la interfície 'eth0' i els paquets tcp procedents de la interfície 'lo', i tots es mostraran per pantalla.
A la seva vegada li podem aplicar un filtre de visualització contingut en el fitxer 'visual.txt' com el següent:

```
any any|any|12 > any|any|20
```

Per pantalla ens mostraria els paquets udp procedents de la interfície 'eth0' i els paquets tcp procedents de la interfície 'lo' que tinguessin com a port origen el 12 i destí el 20. Aquesta configuració s'executa d'aquesta manera:

```
capturador -i eth0 -v -fc 'udp' -i lo -fc 'tcp' -v -fv ./visual.txt
```

4.2.5. Fitxer de configuració d'events

El fitxer de configuració d'events té l'objectiu de configurar l'aplicació perquè busqui paquets amb un cert format i en reporti la detecció a l'usuari, i també se'n pot especificar un per cada interfície o fitxer des d'on es captura. Aquestos events poden ser alertes o logs, la diferencia entre els quals és que les alertes generen un missatge per pantalla amb la informació del paquet detectat i si s'escau el registren en un fitxer, i els logs nomes registren la informació del paquet en qüestió en el fitxer.
El fitxer s'ha d'especificar amb el prefix '-a' i ha de tenir un format específic, explicat amb més detall a l'apartat 3.5.2.1 Format de filtres:

```
alm tcp any|192.168.0.1|any > any|any|80 -wo /home/usuari/Desktop/alarma.txt
```

En el cas que tinguem dos fitxers de configuració d'events, 'events1.txt' i 'events2.txt', i vulguem capturar de dos interfícies la comanda a executar seria:

```
capturador -i eth0 -a events1.txt -i lo -a events2.txt
```

En aquest cas si es tracten d'alarmes no es necessari activar la mostra de paquets (-v) ja que quan es detecti un paquet coincident es mostrarà un missatge amb el format següent:

```
Alarma!!
MACo:    0:10:ff:c3:12:d4      MACd:  0:10:dd:12:f:00
IPo:     192.168.0.1           IPd:   195.55.80.255
Porto:   46235                 Portd: 80
Protocol: TCP
Temps:     1276195825,443257
```

4.2.6. Sortida de l'aplicació

L'aplicació conté un bucle infinit que no s'atura, per tant quan s'han complert els objectius del programa s'ha d'enviar el senyal SIGINT per provocar la seva finalització. El senyal es pot enviar o bé amb la combinació de tecles Ctrl+C a la consola de comandes on s'està executant, o amb la comanda 'kill -9 $PID' en una altra consola, essent $PID el número de procés de l'aplicació.
Quan es finalitza l'aplicació es mostren estadístiques de cada sessió de captura que s'ha demanat amb el format següent:

```
Estadistiques de eth0:

IP:                          34
TCP:                         24
UDP:                         6
ICMP:                        4
IPV6:                        0
ARP:                         2
REVARP:                      0
VLAN:                        0
DESCONEGUT:                  0

Nombre de paquets que han passat:    36
Nombre de paquets que s'han perdut:  0
```

En aquest cas es mostren les estadístiques corresponents a una captura de la interfície eth0.

CAPÍTOL 5.

JOC DE PROVES

5.1 Captura des de xarxa

A continuació es veu una captura des de la interfície 'lo', mentre des d'una consola estem fent pings a 'localhost'. Val a dir que apareixen com a capturats el doble de paquets que els mostrats perquè en aquesta interfície la màquina origen es la mateixa que la destí, per tant els paquets es dupliquen.

```
user@laptop:~/$ sudo capturador -i lo -v

**********Sniffer de xarxa**********
**                                  **
**        Realitzat per:            **
**                                  **
**         Pilar Arbonès Arbonès    **
**         Alumne d'ETIS per la UOC **
**                                  **
**************************************

Capturant trànsit ...
Timestamp=      1276282269,559122
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=127.0.0.1
Protocol=       UDP
Port origen=    40480
Port destí=     40480

Timestamp=      1276282270,76334
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=127.0.0.1
Protocol=       ICMP Echo

Timestamp=      1276282270,76358
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=127.0.0.1
Protocol=       ICMP Echo reply

^C
Estadistiques de lo:

IP:                     3
TCP:                    0
```

```
UDP:                    1
ICMP:                   2
IPV6:                   0
ARP:                    0
REVARP:                 0
VLAN:                   0
DESCONEGUT:  0

Nombre de paquets que han passat:       6
Nombre de paquets que s'han perdut:     0
```

5.2 Captura des de xarxa amb filtre de captura

A continuació es mostra una captura efectuada immediatament després de l'anterior però filtrant nomes els paquets 'udp'.

```
user@laptop:~/$ sudo capturador -i lo -v -fc udp

**************Sniffer de xarxa************
**                                        **
**         Realitzat per:                 **
**                                        **
**          Pilar Arbonès Arbonès         **
**          Alumne d'ETIS per la UOC      **
**                                        **
****************************************

Capturant trànsit ...
Timestamp=      1276282736,590087
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=       127.0.0.1
Protocol=       UDP
Port origen=    40480
Port destí=     40480

Timestamp=      1276282736,590098
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=       127.0.0.1
Protocol=       UDP
Port origen=    40480
Port destí=     40480

^C
Estadistiques de lo:
```

```
IP:                 26
TCP:                0
UDP:                2
ICMP:               24
IPV6:               0
ARP:                0
REVARP:             0
VLAN:               0
DESCONEGUT:         0

Nombre de paquets que han passat:    52
Nombre de paquets que s'han perdut:  0
```

5.3 Captura des de xarxa amb fitxer de bolcat

Aquí es mostra com es genera un fitxer de bolcat amb els paquets capturats.

```
user@laptop:~/$ sudo capturador -i lo -wo ./bolcat.dmp

**************Sniffer de xarxa************
**                                        **
**          Realitzat per:                **
**                                        **
**           Pilar Arbonès Arbonès        **
**           Alumne d'ETIS per la UOC     **
**                                        **
****************************************

Capturant trànsit ...
^C
Estadistiques de lo:

IP:                 166
TCP:                6
UDP:                148
ICMP:               12
IPV6:               6
ARP:                0
REVARP:             0
VLAN:               0
DESCONEGUT:         0

Nombre de paquets que han passat:    384
Nombre de paquets que s'han perdut:  0
```

5.4 Captura des de xarxa amb fitxer de configuració d'events

Aquesta vegada hem configurat els paràmetres perquè ens alerti de qualsevol connexió TCP que es produeixi cap al port tcp 22. Una vegada engegat des de la mateixa màquina hem intentat una connexió ssh i aquest n'ha sigut el resultat:

```
alm tcp any|any|any > any|any|22 -wo ./alarma.txt
```

```
user@laptop:~/$ sudo capturador -i lo -a alrm

**********Sniffer de xarxa**********
**                                  **
**        Realitzat per:            **
**                                  **
**         Pilar Arbonès Arbonès    **
**         Alumne d'ETIS per la UOC **
**                                  **
**************************************

Capturant trànsit ...
Alarma!!
MACo: 0:0:0:0:0:0          MACd: 0:0:0:0:0:0
IPo:  127.0.0.1            IPd:  127.0.0.1
Porto: 45421               Portd: 22
Protocol: TCP
Temps:  1276283414,576589

^C
Estadistiques de lo:

IP:                 19
TCP:                2
UDP:                5
ICMP:               12
IPV6:               2
ARP:                0
REVARP:             0
VLAN:               0
DESCONEGUT:         0

Nombre de paquets que han passat:   42
Nombre de paquets que s'han perdut: 0
```

5.5 Captura des de fitxer

La possibilitat de recuperar dades des de fitxers salvats anteriorment és interessant, i en aquest moment recuperarem la informació continguda en el fitxer creat a l'apartat 6.3:

```
user@laptop:~/$ sudo capturador -d ./bolcat.dmp

**************Sniffer de xarxa************
**                                        **
**          Realitzat per:                **
**                                        **
**           Pilar Arbonès Arbonès        **
**           Alumne d'ETIS per la UOC     **
**                                        **
****************************************

Capturant trànsit ...
^C
Estadistiques de lo:

IP:                     166
TCP:                    6
UDP:                    148
ICMP:                   12
IPV6:                   6
ARP:                    0
REVARP:                 0
VLAN:                   0
DESCONEGUT:             0
```

5.6 Captura des de fitxer amb filtre de captura

Igualment com en la captura des de xarxa podem aplicar al fitxer un filtre de captura:

```
user@laptop:~/$ sudo capturador -d ./bolcat.dmp -fc icmp

**************Sniffer de xarxa************
**                                        **
**          Realitzat per:                **
**                                        **
**           Pilar Arbonès Arbonès        **
**           Alumne d'ETIS per la UOC     **
**                                        **
****************************************
```

```
Capturant trànsit ...
^C
Estadistiques de lo:

IP:                 12
TCP:                0
UDP:                0
ICMP:               12
IPV6:               0
ARP:                0
REVARP:             0
VLAN:               0
DESCONEGUT:         0
```

5.7 Captura des de fitxer amb fitxer de configuració d'events

És útil també poder aplicar la funcionalitat d'events a fitxers amb captures, ja que ens permet que el programa analitzi tota la captura i ens avisi de diverses accions o coincidències que puguem buscar una vegada el trànsit s'ha produït. Hem configurat els paràmetres perquè ens alerti de qualsevol connexió TCP que es produeixi cap al port tcp 22, igual que en l'apartat 6.4. Una vegada engegat des de la mateixa màquina hem intentat una connexió ssh i aquest n'ha sigut el resultat:

```
alm tcp any|any|any > any|any|22 -wo ./alarma.txt
```

```
user@laptop:~/$ sudo capturador -i lo -a alrm

**********Sniffer de xarxa**********
**                                  **
**        Realitzat per:            **
**                                  **
**         Pilar Arbonès Arbonès    **
**         Alumne d'ETIS per la UOC **
**                                  **
**************************************

Capturant trànsit ...
Alarma!!
MACo: 0:0:0:0:0:0          MACd: 0:0:0:0:0:0
IPo:  127.0.0.1            IPd:  127.0.0.1
Porto: 45421               Portd: 22
Protocol: TCP
Temps:  1276283414,576589
```

```
^C
Estadistiques de lo:

IP:                 19
TCP:                2
UDP:                5
ICMP:               12
IPV6:               2
ARP:                0
REVARP:             0
VLAN:               0
DESCONEGUT:         0
```

5.8 Captura des de dues interfícies

Una de les diferencies amb snort que s'ha volgut que aquesta aplicació tingui és la possibilitat d'iniciar vàries sessions de captura en vàries interfícies simultàniament. En el següent exemple capturem de les interfícies 'lo' i 'eth0' fent pings per cadascuna d'elles, i aquest n'és el resultat:

```
user@laptop:~/$ sudo capturador -i lo -v -i eth0 -v

**********Sniffer de xarxa**********
**                                  **
**        Realitzat per:            **
**                                  **
**         Pilar Arbonès Arbonès    **
**         Alumne d'ETIS per la UOC **
**                                  **
**************************************

Capturant trànsit ...
Timestamp=      1276428778,691795
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=       127.0.0.1
Protocol=       ICMP Echo

Timestamp=      1276428778,691813
MAC origen=     0:0:0:0:0:0
MAC destí=      0:0:0:0:0:0
Tipus=          IP
IP origen=      127.0.0.1
IP destí=       127.0.0.1
Protocol=       ICMP Echo reply
```

```
Timestamp=      1276428778,945906
MAC origen=     0:1:2:3:4:5
MAC destí=      5:4:3:2:1:0
Tipus=          IP
IP origen=      192.168.0.2
IP destí=       192.168.0.1
Protocol=       ICMP Echo

Timestamp=      1276428778,946242
MAC origen=     5:4:3:2:1:0
MAC destí=      0:1:2:3:4:5
Tipus=          IP
IP origen=      192.168.0.1
IP destí=       192.168.0.2
Protocol=       ICMP Echo reply
^C

Estadistiques de lo:

IP:                     2
TCP:                    0
UDP:                    0
ICMP:                   2
IPV6:                   0
ARP:                    0
REVARP:                 0
VLAN:                   0
DESCONEGUT:             0

Nombre de paquets que han passat:    4
Nombre de paquets que s'han perdut:  0

Estadistiques de eth0:

IP:                     2
TCP:                    0
UDP:                    0
ICMP:                   2
IPV6:                   0
ARP:                    0
REVARP:                 0
VLAN:                   0
DESCONEGUT:             0

Nombre de paquets que han passat:    2
Nombre de paquets que s'han perdut:  0
```

CAPÍTOL 6.

CONCLUSIONS i MILLORES

6.1. Conclusions

Tal i com es deia a l'inici d'aquest document l'objectiu del TFC era la creació d'una eina per a la captura i posterior anàlisi del trànsit d'una xarxa determinada, en entorn GNU; aplicant els coneixements adquirits al llarg dels estudis d'Enginyeria Tècnica Informàtica de Sistemes dins les diferents apartats de disseny, programació i implementació.

Si tinc en compte que he tingut de compaginar la feina, la família, altres matèries estudiades, aquest treball i problemes de hardware que vaig tenir a l'inici, crec que l'objectiu marcat ha estat aconseguit, amb esforç i moltes dificultats.

Una dificultat que he tingut és que els coneixements adquirits al llarg dels estudis han hagut de ser ampliats i molts altres recordats. En l'apartat de programació he tingut un retard important, he hagut de fer un "reciclatge" molt ràpid perquè tenia mancances.

L'altra dificultat és que no he pogut arribar al punt que m'hagués agradat (per manca de temps i les dificultats abans dites) que era realitzar la interfície gràfica visualment agradable. Considero que la dificultat i el temps puja exponencialment amb la complexitat del programa i que aquest treball comporta bastanta càrrega de temps. Aquest aspecte m'ha angoixat bastant, veia que el plaç d'entrega s'anava acostant i semblava que no adelantava.

Per sort he tingut el suport del sofware en la creació dels diagrames (MagicDraw) i quan realitzava codi (Eclipse), això m'ha permès anar una mica més descansada en aquest aspecte.
M'ha ajudat el fet que tingues molt recent els coneixements i pràctiques realitzades a les assignatures de Estructura de Xarxes de Computadors i Seguretat en Xarxes de Computadors.
També ha estat positiu el fet que al mateix temps de realitzar aquest treball, estigués cursant Enginyeria del programari, cosa que m'ha permès desenvolupar amb més facilitat els gràfics dels casos d'ús, diagrama de seqüències, etc.

Al llarg dels estudis, totes les matèries cursades estan molt planificades i si segueixes els terminis, aquestos et van portant a aconseguir superar la matèria. En el TFC això no és així, és un mateix el que ha de posar la planificació i la temporalització, i si no es preveu correctament o (el que m'ha passat) es tenen dificultats tècniques; es pot pagar un alt preu. Per sort, he arribat a aconseguir una aplicació de bona utilitat.

6.2. Millores

De ben segur estarem d'acord en que totes les coses son millorables i aquesta aplicació no n'és cap excepció. Així doncs, seguidament anomeno els aspectes que es podrien millorar o introduir.

- **Interfície gràfica:** Posar-hi menús per realitzar la captura.
- **Gràfics estadístics:** Opcionalment als menús.
- **L'escalabilitat:** Que es pugui utilitzar amb equips més potents i xarxes més grans, amb més demanda.
- **Donar-li prestacions de IDS** (Intrusion Detection System): Alarmes i logs a més gran nivell.

GLOSSARI

Ethernet: És el protocol de xarxa de nivell 2 (capa física) més popular de la tecnologia LAN que es fa servir actualment. És popular perquè permet un bon equilibri entre velocitat, cost i facilitat d'instal·lació. Aquests punts, l'àmplia acceptació en el mercat i l'habilitat de suportar virtualment tots els protocols de xarxa populars, fan d'Ethernet la tecnologia ideal per a la majoria dels usuaris de la informàtica actual. La norma Ethernet va ser definida per l'Institut per als Enginyers Elèctric i Electrònics (IEEE) com a IEEE Standard 802.3. Afegint-se a la norma de l'IEEE, els equips iprotocols de xarxa poden interoperar eficaçment.

Intranet: Xarxa privada disponible només dins la organització que l'ha creada. Utilitza recursos de Internet. Permet un accés fàcil, ràpid a la informació corporativa dels usuaris que la integren. Així mateix unifica el tipus d'eines que es fan servir.

LAN: *(Local Area Networks)*. **Xarxa Local**. Conjunt d'ordinadors independents que es comuniquen a través d'un mitjà de xarxa compartit. Les xarxes d'àrea local són aquelles que connecten una xarxa d'ordinadors d'una àrea geogràfica o física més o menys àmplia com poden ser un campus universitari, o un edifici…. El desenvolupament de normes de protocols de xarxa i mitjans físics, han fet possible l'augment de LAN's en grans organitzacions multinacionals, aplicacions industrials, educatives, comercials, etc.

WAN: *(Wide Area Network)*. **Xarxa de gran abast**. Aquestes xarxes, connecten múltiples LAN's que estan geogràficament disperses. Això es possible realitzant la connexió de diferents LAN mitjançant serveis de línies telefòniques llogades (punt a punt), línies telefòniques normals amb protocols síncrons i asíncrons, enllaços via satèl·lit i serveis portadors de paquets de dades.

VLAN: Una VLAN (acrònim de *Virtual LAN*, xarxa d'àrea local virtual) és un mètode per a crear xarxes lògicament independents dins una mateixa xarxa física. Varies VLAN's poden coexistir en un únic computador físic o en una única xarxa física. Són útils per a reduir el tamany del "domini de difusió" i ajuden en l'administració de la xarxa separant segment lògics d'una xarxa local (com els departaments d'una empresa) que no han d'intercanviar dades utilitzant la xarxa local (malgrat ho poden fer mitjançant un enrutador o switch). Una "VLAN" consisteix en una xarxa d'ordinadors que es comporten con si estiguessin connectat al mateix commutador, malgrat poden estar en realitat connectats físicament a diferents segments d'una xarxa d'àrea local. Els administradors de xarxa configuren les VLAN's mitjançant software en lloc de hardware, el que fa que siguin molt flexibles. Una de les gran avantatges de les VLAN's és quan es trasllada físicament un ordinador a un altre lloc: poden estar en la mateixa VLAN sense necessitat d'haver de cambiar la configuració IP de la màquina.

Protocol: Normes que permeten als ordinadors comunicar-se. Descripció formal de formats de missatges i regles que dos o més màquines han de seguir per a poder intercanviar missatges.

TCP: **Protocol de control de transmissió**, de nivell de transport TCP/IP que proporciona el servei de flux confiable full dúplex i del qual depenen moltes aplicacions. És un dels protocols fonamentals a Internet. Aquest protocol garanteix que les dades seran entregades a la seva destinació sense errades i amb el mateix ordre que es van enviar. També proporciona un

mecanisme per a distingir diferents aplicacions a dins d'una mateix màquina, a través del concepte de port. Suporta moltes de les aplicacions de Internet com HTTP, SMTP i SSH.

UDP: (*User Datagrama Protocol)*. És un protocol del nivell de transport basat en l'intercanvi de datagrames. Permet l'enviament de datagrames a través de la xarxa sense que hi hagi connexió prèvia establerta. El mateix datagrama incorpora a la seva capçalera la informació de direccionament necessària. El seu ús principal és per a protocols ARP, ICMP, DHCP, BOOTP, ,… i per altres protocols on intercanvi de informació de paquets és molt gran o es necessita un alt rendiment, com poden ser les videoconferències

ICMP: (*Internet Control Message Protocol*). **Protocol de Missatges de Control d'Interxarxa**. És un protocol que per al seu funcionament utilitza directament el protocol IP dins de l'arquitectura TCP/IP. La seva funció és informar de l'estat i situacions d'error en el funcionament de la capa de xarxa, sobretot en aspectes com l'encaminament, congestió, fragmentació, etc.
Els missatges ICMP es transmeten a l'interior de datagrames IP. Consten d'una capçalera de 64 bits, on apareixen camps com tipus, codi, SVT i informació variable. A continuació de la capçalera trobem l'espai per a dades.
ICMP, al contrari que TCP i UDP, no s'utilitza directament per les aplicacions d'usuari. L'excepció és l'aplicació Ping, que envia missatges de petició Echo Request (i rep missatges de resposta Echo Reply) per a determinar si un host està disponible i el temps que empren els paquets en anar i tornar a eixa màquina.

IP: Protocol d'Internet. Unitat bàsica de informació que passa a través d'una xarxa. És un protocol no orientat a connexió que es fa servir tant per l'origen com pel destí de la informació de paquets commutats. Les dades són enviades en blocs anomenats datagrames (paquets). IP, no necessita cap configuració per enviar paquets entre equips que no s'havien connectat anteriorment.

IPv6: És la versió 6 del protocol d'Internet (IP), és un estàndard de nivell de xarxa que s'encarrega de dirigir i encaminar els paquets commutats. Està dissenyat per a substituir l'actual IPv4, ja que el seu límit en el nombre d'adreces de xarxa disponibles està començant a restringir el creixement d'Internet (l'elevat increment d'assignacions a zones d'Àsia n'està accelerant l'exhauriment). Actualment, ja hi ha assignades més de 2/3 parts de les que disposa. La nova IPv6, permetrà proporcionar als futurs telèfons cel·lulars i mòbils, una direcció fixa i pròpia per a cada un d'ells.

ARP: (*Address Resolution Protocol)***. Protocol de resolució d'adreces d'internet.** És un protocol de nivel de xarxa responsable de trobar l'adreça hardware (Ethernet MAC) que correspon a una determinada adreça IP. Per a fer-ho, s'envia un paquet (ARP request) a l'adreça de difusió de la xarxa (broadcast (MAC= xx xx xx xx xx xx) que conté l'adreça IP per la qual es pregunta, i s'espera que la màquina respongui (ARP reply) amb l'adreça Ethernet que li correspon. Cada màquina manté una cache amb les adreces traduïdes per a reduir el retard i la càrrega. ARP permet a l'adreça d'Internet ser independent de l'adreça Ethernet, per això només funciona si totes les màquines el suporten.

REVARP o RARP*: (Reverse Address Resolution Protocol).* **Protocol de resolució d'adreces invers.** És un protocol utilitzat per a resoldre l'adreça IP d'una adreça hardware donada (com una adreça Ethernet).

Paquet: Qualsevol bloc de dades enviat a través d'una xarxa. La unitat de dades que s'envia a per una xarxa la qual es composa d'un conjunt de bits que viatgen junts. En Internet la informació transmesa es divideix en trossos més petit, paquets, que es reagrupen una vegada arribats a destí.

Datagrama: És un fragment de paquet que es enviat amb la suficient informació com per a que la xarxa pugui encaminar-lo cap a l'equip terminal receptor de dades, de forma independent a la resta de fragments. Això garanteix que els paquets arribin amb l'ordre adequat o que tots arribin a la seva destinació. Els datagrames tenen la seva cabuda dins els serveis de xarxa no orientats a connexió.

BIBLIOGRAFIA

Llenguatge C++. Eclipse_cdt v4.03. Programari lliure i complementari per als estudis d'Informàtica, Multimèdia i Telecomunicacions. **UOC**

UML. Magic Draw v16.0. Programari lliure i complementari per als estudis d'Informàtica, Multimèdia i Telecomunicacions. **UOC**

Luis Joyanes Aguilar y Ignacio Zahonero Martínez (2001). *PROGRAMACIÓN EN C. Metodología, algoritmos y estructura de datos*. **Madrid: McGraw-Hill**/Interamericana de España, S.A.U

Alejandro López Monge (2005). *Aprendiendo a programar con Libpcap.*
kodemonk@emasterminds.net

http://www.tcpdump.org

http://www.arrakis.es/~terron/tcpdump.html

http://www.wireshark.org

http://www.wikipedia.com

http://www.cplusplus.com

ANNEX

Apèndix A: Primitives de filtrat TCPDUMP

BPF té el seu propi llenguatge de programació de filtres, un llenguatge de baix nivell semblant a l'ensamblador. De manera que **Libpcap** implementa un llenguatge molt més amigable per definir els seus filtres. Aquest llenguatge lògicament ha de ser compilat a **BPF** compatible abans d'ésser aplicat.

A continuació es detalla como programar filtres amb el llenguatge d'alt nivell desenvolupat per Libpcap i popularitzat per TCPDUMP que a dia d'avui s'ha convertit en l'estàndard per definir filtres de captura.

L'expressió que s'utilitza per a definir el filtre té una sèrie de primitives i tres possibles modificacions a les mateixes. Aquesta expressió pot ser *verdadera*, en aquest cas el paquet passa a zona d'usuari o *falsa*, i el paquet es descarta sense arribar a sortir en cap cas de la zona Kernel.

Les tres modificacions possibles són:

tipo Pot ser *host*, *net* o *port*. Indicant respectivament: màquina (per exemple **host 192.168.1.1**), una xarxa completa (per exemple **net 192.168**) o un port concret (per exemple **port 22**). Per defecte s'assumeix el **tipo host**.

dir Especifica des d'on o cap on es mirarà el flux de dades. Tindrem *src* o *dst* i podrem combinar-los amb *or* i *and*. Per al cas de protocols punt a punt podem substituir per *inbound* o *outbound*. Per exemple, si volem l'adreça de destí 10.10.10.2 i l'origen 192.168.1.2, el filtre serà **dst 10.10.10.2 and src 192.168.1.2.** Si es vol que l'adreça de destí sigui 192.168.1.1 i la d'origen 192.168.1.2 el codi serà **dst 192.168.1.1 or src 192.168.1.2.** Poden seguir-se combinant amb l'ajuda de parèntesis o les paraules **or** i **and**. Si no existeix se suposa **src or dst**.

proto En aquest cas és el protocol que volem capturar, pot ser **tcp**, **udp**, **ip**, **ether** (aquest últim cas captura trames a nivell d'enllaç), **arp** (peticions arp), **rarp** (peticions reverse-arp), **fddi** per xarxes FDDI.

Aquestes expressions sempre poden combinar-se amb l'ajuda de parèntesis i operadors lògics, Negació (!not), Concatenació (&& and), Alternativa (|| or).

A continuació es detallen les primitives que poden utilitzar-se. El que apareix entre [ ] és opcional, i el |(pipe) significa XOR (.o"exclusiva)

[dst | **src]** host *<màquina>*

Vertader si l'adreça destí u origen del paquet coincideix amb *<màquina>,* la qual pot ser una adreça IPv4 (o IPv6 si s'ha compilat suport adient), o un nom resoluble per DNS

Exemples:

*Capturar el trànsit originat per la IP 192.168.1.1
src host 192.168.1.1

*Capturar tot el trànsit on l'adreça origen o destí sigui 192.168.1.2
host 192.168.1.2

ether src | dst | host <edir>
Aquest filtre es cert si l'adreça origen (src), i la de destí (dst) o el de qualsevol de les dos (host) coincideix amb edir. És obligatori especificar un dels tres modificadors.

Exemples:

*Capturar el trànsit destinat a 0:2:a5:ee:ec:10
ether dst 0:2:a5:ee:ec:10

*Capturar el trànsit on l'adreça origen o destí sigui 0:2:a5:ee:ec:10
ether host 0:2:a5:ee:ec:10

gateway *<màquina>*
Vertader en cas de que el paquet utilitzi *<màquina>* com gateway. *<màquina>* ha d'estar definida *en /etc/ethers i etc/hosts.* Els paquets capturats per aquest tipus de filtre, son aquells l'adreça Ethernet de destí és *<màquina>*, però l'adreça IP destí es la d'un altre sistema.

[dst | src] net *<xarxa>*
Vertader en cas de que la xarxa de l'adreça destí, origen o ambdues sigui *<xarxa>.* El paràmetre xarxa pot ser una adreça numèrica (per exemple 192.168.1.0) o bé un nom resoluble mitjançant /etc/networks. Ressaltar que l'ús de net, mask, etc, no és compatible amb adreces **IPv6**. Si es vol fer referència a la xarxa destí, s'utilitza **dst** com prefix. Para la xarxa origen, s'utilitza **src**.

Exemples:

*Capturar tot el trànsit on la xarxa destí sigui 192.168.1.0
dst net 192.168.1.0

*Capturar tot el trànsit on la xarxa origen sigui 192.168.1.0/28
src net 192.168.10 msk 255.255.255.240 o també src net 192.168.1.0/28

*Capturar tot el trànsit amb origen o destí en la 10.0.0.0/24
net 10.0.0.0/24 o també not 10.0.0.0 mask 255.255.255.0

[dst | src] port *<port>*

El filtre capturarà el paquet en cas de que el port (udp o tcp) coincideixi amb <port>. El port és un valor numèric entre 0-6553 o bé a través del */etc/services*

Exemples:

*Capturar tot el trànsit amb destí al port 23
dst port 23

*Capturar tot el trànsit amb origen o destí al port 80
port 80

less <longitud>
Vertader en cas de que el tamany del paquet sigui menor o igual a <longitud>

greater <longitud>
Vertader en cas de que el tamany del paquet sigui més gran o igual a <longitud>

ip proto protocol
En aquest cas escolta el protocol que s'indiqui. El protocol pot ser *icmp, icmp6, igmp* (internet group managent protocol), *igrp* (interior gateway routing protocol), *pim* (protocol independen multicast), *ah* (IP Authentication header), esp (encapsuling segurity payload), *udp* o *tcp*.
Per comoditat disposem dels àlies tcp, udp i icmp que equivalen a ip proto tcp or ip6 proto tcp, etc...

Exemples:

*Capturar tot el paquet icmp
ip proto icmp

*Capturar tot el trànsit udp
ip proto udp

ip6 protochain <protocol>
El <protocol> és el numero de protocol que es pot trobar en /etc/protocols

ip protochain protocol
Igual que el cas anterior però per IPv4

ether broadcast
Vertader si la trama capturada va dirigida cap a l'adreça de difusió ethenet.

ip broadcast
Vertader si el paquet va dirigit a l'adreça de difusió de IP.
ether multicast

Vertader si la trama va dirigida cap a l'adreça multicast ethenet.

ip multicast
Vertader si el paquet va dirigit a l'adreça multicast IP.

ip6 multicast
Vertader si el paquet va dirigit a l'adreça multicast IPv6.

ether proto protocol
Vertader si el protocol que conté la trama és de tipus <protocol>. Els protocols són: ip, ip6, arp, rarp, atalk, aarp, decnet, sca, lat, mopdl, moprc i iso.

Exemples:

*Capturar tot el trànsit *arp*
ether proto arp

*Capturar tot el trànsit *ip*
ether proto op

vlan [vlanid]
Vertader si la trama capturada és un paquet 802.1Q VLAN. S'ha de tenir en compte que això cambia la resta de la interpretació del paquet capturat, en especial els desplaçaments a partir dels quals es comencen a descodificar els protocols, ja que s'assumeix que s'estan capturant paquets que viatgen en trames VLAN. Per últim si està present el paràmetre "vlanid", solament es mostrarà aquells paquets que vagin a la VLAN vlanid.

Combinant filtres
Es poden combinar les expressions anteriors mitjançant els operadors not, and, or , i els equivalents del llenguatge C (!, && , ||), fent així filtres més complexos.

Exemples:

*Capturar tot el trànsit web (TCP port 80)
tcp and port 80

*Capturar totes les peticions DNS
udp and dts port 53

*Capturar el trànsit al port telnet o ssh
tcp and (port 22 or port 23)

*Capturar tot el trànsit excepte el web
tcp and not port 80

Filtres Avançats

Podem crear filtres manualment, consultant els continguts de cadascun dels octets d'un paquet, l'expressió general per a crear un filtre avançat és:
expr relop expr

On:
relop pot ser qualsevol dels operadors relacionals de C (¿, ¡, ¿=, ¡=, = i !=)
expr és una expressió aritmètica composta per una sèrie de nombres enters, els operadors binaris de C, (+, -, *, /, &, i |), un operador de longitud, "len" i una sèrie de paraules reservades que ens permeten l'accés als diferents paquets de dades (ether, fddi, tr, ip, arp, rarp, tcp, udp, icmp, ip6).

Per accedir a les dades dins d'un paquet, s'utilitzen els modificadors anteriors i una expressió sencera.
Opcionalment es pot especificar el tamany de les dades a les quals s'accedeix.

protocol [expr:tam]

Així per exemple, el primer byte de la trama ethernet serà ether[0], la primera paraula serà ether [0:2]. El paràmetre "tam" pot ser 1 (per defecte), 2 o 4.

Cal tenir en compte: En cas d'utilitzar tcp[índice] o upd[índice], implicitament s'aplica una regla per esbrinar si és un paquet fragmentat, és a dir, utilitzant la notació d'aquestos filtres ip[0:2]& 0x1fff=0, udp[0] o tpc[0] es refereixen al primer byte de la capçalera UDP o TCP.

Apèndix B: MUTEX

Exclusió mútua, (comunament abreujada com *mutex* per *mutual exclusion*), es una expressió utilitzada en programació concurrent que fa referència al fet d'evitar l'accés simultani de dos fragments de codi a un recurs compartit (per exemple una cua, un comptador, etc). Així, aquests fragments de codi (seccions crítiques) s'han *excloure mútuament* per no provocar inconsistències en les dades que estan actualitzant.

En general, és necessari disposar de mecanismes adequats per garantir accés exclusiu en sistemes multitasca on diversos fils d'execució poden intentar accedir a un recurs al mateix temps, o també en recursos que poden accedits en context d'interrupció de la CPU.

La major part d'aquests recursos són els senyals, comptadors, cues i altres dades que s'empren en la comunicació entre el codi que s'executa quan es dóna servei a una interrupció i el codi que s'executa la resta del temps. Es tracta d'un problema de vital importància perquè, si no es prenen les precaucions degudes, una interrupció pot passar entre dues instruccions qualsevol del codi normal i això pot provocar greus errors.

La tècnica que s'utilitza normalment per aconseguir l'exclusió mútua és inhabilitar les interrupcions durant el conjunt d'instruccions més petit que impedirà la corrupció de l'estructura compartida (la secció crítica). Això impedeix que el codi de la interrupció s'executi al mig de la secció crítica.

En un sistema multiprocessador de memòria compartida, es fa servir l'operació indivisible test-and-set sobre una bandera, per esperar fins que l'altre processador la rebuig. L'operació test-and-set sempre realitza les dues operacions sense alliberar el bus de memòria a un altre processador. Així, quan el codi deixa la secció crítica, s'aclareix la bandera. Això es coneix com spin lock o espera activa.

Alguns sistemes tenen instruccions multioperació indivisibles similars a les anteriorment descrites per manipular les llistes enllaçades que s'utilitzen per les cues d'esdeveniments i altres estructures de dades que els sistemes operatius fan servir comunament.

La majoria dels mètodes d'exclusió mútua clàssics intenten reduir la latència i espera activa mitjançant les cues i canvis de context. Alguns investigadors afirmen que les proves indiquen que aquests algorismes especials perden més temps del que estalvien.

Malgrat tot el que s'ha dit, moltes tècniques d'exclusió mútua tenen efectes col laterals. Per exemple, els semàfors permeten interbloquejos (deadlocks) en què un procés obté un semàfor, un altre procés obté el semàfor i tots dos es queden a l'espera que l'altre procés alliberi el semàfor. Altres efectes comuns inclouen la *inanició* , en el qual un procés essencial no s'executa durant el temps desitjat, i la *inversió de prioritats* , en el qual una

tasca de prioritat elevada espera per altra tasca de menor prioritat, així com la *latència alta* en què la resposta a les interrupcions no és immediata.

La major part de la investigació actual en aquest camp, pretén eliminar els efectes anteriorment descrits. Si bé no hi ha un esquema perfecte conegut, hi ha un interessant esquema no clàssic d'enviament de missatges entre fragments de codi que, tot i que permet inversions de prioritat i produeix una major latència, impedeix els interbloquejos.

Els algorismes d'exclusió mútua que s'usen en programació concurrent per evitar l'ús simultani de recursos comuns, com a variables globals, per fragments de codi coneguts com a seccions crítiques.

Una forma habitual de garantir exclusió mútua en sistemes uni-processador es deshabilitar les interrupcions a l'entrada de la secció crítica i habilitar-les a la sortida. Aquesta solució, permet que mentre s'accedeix al recurs no hi hagi canvis de context deguts a una interrupció. En sistemes multi-processador de memòria compartida (per exemple SMP) s'utilitza una operació atòmica anomenada test-and-set, que permet comprovar l'estat d'una adreça de memòria i actualitzar el seu valor en un sol pas (sense alliberar el bus de memòria). Així, el processador que vulgui accedir al recurs compartit entrarà en espera activa fins que processador que en aquell moment utilitza el recurs l'alliberi i re-inicii el valor d'una variable (aquesta tècnica s'anomena spinlock).

La majoria de les solucions estan encaminades maximitzar l'eficiència en l'ús del processador, evitant en la mesura del possible efectes no desitjats tals com l'interbloqueig, la inanició de recursos o la inversió de prioritat.

Apèndix C: THREADS

En informàtica, un fil d'execució (thread en anglès) és una característica dels sistemes operatius que permet a un procés executar diferents tasques al mateix temps. Cada fil té un procés que ha de ser executat. Aquesta característica dona la possibilitat al programador de dissenyar un programa que executi diferents funcions concurrentment.

La tècnica de programació amb fils d'execució s'anomena multifil (Multithreading en anglès) i permet simplificar el disseny d'aplicacions concurrents i millorar el rendiment de la creació de processos. Cadascun dels fils accedeix a unes dades, quan una d'aquestes es utilitzada per dos o més fils es diu que la dada està en conflicte. Cada fil té una secció a on s'accedeix a aquestes variables, la qual s'anomena secció crítica. Tots el fils s'executen concurrentment. Aquesta característica és possible gracies al canvis de context.

Els canvis de context es produeixen quan un fil que està al processador és eliminat i entra un altre. Fent canvis de context molt ràpidament donem la sensació de què totes els fils s'executen simultàniament.

En els sistemes multifil, un mateix procés pot estar format per múltiples fils d'execució. Els diferents fils que formen part d'un mateix procés, comparteixen certs recursos com l'espai de memòria, els arxius oberts, els permisos, etc. En canvi, cada fil consta de les seves pròpies instruccions, la seva pròpia pila d'execució, s'executen a diferents velocitats (cada fil té el seu propi registre PC) i tenen el seu propi estat d'execució.

Els fils d'execució, també són coneguts com a processos lleugers. L'origen del nom rau en el fet que els fils d'execució consumeixen menys recursos de sistema que els processos.

La majoria de llenguatges de programació moderns, disposen de llibreries específiques per tal de programar amb fils i altres com C o C++ han d'utilitzar les crides de sistema que donen aquest suport.

Un parell d'exemples típics on s'utilitzen fils són:

- Aplicacions gràfiques: Un fil s'encarrega de la interfície gràfica d'usuari mentrestant un altre realitza les operacions.
- Aplicacions client/servidor: el servidor crea múltiples fils per tal de donar servei a múltiples clients alhora.

En sistemes POSIX hi ha 2 llibreries per a treballar amb fils d'execució:

- Native POSIX Thread Library per a Linux
- POSIX Threads standard