Analitzador del trafic de Xarxa per entorn Linux
“ Sniff All ”

Marc Berbel Querol
Enginyeria Tècnica en Informàtica de Sistemes

Consultor: David Carrera Pérez

Resum

En aquesta memòria es pretén tractar tots els punts del disseny d'un analitzador de xarxa (més conegut com sniffer) des de les seves primeres fases (recollida i documentació de requisits) fins a la seva implementació, tenint en compte totes les eines (llibreries, tutorials, col·laboracions, ...) que s'han utilitzat en cada un dels processos, així com la justificació de les decisions de disseny.

També es tractarà en profunditat els temes col·laterals al disseny d'un software amb interfície gràfica, com temes relacionats amb la eina CASE seleccionada (en aquest cas serà el Glade), així com de la llibreria en la que aquesta eina ens escriu el codi, que en aquest cas serà la llibreria GTK (Gimp Tool Kit). Un altre eina a que s'estudia en aquesta memòria és la llibreria que s'ha utilitzat per la captura i tractament de paquets que van per la xarxa, que és la "pcap" (de packet capture), llibreria la qual ens proporciona potents crides per manipular tant les targes de xarxa i posar-les en mode promiscu com per la captura i extracció de la informació dels paquets de xarxa.

Per últim, però en absolut menys important o decisiu, està el tema de les dues llibreries que m'han permès implementar el meu projecte com un software en que s'utilitzi el multi-threading. Una de les llibreries és la "pthread", que ens brinda totes les utilitats per implementar threads segons les normes POSIX, que són les utilitzades als sistemes operatius Linux. L'altra és la llibreria "gthread-2.0", que ens permet que la llibreria "glib", sobre la qual funciona la llibreria GTK, sigui el que se'n diu "multi-thread safe", és a dir, segura en la utilització de tècniques de multi-threading.

Index

1. Introducció

1.1 Objectius

Els objectius d'aquest Treball de Fi de Carrera són clars en quant a funcionalitat: es tracta de fer un analitzador de xarxa (vulgarment conegut com a sniffer) que funcioni en entorns Linux i amb interfície d'usuari gràfica.

He batejat l'sniffer amb el nom de "Sniff All", ja que com es veurà més endavant podem escollir la interfície de xarxa a analitzar, o podem analitzar tot el tràfic de xarxa que passa pel nostre ordinador a través de la interfície "all".

Bàsicament es tracta d'un senzill analitzador de xarxa, al que podem aplicar filtres depenent de diversos paràmetres dels protocols de xarxa (aquest tema es tractarà detalladament més endavant) i que està fet per treballar amb la pila de protocols TCP / IP.

El punt de ha partida ha estat zero. Tot i que hi ha diversos sniffers per TCP/IP ja implementats amb llicència GNU i dels quals es pot obtenir el codi font (com el tcpdump, el ethereal o d'altres), he volgut fer aquest projecte partint de zero, ja que era una oportunitat per construir un programa d'una forma integral, cosa que no és gaire habitual avui en dia.

1.2 Enfocament i mètode seguit

L'enfocament ha estat el de crear un sniffer que bàsicament funcionés d'una manera més que acceptable, ja que avui en dia hi ha una gran quantitat d'sniffers al mercat que son programes molt madurs (com el ehtereal) i amb moltes versions a les seves esquenes. Per tant és una mica il·lògic intentar fer un sniffer millor que aquests i partint de zero.

Donat aquests fets, li he donat un enfocament purament didàctic, i he preferit premiar la senzillesa de codi i el fet que sigui fàcilment comprensible per davant de sofisticades funcionalitats copiades d'algun altre codi font disponible per Internet.

El mètode seguit ha estat el de fer servir biblioteques especialitzades per cada una de les seves funcions, i com a llenguatge de programació, el C. En el cas de la llibreria per a l'entorn gràfic, he fet servir una eina CASE (Computer Aided Software Engineering) que per sota fa servir la llibreria GTK, que al seu torn es basa en la llibreria "Glib".

Respecte a les funcions de xarxa, he fet servir la llibreria anomenada “pcap” (de packet capture), utilitzada en sniffers coneguts com el tcpdump, i que inclòs ha estat portada a Windows (sota el nom de WinPcap), gràcies a lo qual el tcpdump també ha pogut portar-se a Windows sota el nom de WinDump. Pels temes de multi-threading, he utilitzat les llibreries “pthread” (que ens dona possibilitats de threading per a qualsevol programa que hàgim d’escriure en C), i “gthread” (especialitzada en el threading de programes que utilitzin “glib” o GTK).

Aquestes dues llibreries es distribueixen sota llicència GNU i j fa temps que van veure la llum, de manera que estem parlant de llibreries bastant madures en el que a robustesa i fiabilitat respecta.

1.3 Planificació del projecte

La planificació d’aquest TFC, es manté de moment fidel a la planificació que es va lliurar al seu dia, i que era la següent:

Setmana	Dates	Activitat	Esdeveniment
1	1 - 7 març	Escollir projecte Confecció P.T.	
2	8 - 14 març	Recollida i documentació dels requisits del sniffer	Lliurament Pla Treball
3 i 4	15 - 28 març	Consulta del material necessari per repassar tota la base teòrica de les xarxes TCP / IP	
5, 6 i 7	29 març - 18-abr	Disseny de l'interfície d'usuari, així com de les diferents pantalles, començar a redactar la memòria del projecte	Lliurament PAC2
8, 9, 10 i 11	19 abril - 16 maig	Implementació de les funcionalitats del soft, continuar amb la redacció de la memòria	
12 i 13	17 - 30 maig	Proves exhaustives en diferents entorns de xarxa, descripció de les proves a la memòria del projecte	Lliurament PAC3
14	31 maig - 6 juny	Enllestir memòria i proves finals del programa, preparar presentació virtual	
15 i 16	7 - 20 juny	Preparar presentació virtual	Lliurament TFC

Evidentment, aquesta planificació pot patir desviacions en funció dels punts on es trobin dificultats imprevistes (que segur que s'hi trobaran) i en funció del temps que es trigui a trobat una solució adient per aquests problemes. De vegades aquesta solució pot implicar desfer alguna de les decisions que ja s'havien pres.

Degut a això, a l'hora de fer la planificació vaig mirar de deixar prou temps de marge.

1.4 Desviació entre la planificació i la realitat

A l'hora de la veritat, s'han produït certes diferències entre el que estava planificat i la realitat del desenvolupament del projecte. Hi ha que tenir en compte, tal i com he dit abans, que es tracta del primer programa que faig que comença a tenir unes certes dimensions, i llavors es fa difícil fer una planificació exacte per endavant.

Bàsicament m'he endarrerit a dos punts concrets:

- El primer punt ha estat a la fase del projecte que s'havia de dur a terme les setmanes 8, 9, 10 i 11, que correspon a la implementació de les funcionalitats de soft. Com ja explico més endavant, vaig decidir fer el meu projecte amb una tècnica de multi-threading, cosa que en un principi no havia previst inicialment. En realitat això no va provocar un endarreriment en la temporització, sinó que el que vaig fer és augmentar el nombre d'hores setmanals que tenia previst en un principi dedicar-li al projecte.
- El segon punt ha estat la fase del projecte que s'havia e portar a terme la setmana 14 del projecte. Resulta que el programa que vaig utilitzar per fer el disseny de la interfície gràfica, el Glade, peca de poca portabilitat. Quan li vaig fer el lliurament de la PAC3 al meu consultor, resulta que el projecte, que a mi em funcionava realment bé, a ell no li funcionava, ni tan sols el executable compilat. En un principi vaig decidir no amoïnar-m'hi, ja que el consultor havia vist funcionar el projecte a la trobada de síntesi, on vaig portar un portàtil amb el projecte funcionant, però després em va poder el cuc de lliurar un projecte que al meu consultor li funcionés, així que em vaig baixar una Red Hat 9 (distribució que em va comentar que utilitzava), me la vaig instal·lar, i vaig fer la migració del meu projecte a la Red Hat. El meu projecte, originàriament havia estat desenvolupada a una Mandrake 9.2, i vaig necessitar unes horetes per polir tots aquells punts en els que se m'encallava el compilador a la Red Hat.

En tots dos punts, per no desviar-me de la planificació, vaig optar per prémer una mica l'accelerador i dedicar-li més hores, ja que com he comentat abans em vaig deixar unes quantes hores a la recambra per si la planificació no havia estat gaire realista.

De totes maneres, no m'he quedat gaire decebut de la manera com ha anat el projecte, ja que la planificació inicial només ha comptat amb la presència d'aquests dos imponderables, i crec que en tot projecte moltes vegades es deuen donar un parell de diferències entre la planificació i el que passa en la realitat.

1.5 Producte obtingut

El producte obtingut serà una sèrie de fitxers, que una vegada compilats amb el make quedaran en un sol arxiu executable el qual arrencarà el menú principal de l'sniffer. Els arxius seran els típics que produeix l'entorn Glade, com son el "callbacks.c" i el mateix en versió capçalera (.h), o el "support.c" i el seu homònim en versió capçalera.

L'estructura del producte serà una carpeta, dins la qual hi hauran una sèrie de fitxers i de directoris, que passo a enumerar junt amb una breu explicació:

Directori General: /TFCv3

A aquest directori no hi ha cap arxiu que jo hagi fet pel projecte, tots els arxius que hi ha els crea el Glade. Únicament vaig haver de modificar l'arxiu "configure.in" per tal d'afegir la línia:

```
LIBS="$LIBS –lpcap –lpthread –lgthread-2.0"
```

Aquesta línia la vaig afegir per tal de fer que a la llista de llibreries que s'havien d'incloure a la compilació del programa afegís la llibreria de captura de paquets i les dues de threads que es necessiten pel projecte, una bàsica a nivell de codi (la lpthread) i l'altre a nivell gràfic (que és la lgthread-2.0).

Directori macros: /TFCv3/macros

En aquest directori no hi ha res que jo hagi fet, tot el que hi ha és afegit pel Glade.

Directori src: /TFCv3/src

En aquest directori és sobre el que jo he treballat bàsicament. Hi ha quatre arxius de C amb els seus corresponents fitxers de capçalera .h.

El primer és el callbacks.c, on es troba principalment el motor del sniffer, on hi ha les principals crides a la llibreria pcap i on es mostren per pantalla les informacions rellevants. També és en aquest fitxer on crea i es finalitza el thread que s'encarrega de la captura dels paquets.

El segon és el interface.c, on el Glade deixa tot el codi utilitzat per la interface gràfica, com la creació de finestres, botons, quadres d'entrada de texte... En aquest arxiu no es pot tocar res, ja que cada vegada que afegeixes quelcom, el Glade el sobrescriu.

El tercer és el main.c, de funcionalitat òbvies. Fa la crida per la creació de la finestra principal i després entra en un bucle gtk-main() que és el que controla tot el que passa a la interface gràfica. Aquest arxiu es pot editar pel programador, i de fet ho he hagut de fer pels temes dels threads gràfics. Per més referència, veure els comentaris que he posat al codi.

El quart és el support.c, fitxer en el qual posaríem les funcions de suport en cas de que haguéssim habilitat tal funció (cosa que jo no he fet, ja que no ho he de fer servir). També hi ha funcions per crear pixmaps i altres funcions gràfiques.

Els altres arxius han estat creats pel glade, o son fitxer objecte ".o" resultat de la compilació del projecte.

Per compilar el projecte, ens hem de posar al directori principal i fer un "make", el qual ens crearà el fitxer executable "tfcv3", i ens el deixarà a la carpeta "src".

2. Decisions prèvies

2.1 Llenguatge de programació

A l'hora de veure quin llenguatge de programació faria servir en el desenvolupament del meu treball de fi de carrera, no vaig dubtar que havia de ser en C, ja que em semblava el llenguatge més adient donat que per fer un sniffer s'ha de tenir control a molt baix nivell.

També vaig tenir en consideració el fer el TFC en C++, però donada la meva poca experiència en aquest llenguatge la veig desestimar gairebé d'immediat.

Evidentment, a l'hora de considerar el llenguatge de programació a utilitzar, havia de tenir en compte que el TFC seria el desenvolupament més gran de software que mai havia fet, ja que jo no treballo com a programador, i els programes que habitualment havia fet eren petits clients de correu per altres assignatures o petits "scripts" per temes laborals. Així doncs, el C semblava la opció més coherent, tot i que el tema dels punters sempre hi ha que repassar-ho, ja que als dos mesos de no programar en C sempre se'n oblida un de com funcionen.

Respecte al sistema operatiu, quedava clar pels requisits del TFC que havia de ser en una plataforma Linux, tot i que a la trobada de principi de semestre es va deixar la porta oberta a poder-ho fer sota Windows. De totes maneres, tenia clar que el volia desenvolupar en Linux per dos motius:

- El suport que dona Linux a C és gairebé total, de fet molts dels mòduls de aquest sistema operatiu estan escrits en C
- D'altra banda, era una bona oportunitat per tal de conèixer millor aquest entorn que cada vegada s'està estandarditzant més i que cada dia compta amb més usuaris

D'altra banda, per a la realització del sniffer, havia començat a documentar-me en pàgines com les del tcpdump o les del ethereal (dos sniffers que son freeware sota llicència GNU, i bastant coneguts), i vaig veure que tots dos softwares utilitzaven una llibreria d'utilitats per a la captura de paquets, anomenada "libpcap" (packet capture), i tal llibreria estava escrita originàriament en C, tot i que hi ha migracions per altres entorns, inclòs per a Windows.

2.2 Entorn gràfic

En quant a l'entorn gràfic, havia d'utilitzar una eina per al desenvolupament de les diferent finestres que formen l'aplicació (que de fet no son gaires), en fi, per al desenvolupament de l'entorn gràfic.

Els entorns gràfics tinguts en compte varen ser dos, el Qt (de la casa trolltech – http://www.trolltech.com - , però que té una versió gratuïta) i el Glade (freeware sota llicència GNU). Vaig estar mirant els dos, i em vaig instal·lar els dos al meu PC, on corre una Mandrake 9.2, però llavors vaig veure que el Qt estava orientat únicament a C++, mentre que el Glade podia generar codi C, C++ i Eiffel.

Així doncs, la eina CASE quedava clar que seria Glade. Em vaig posar a jugar amb aquesta eina i a fer les primeres proves i programets, i vaig buscar informació a Internet sobre el desenvolupament d'aplicacions amb aquesta eina.

Bàsicament vaig trobar dos petits manuals (cap d'ells arriba a les 10 pàgines, i tenen molts screenshots):

- Manual de Glade (de Marcelo Pérez Medel)
- Desarrollo de aplicaciones científicas con Glade (Fco. Domínguez-Adame)

Però haig de dir que aquests manuals em van servir per escriure el típic programa de "Hello World" i poca cosa més.

Així doncs, buscant per l'ajuda del Glade (que és una mica més completa que els dos manuals citats anteriorment, però que no té cap versió per imprimir...), i per la seva pàgina web (http://glade.gnome.org) vaig veure que aquest editor genera codi basat en una llibreria anomenada GTK (que vé de Gimp Tool Kit) (http://www.gtk.org). Em vaig baixar el reference guide de la llibreria GTK+ (que imprès ve a ocupar unes 150 pàgines) i em vaig mirar una mica la manera de treballar que té aquesta llibreria, ja que en certs aspectes veia clar que hauria de fer coses que el Glade no em suportaria directament. Finalment em vaig haver d'apuntar a una "mail list" de glade, ja que hi havia temes que em presentaven punts foscos, com per exemple, com refrescar la pantalla a petició del programa, i no només a l'entrada i sortida de les rutines.

Una de les incomoditats de treballar amb una eina CASE, és que et genera una certa estructura de producte, és a dir, una jerarquia de carpetes i fitxers que has de respectar, ja que si no, quan et posis al directori pare i executis un make, no et compilarà i et donarà errors per tot arreu.

2.3 La captura de paquets

La gran decisió, pel que respecte al "motor" de l'aplicació, que no deixa de ser un sniffer de paquets que circulen per la xarxa, era veure de quina manera es podien capturar els paquets i manipular-los, com poder esbrinar la seva procedència i altres dades, com poder posar la tarja a treballar en mode promiscu, etc...

Al principi de tot, quan vaig buscar informació sobre el tema, vaig tenir la sort de haver el CCNA de Cisco i de tenir clar el concepte de xarxa ethernet i de TCP/IP, ja que si no, al buscar per temes com "xarxa", o "captura de paquets" o altres similars, la quantitat d'informació que trobes és tan enorme (i la gran majoria no era gens interessant pel

desenvolupament del projecte), que has de tenir molt clar què et pot interessar i què no (en resum, hi ha que saber molt bé què es busca).

Així que el que vaig fer es anar a veure com havien resolt aquest tema altres sniffers com ara podria ser el tcpdump, que treballa per línia de comandes, o el ethereal, aquest ja amb un bonic entorn gràfic. I llavors va ser quan vaig descobrir el que se'n diu "llibreria pcap" (libpcap), que és una llibreria que va ser dissenyada per la captura i tractament de paquets que circulen per la xarxa. He de dir que m'he quedat gratament sorprès per la potència d'aquesta llibreria i per la fiabilitat de les seves crides, tot i que moltes vegades es passa un número considerable de paràmetres en les crides, i molts són punters, de manera que quan es descontrola l'ús d'un d'ells per un error de programació, ràpidament entra en joc el famós error "segmentation fault" (en el qual m'he tornat tot un expert).

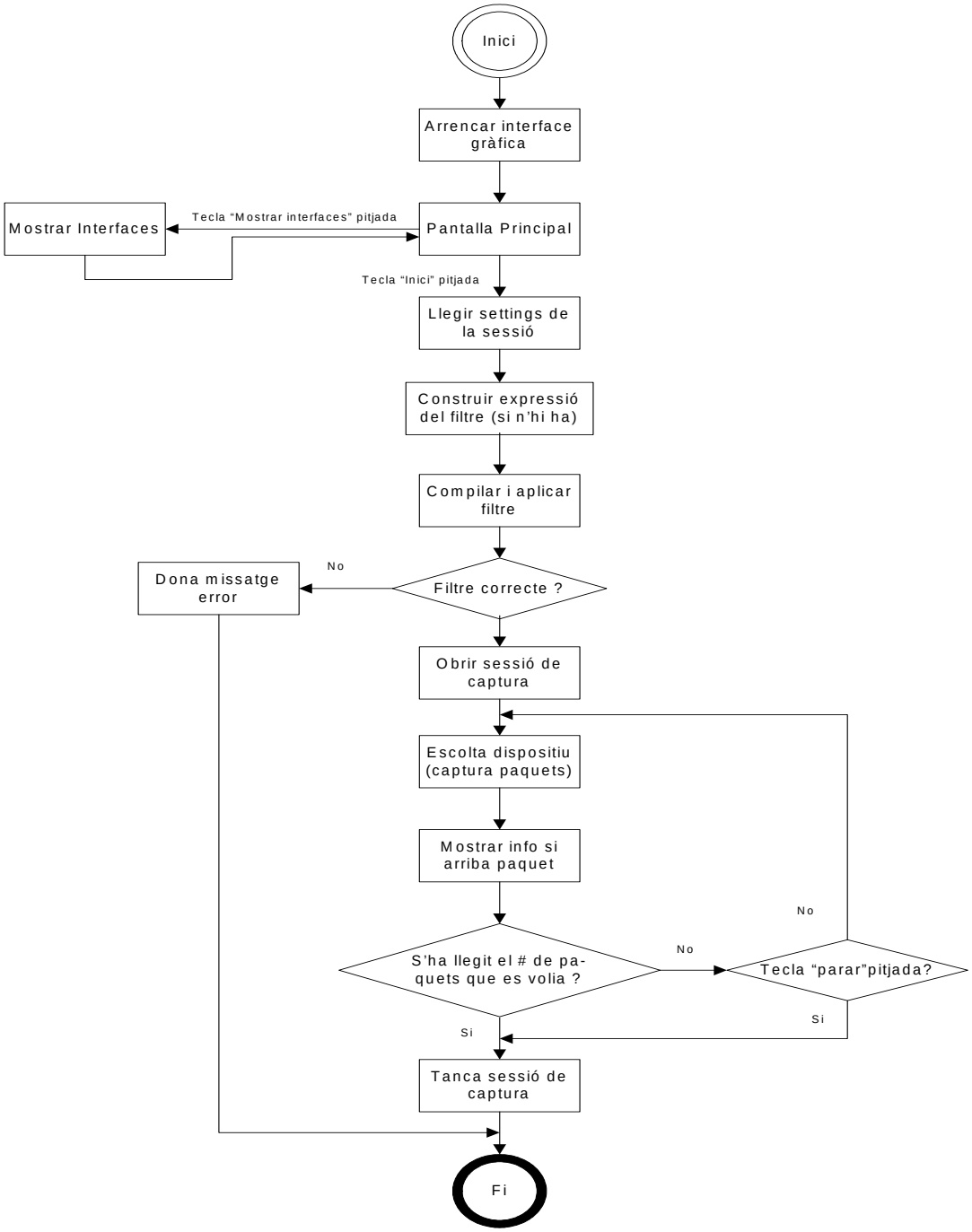
Així doncs, un cop feta la troballa d'aquesta llibreria hi havia que documentar-se, ja que com he dit abans el seu ús és còmode, però en absolut és trivial (un error en una de les seves crides et pot fer para boig). Per això vaig fer servir la documentació sobre la "libpcap" que vaig trobar a la pàgina web del tcpdump (http://www.tcpdump.org), així com a la pàgina d'en Tim Carstens, tota una autoritat a l'hora de fer filigranes amb aquesta llibreria.

D'altra banda, també va ser útil apuntar-me al mailing list del tcpdump, ja que et van arribant mails amb dubtes d'altra gent, però que et serveixen per veure quins tipus d'aplicacions s'estan construint amb aquesta llibreria i amb quina estructura de producte ho estan fent, de manera que sempre et poden donar idees.

3. Realització del projecte

3.1 Esquema de funcionament intern

Pel que respecte al funcionament del programa, crec que és interessant veure un diagrama de com funciona per tal d'entendre algunes de les decisions de disseny que durant la realització del mateix s'han pres. Anem a veure a nivell intern com funciona el "Sniff All":



És evident que durant la fase d'implementació del projecte s'han hagut de canviar certes coses que hauria volgut fer i que vaig preveure a la fase de recollida de requisits i d'especificació, però el resultat final crec que ha és un bon resultat de compromís entre la funcionalitat que un analitzador de xarxa ha de tenir i la que idealment m'hauria agradat que tingués el meu.

Pel que respecta a aspectes del funcionament intern, i del que passa exactament a cadascun d'aquests passos, s'anirà explicant en les successives seccions.

3.2 Llibreria Pcap, el motor de l'Sniff All

Com ja he comentat en diverses ocasions al llarg d'aquesta memòria, el motor d'aquest projecte en el que captura de paquets es refereix és una llibreria anomenada "pcap" (de packet capture).

Originàriament aquesta llibreria es va desenvolupar paral·lelament al conegut software tcpdump (www.tcpdump.org) , i a dia d'avui està a la versió 0.8.3, la qual es pot considerar totalment estable. La llibreria es va pensar originàriament per treballar en C, però avui en dia existeixen versions per C++, Windows, Java, ...

Bàsicament, la manera que té de treballar la llibreria pcap és molt senzilla, es pot dir que unes deu línies de codi es podria fer un sniffer (evidentment molt senzillet). El tema ja s'allarga una mica més si volem afegir opcions de filtrat, és a dir, que només capturi els paquets que venen de un adreça IP determinada, o que vagin apuntats cap a un port determinat, o que siguin d'una adreça ethernet determinada (i tots aquests settings, poden ser d'origen o de destí...).

Anem a veure esquemàticament com funciona, i després farem una repassada a les crides:

PRIMER DE TOT, DECIDIR QUINA INTERFACE VOLEM ESCOLTAR

Efectivament, el primer que hem de fer és decidir quina interface de xarxa volem escoltar. Per saber quines interfaces hi ha al nostre sistema Linux, hi ha una instrucció molt senzilleta que podem executar per línia de comandes:

```
ifconfig
```

Veurem que a tot sistema Linux, generalment, hi ha dos interfaces lògiques (que no físiques): la **lo**, que és la interface de loopback, aquella que s'encarrega de contestar quan li fem un ping a l'adreça 127.0.0.1, i la **any**, que és una interface per on passa

qualsevol paquet que arribi a qualsevol interface de xarxa, és a dir, és com el conjunt de totes les interfaces. Si tenim, com en el meu cas, una tarja de xarxa ethernet, el més normal és que també ens aparegui una interface de xarxa anomenada **eth0**, que serà el nom que per defecte li posa el Linux.
A la llibreria pcap hi ha una comanda que ens retorna totes les interfaces de xarxa que es troben al sistema, que és la que jo faig servir quan es pitja el botó "mostrar interfaces" del Sniff All. La crida té la següent signatura:

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

La crida ens retorna 0 si tot ha anat bé, i en cas contrari ens retorna –1. En aquest últim cas, trobarem un missatge d'error que podrem consultar al paràmetre **errbuf.**

El punter **alldevsp** apunta cap al primer element a la llista de interfaces de xarxa que ens retorna aquesta funció. Cada element d'aquesta llista te els següent subelements:

next: que si no és NULL, apunta cap a la següent interface de xarxa de la llista
name: el nom que té la interface al sistema
description: que pot estar buit o tenir una breu descripció de la interface
addresses: un punter al primer element d'una estructura on hi ha les adreces de xarxa d'aquesta interface
flags: conté un indicador que ens diu si la interface és de loopback o no

Doncs bé, ja hem vist dues maneres de determinar les interfaces de xarxa que hi ha al nostre sistema. Ara hem d'escollir una.

UN COP ESCOLLIDA UNA INTERFACE, DESCOBRIM LES SEVES ADRECES

Un cop hem escollit una interface de xarxa per escoltar, el següent és saber l'adreça de xarxa i la màscara de subxarxa, paràmetres necessaris per poder obrir una sessió de captura.

Per saber aquests paràmetres sobre una interface concreta, podem executar la següent crida de la llibreria pcap:

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

La crida ens retorna 0 si tot ha anat bé, i en cas contrari ens retorna –1. En aquest últim cas, trobarem un missatge d'error que podrem consultar al paràmetre **errbuf.**

El primer paràmetre (**device**) ja es veu que és on hem de dir la interface de la qual volem esbrinar les adreces. Pel segon paràmetre (**netp**) és per on la crida ens retorna l'adreça de xarxa i la màscara de subxarxa ens la retorna pel tercer paràmetre (**maskp**).

Ara ja tenim una interface escollida i tenim les dades que necessitem d'aquesta interface.

OBTENIM UN DESCRIPTOR PER LA NOSTRA SESSIO DE CAPTURA DE PAQUETS

El següent que hi ha que fer, és obtenir un descriptor per a la nostra sessió de captura, cosa que s'ha de fer amb la següent crida:

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

La crida ens retorna un descriptor (del tipus pcap_t) si tot ha anat bé, i en cas contrari ens retorna NULL. En aquest últim cas, trobarem un missatge d'error que podrem consultar al paràmetre **`errbuf.`**

Hi ha altres crides de l'estil "open", que s'utilitzen per obrir captures que han sigut guardades a fitxer, o altres tipus d'operacions, que ja veurem més endavant.

El primer dels paràmetres que li hem de passar a aquesta crida és la interface a escoltar (**`device`**). El segon (**`snaplen`**) és la longitud del fragment del paquet que hen de capturar. Per exemple, si posem 500, només tindrem disponibles els primers 500 bytes del paquet (tamany suficient per capturar les capçaleres desitjades, però petit si volem capturar i mostrar les dades de paquets grans, se'ns quedaria petit). El tercer paràmetre (**`promisc`**) ens serveix per especificar si volem fer la captura en mode promiscu o no. Si posem un 0, no estem en mode promiscu, i amb qualsevol altre valor, si. El quart paràmetere (**`to_ms`**) és el time-out en milisegons.

Sobre el timeout: a l'hora d'arrencar una sessió de captura ho podem fer de tres maneres diferents (com ja veurem més endavant). En dues d'aquestes maneres es consulta aquest paràmetre. Ja ho veurem més endavant.

COMPILEM EL FILTRE

El següent que hi hauria que fer, en cas de que anéssim a filtrar els paquets rebuts, seria compilar l'expressió de filtre. Aquesta expressió no deixa de ser una string, amb una sintaxi determinada, del estil que se'n BPF (Berkley Packet Filter). Donat a la importància que tenen en el mon dels sniffers aquest tipus de filtres, vaig a fer una mica de tutorial dels filtres BPF. Anem a veure per sobre el format que han de tenir les expressions de filtre:

L'expressió que s'utilitza per a definir el filtre te una sèrie de primitives y tres possibles modificadors de aquestes. Aquesta expressió serà vertadera o falsa y farà que s'imprimeixi o no el paquet de que ha arribat.

Els 3 modificadors possibles son:

- tipus. Pot ser host, net o port. Indiquen respectivament una maquina, per exemple host 192.168.1.1 , una xarxa completa, por exemple net 192.168, o un port concreto, per exemple port 22. Per defecte s'assumeix el tipus host.
- dir. Especifica des de o cap a on han de venir els paquets per passar el filtre. Tenim src o dst i podem combinar-los amb or i and. Pel 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 la d'origen 192.168.1.2, el filtro serà dst 10.10.10.2 and src 192.168.1.2 . Si es vol que sigui l'adreça destí 192.168.1.1 o l'adreça origen 192.168.1.2, serà dst 192.168.1.1 or src 192.168.1.2. Es poden seguir combinant amb l'ajut de parèntesis o les paraules or i and. Si no existeix es suposa src or dst. Evidentment, això es pot combinar amb els modificadors de tipus anteriors.

- proto. En aquest cas és el protocol que volem capturar. Pot ser tcp,udp,ip,ether (en aquest caso captura trames a nivell d'enllaç, arp (peticions arp), rarp (petcions reverse-arp), fddi (per a xarxes FDDI, però realment l'encapsulat és igual al ether). Hi ha altres nivells d'enllaç per xarxes Decnet i lat, però no entraré a comentar-los, ja que no es fan servir ni de lluny a aquest projecte.

I les possibles primitives són:

- [dst|src] host màquina. Cert si l'adreça destí o origen del paquet es **màquina,** que pot ser una adreça IPv4 (o IPv6 si s'ha compilat el seu suport), o un nom del DNS. Opcionalment, si volem restringir a l'adreça de destí podem posar **dst**, i a la d'origen, amb **src**.
- ether src|dst|host MACdir. Aquest filtre és cert si l'adreça destí (**dst**) o origen (**src**), o qualsevol de les dues (**host**) coincideix amb l'adreça MAC que li passem. Fixem-nos que hi ha que posar una de les tres (**src**, **dst** o **oct)**.
- [dst|src] net xarxa. Aquest filtre deixa passar el paquet en cas de que la xarxa de l'adreça de destí, origen o totes dues sigui l'especificada. El paràmetre **xarxa** pot ser una adreça numèrica (p.e. 192.168.1.0) o bé un nom que se resolgui a adreça, en Linux, con l'ajut del fitxer **/etc/networks**.
- [dst|src] port port. Cert en cas de que el port (ja sigui udp o tcp) coincideixi amb **port**. Si no s'especifica **dst** o **src**, serà cert tant si es tracta del port origen com destí. Si volem restringir a destí usem **dst** i a origen usem **src**. El port és un valor numèric entre 0-65535 o bé un nom que en Linux es resolgui a través del fitxer */etc/services*.

Amb la mica de tutorial que hem fet, ja podem posar uns petits exemples que il·lustrin com fer certs filtres de paquets:

Exemples:

- Capturar el tràfic amb destí a l'adreça ethernet 0:2:a5:ee:ec:10.
  - ether dst 0:2:a5:ee:ec:10
- Capturar el tràfic amb origen / destí a la màquina amb l'adreça MAC 0:2:a5:ee:ec:10.
  - ether host 0:2:a5:ee:ec:10
- Capturar el trafico la IP origen del qual sigui 192.168.1.1
  - src host 192.168.1.1
- Capturar tot el tràfic d'origen / destí de la IP 192.168.1.2
  - host 192.168.1.2
- Capturar tot el tràfic que vagi destinat al port 23
  - dst port 23
- Capturar tot el tràfic que vagi o vingui del port 80
  - port 80

Un cop vist això, ja és molt fàcil de veure que amb les expressions “and” o “or” es podem combinar filtres per “pescar” només aquells paquets com compleixin unes característiques molt determinades:

- Capturar tot el tràfic que surti des de l’estació amb IP 192.168.10.2 i que surti del port 21 ó 23
  - src host 192.168.10.2 and (src port 21 or src port 23)

I com aquest es podrien posar exemples que, amb l’ajut de parèntesis i de “and” i “or”, formarien llargues expressions de filtre.

Així doncs, ara ja sabem quin format ha de seguir l’expressió del filtre que volem aplicar. Ara el que hi ha que fer, és compilar aquest filtre per poder-lo aplicar. Ho faríem amb la següent crida a la llibreria pcap:

```
int pcap_compile(pcap_t *p, struct bpf_program *fp,
char *str, int optimize, bpf_u_int32 netmask)
```

En aquesta crida, el primer paràmetre és a quin descriptor de sessió volem aplicar, el filtre, el segon (**`struct bpf_program *fp`**) és una estructura que ens omplirà aquesta funció en la que tindrem el filtre ja compilat, el tercer paràmetre (**`char *str`**) és la cadena de caràcters que ha d’estar muntada amb la nomenclatura que he explicat abans, el quart paràmetre (**`int optimize`**) ens serveix per indicar si volem que la funció miri d’optimitzar l’expressió de filtre, i a l’últim paràmetre (**`bpf_u_int32 netmask`**) hem de posar-li la màscara de subxarxa del dispositiu al que volem aplicar el filtre.

APLIQUEM EL FILTRE

El següent que hi hauria que fer, un cop compilat el filtre, seria aplicar-lo, ja que compilant el filtre, l’únic que hem fet és convertir el filtre a un format entenible per la llibreria pcap, i el deixem llest per poder-lo aplicar quan volem.

Per aplicar el filtre, es fa amb la funció:

```
int pcap_setfilter(pcap_t *p, struct bpf_program *fp)
```

on els dos paràmetres que hi ha que posar són evidents a aquestes alçades del manual.

Doncs bé, a aquestes alçades ja estem en disposició de comentar la captura de paquets.

COMENÇA LA CAPTURA

Per la captura de paquets tenim diferents crides, i la diferència entre elles és subtil. Comencem pel principi; de les tres crides possibles que hi ha per la captura de paquets, hi ha una part que és comuna, i és la funció de callback. En totes tres crides hi ha un paràmetre que és el nom de la funció que es crida cada vegada que arriba un paquet. Aquesta funció de “callback” (o sigui, funció que s’executa cada vegada que passa

algun event determinat, en aquest cas l'arribada d'un paquet per la xarxa) ha de tenir el següent format:

```
void paquet_arribat(u_char *args, const struct
pcap_pkthdr *header, const u_char *packet)
```

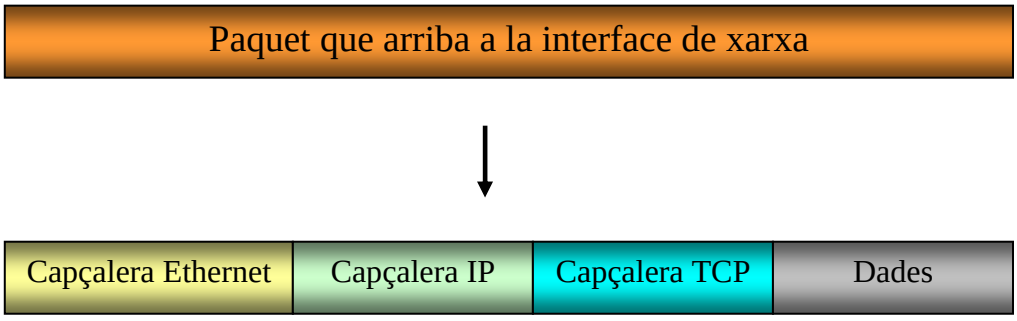
I els tres argument que té aquesta funció són molt simples; el primer és un punter que el podem utilitzar per passar un argument, el segon és una estructura que s'omple cada vegada que arriba un paquet, i que conté la informació següent:

```
Struct pcap_pkthdr {
    struct timeval ts;    /* time stamp */
    bpf_u_int32 caplen;   /* porció present */
    bpf_u_int32 len;      /* longitud real del paquet */
};
```

Per últim, el tercer argument és un punter que apunta cap a la cadena on es troba tota la informació del paquet, una cosa darrera de l'altre. Hi ha que dir que aquesta "botifarra de dades" s'agafa directament del nivell d'enllaç de dades, és adir, que la llibreria "pcap" utilitza els fitxers "ether.h", "ip.h" i "tcp.h" de la carpeta /usr/include/netinet/ (o sigui, d'una de les carpetes de sistema) per extreure les estructures de dades amb les que funciona el nostre sistema.

Per això, en el meu projecte, hi ha un define una mica "anormal" al principi del "callbacks.c", que fa referència a definir el BSD_SOURCE. Amb aquest define el que s'està fent és definir el tipus de capçalera TCP que farem servir de les dues que hi ha definides al meu sistema al fitxer "tcp.h".

La manera que tenen d'arribar-nos les dades també fa que de s'hagi de trossejar el que ens arriba, ja que com ja he dit abans, tot el paquet que la llibreria pcap ha capturat, ens arribat com un string de dades. Per tant, en el cas de que estiguem escoltant un dispositiu que sigui ethernet (com pugui ser "eth0"), hem de saber fins a on arriba la capçalera ethernet, després fins a on arriba la capçalera IP i després fins a on arriba la capçalera TCP.



Bàsicament, el que fa la nostra funció de “callback” és això, agafar l’string de dades, i posar cada cosa dins l’estructura de dades que li correspon, i a més a més, en el meu cas treure per pantalla a la finestra corresponent aquesta informació.

Però com ja he dit abans, la funció de “callback” és una part comuna de les tres possibles maneres que te la llibre “pcap” de capturar paquets. En total hi ha tres crides diferents:

```
int pcap_dispatch(pcap_t *p, int cnt, pcap_handler
callback, u_char *user)
```

on el primer paràmetre (**`pcap_t *p`**) és el descriptor de sessió, el segon (**`int cnt`**) és el màxim nombre de paquets a capturar abans de que la funció retorni, el tercer paràmetre (**`pcap_handler callback`**) és la nostra funció de callback, i que ha de tenir el format específic que he explicat abans, i el quart paràmetre ens el deixa per si li volem passar paràmetres a la funció de callback. La funció ens retorna un enter amb el nombre de paquets capturats (un valor de -1 indica que hi ha hagut un error).

La segona de les possibles crides és la següent:

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler
callback, u_char *user)
```

i tot el que s’ha explicat per **`pcap_dispatch`** és aplicable al paràmtres d’aquesta crida.

Aixó doncs, quina és la diferència entre utilitza la primera i la segona crida ?

Doncs que la primera crida (**`pcap_dispatch`**) fa una ullada a un paràmetre que abans he mencionat, però que no he volgut comentar, i que és el paràmetre de “temps en mil·lisegons” que té la crida **`pcap_open_live`**. Si utilitzem **`pcap_dispatch`**, forcem a que si ha passat els mil·lisegons que hem especificat, i no s’ha rebut cap paquet, es retorni igualment de la funció de callback, amb un valor de retorn de zero, encara que no hagi arribat cap paquet. O sigui, que se al obrir el descriptor, li posem 200, cada 200 mil·lisegons retornarà de la funció. En canvi, a la crida **`pcap_loop`** , no es retorna fins que no s’hagi llegit el nombre de paquets que li hàgim especificat a **`cnt`**. Això ens pot provocar que el nostre programa es quedi “enganxat” en cas de que no arribi el nombre de paquets que esperàvem.

Així doncs, perquè no utilitzem sempre la crida **`pcap_dispatch`**? Doncs per que a la documentació que es pugui trobar de la llibreria pcap s’especifica que el tema dels temporitzadors no està ni de bon tros garantit, i que hi ha moltes plataformes en les que no funciona, com ha estat el meu cas. Així doncs, al meu programa he utilitzat a crida **`pcap_loop`**, que gasta menys recursos, i que de fet a mi no m’afectava que es quedés “enganxada”, ja que de fet, des de al poc temps de començar l’sniffer em vaig donar compte de que l’hauria de fer multi-threading.

I la tercera crida de la llibreria “pcap” que ens pot servir per la captura de paquets? Doncs és:

```
const u_char *pcap_next(pcap_t *p, struct pcap_pkthdr
*h)
```

però és una funció que en aquest cas ens serveix de poc, ja que només serveix per llegir els paquets d'un en un, o sigui que t'has de fer tu el bucle, i el seu valor de retorn és un punter cap al lloc on t'ha deixat l'string de dades. Els dos paràmetres que te aquesta funció són obvis.

Així que finalment, després d'haver capturat els paquets que volíem, ens queda:

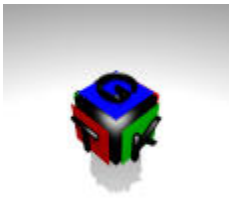
TANCAR LA SESSIÓ DE CAPTURA

És tan simple com utilitzar la crida:

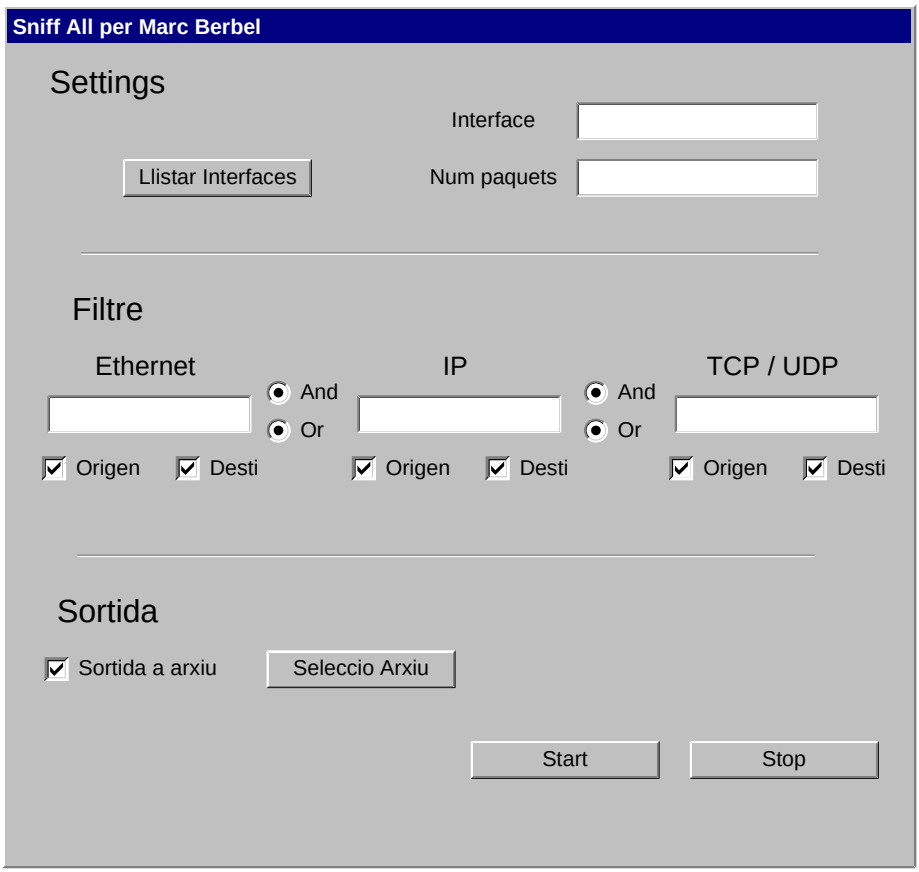
```
void pcap_close(pcap_t *p)
```

que no retorna res i que com a únic paràmetre el que necessita és el descriptor de captura que volem tancar, i alliberar així els recursos que la estructura de dades associada pugui consumir.

3.3 Llibreries Gràfiques



Dins de l'apartat de l'entorn gràfic, el primer que vull fer és el disseny de les pantalles principals de la meva aplicació. La idea inicial per la pantalla principal és quelcom com el que segueix:



Evidentment, la pantalla no tindrà un aspecte idèntic a aquest, ja que aquest disseny de pantalla va estar fet amb el MS Visio, i els colors i certes textures no sé si es podran reproduir amb el Glade. Com es pot veure, he primat la senzillesa i l'efectivitat, de manera que tot el que hi ha és el que es veu, no hi ha menús ocults ni desplegables a la capçalera de la finestra.

Apart d'aquesta finestra principal, hi haurà un model més de finestra, però que serà molt senzilla. Serà una finestra només per a sortida de text, a on s'aniran mostrant els paquets capturats, o les interfícies que hi han al sistema, depenent de si hem demanat una funcionalitat o una altra. Aquestes dues finestres aniran amb les seves corresponents barres de desplaçament vertical (i horitzontal en cas que sigui necessari), ja que si per exemple, demanem de capturar 1000 paquets, evidentment hi ha d'haver un mecanisme que després ens permeti desplaçar-nos per tota la captura.

Un altre model de finestra que tindrem, serà una finestra d'error, que ens sortirà bàsicament en un parell de circumstàncies:

- Quan hàgim demanat sortida a arxiu i no haguem especificat l'arxiu
- Quan demanem iniciar una sessió de captura i ja n'hi ha una executant-se

Respecte a l'eina utilitzada per la construcció de la interfície gràfica, el Glade, com ja he dit abans, ens genera codi de la llibreria GTK, la qual alhora és una extensió de Glib, per això hi ha un include que es:

```
#include <gtk/gtk.h>
```

que en realitat ja inclou també la llibreria Glib.

Un dels inconvenients que li he trobat a aquesta eina ha estat el fet de que no pots tocar res als arxius “interface” (ni al “.c” ni al “.h”) ni al “support” (tampoc a cap dels dos, però en aquest cas, no ens afecta), ja que cada vegada que afegeixes un widget, o sigui, un element gràfic, aquests arxius es rescriuen sense tenir en compte possibles modificacions que hagi pogut fer el programador.

Aquí, si ens donem compte, he utilitzat una paraula curiosa, el terme “widget”. Doncs bé, un widget és qualsevol element de l’entorn gràfic amb el que l’usuari pot interactuar, com ara una finestra, un botó, una caixa d’entrada de text, la caixa pel text de sortida i el text de sortida en sí mateix, etc ...

La llibreria GTK+ (que com ja he dit, inclou la llibreria Glib), entra en una mena de funció “ociosa” (que en el fons és un bucle infinit) que és la funció:

```
gtk_main();
```

que el que fa es que quan es pitja un botó o es produeix un event, llença un senyal. Molts dels events no els tindrem contemplats a la nostra aplicació, per tant no passarà res quan fem aquella acció en concret. Per exemple, en el cas de la meva aplicació no passa res quan es mou la finestra principal de lloc a l’escriptori, però podria haver fet que al moure la finestra sortís algun missatge o sonés un soroll d’error, etc ... En canvi, sí que tindrem contemplat l’event de quan pitgem el botó de començar la sessió de captura, i per lo tant tindrem definida una funció de callback per quan això ocorre. Per connectar un senyal concret amb una funció de callback, hem de fer el següent:

```
gtk_signal_connect (GTK_OBJECT (buto_selec_fitxer),
"clicked", GTK_SIGNAL_FUNC
(on_buto_selec_fitxer_clicked), NULL);
```

Amb aquesta instrucció, el que fem és capturar el senyal que es llença quan “cliquem” sobre el botó de selecció d’arxiu de sortida i connectar aquest senyal amb la funció “**`on_buto_selec_fitxer_clicked`**”, ja es pot veure que el tema no té gaire secret.

Si ens fixem en el fitxer “interface.c” veurem que primer hi ha tots els elements de la interfície gràfica (aquest codi hem de recordar que en realitat el crea el Glade) i que després tenim un munt de connectors de senyals, que capturen els events que ocorren i criden a la funció de callback adequada.

Però de tot, el més complicat ha estat poder fer que la interacció de la llibreria gràfica no em fes anar lent el cor del sniffer (o viceversa). És a dir, que en un principi, a la primera versió del projecte, la interacció no era gaire bona. Per veure el problema que vaig tenir de bon principi, hem de remuntar-nos al principi del projecte. El primer que vaig fer pel projecte va ser construir l’sniffer per funcionar per línia de comandes, de manera que només intervenia la llibreria “pcap” i les crides habituals a un programa de C. Fins aquí tot correcte. Però el problema va venir quan li vaig començar a afegir tota la part de interfície d’usuari, és a dir, tota la part gràfica. Si ens recordem de com funciona la llibreria pcap (veure apartat 3.2), veurem que quan comencem la captura

pròpiament dita, s'entra a un bucle del que no en sortim fins que s'han capturat tot els paquets que volíem. Això feia que el programa es quedés bloquejat aquí fins que s'havia acabat la captura, moment en el qual llavors el programa tornava a la funció ociosa "gtk_main()", on llavors es redibuixava tota la part de GUI. És a dir, que no es refrescava la pantalla fins que no s'acabava la captura, cosa que en el cas de que volguéssim deixar l'sniffer funcionant indefinidament era totalment inviable.

El primer que vaig fer va ser apuntar-me al mailing list del GTK, ja que vaig pensar que segur que hi devia haver alguna solució "fàcil" pel meu problema. Va ser llavors quan vaig descobrir una instrucció (que per cert, no venia comentada al tutorial del GTK que em vaig descarregar) que era:

```
gtk_main_iteration();
```

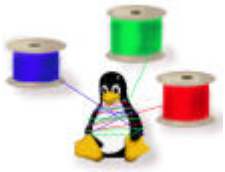
i que el que feia era fer només una passada pel bucle "gtk_main" sense quedar-s'hi indefinidament a l'espera de que es pitgessin nous botons o succeïssin nous events. Amb això, vaig aconseguir que cada vegada que arribés un paquet més o menys es redibuixés la interfície gràfica, però ho feia a mitges (de l'aplicació només redibuixava la finestra activa), i el resultat continuava sent insatisfactori pel que respecta a percepció de velocitat per part de l'usuari. D'altra banda, si per exemple, es desconnectava en una sessió de captura el cable de xarxa em trobava que havia de fer un "Ctrl-C" per acabar el programa (o un kill, si l'havia llançat des de les X). A aquesta fase del projecte correspon el segon lliurament que vaig fer de la mateixa.

En fi, ja no quedava res més que rendir-se a les evidències: havia de fer un programa que es bifurqués en dos processos (al més pur estil "fork") o si no fer un programa amb diversos fils d'execució (el que se'n diu multithread). La meva primera idea va ser fer-lo en dos processos, ja que un sempre tendeix al que coneix, i la tècnica del multi-threading no la coneixia pas (només coneixia el concepte teòric). Però vaig començar a documentar-me sobre el multi-threading, i vaig veure que pel tipus de programa que estava desenvolupant potser seria bo provar de fer-ho amb aquesta tècnica.

Aquí es va produir un punt d'inflexió al meu programa, ja que era una nova etapa de documentació, degut a que jo no havia treballat mai amb el tema dels threads, i no sabia quan temps em podia enrederir el tema d'aprendre i sobretot, reestructurar (o fins i tot reiniciar) el meu projecte.

D'altra banda, i com es veurà en el put següent, l'aplicació de tècniques de multi-threading no ha sigut només crear un thread i disparar-lo en el moment adequat, si no que hi ha calgut fer diverses operacions de threads amb la llibreria del GTK, ja que d'altra manera et pots tornar assidu al "segmentation fault".

3.4 El multi-threading



Evidentment, per l'existència d'aquest punt ja es pot suposar que l'implementació del multi-threading va ser possible i que no produir grans endarreriments, però tot i això va estar un tema clau del projecte.

Al començar a documentar-me sobre el multi-threading vaig veure que el meu sistema operatiu Linux, el Mandrake 9.2, portava una llibreria de threads, anomenada "pthreads". Així doncs que ja només quedava posar-se mans a la feina.

La diferència entre utilitzar dos threads i utilitzar dos processos son les següents:

- Un thread continua l'execució des de la funció que l'indiquem, mentre que si un procés nou és creat, torna a començar del principi, i has de tenir clar en tot moment qui és el pare i el fill
- Dos threads arrencats per un procés, sempre pertanyen a aquell procés, de manera que poden compartir fàcilment estructures de dades i altres variables sense haver d'establir un pont de comunicació entre els dos (com s'hauria de fer si fossin dos processos), com podria ser una pipe
- Això sí, degut a que comparteixen les variables globals (per exemple), s'ha de tenir molt en compte els diferents camins que pot agafar un thread, de manera que si dos threads accedeixen a la mateixa variable hi hagi algun mecanisme que impedeixi que hi hagin inconsistències a les dades. Poden ser mecanismes d'exclusió mutus (mutex – de mutual exclussion) o semàfors

Anem a veure amb una mica més de detall les instruccions que he utilitzat al meu projecte, la signatura de totes les quals es troba al fitxer del sistema operatiu "pthread.h":

DEFINICIÓ D'UNA VARIABLE PEL THREAD

El primer que he fet és definir una variable pel meu thread. Es podrà veure que només utilitzo un thread al meu projecte, però en el fos és com si utilitzés dos, ja que tenim el thread que s'arrenca en un cert moment, més el transcurs normal del programa, que seria l'altre thread. La variable per guardar l'identificador del thread que he utilitzat al projecte està declarada així:

```
pthread_t fil_sniffer;
```

El pas següent a fer seria:

CREEM EL THREAD

Crear el thread es fa amb la instrucció següent:

```
pthread_create(&fil_sniffer, NULL, comenca_captura,
NULL);
```

Com que en un parell de paràmetres hi tinc un NULL posat, vaig a posar la signatura original d'aquesta funció:

```
int pthread_create(pthread_t *thread, const
pthread_attr_t *attr,
  void *(*start_routine)(void*), void *arg);
```

Els paràmetres d'aquesta crida són els següents: el primer paràmetre és el nom que li donem al thread (que hem declarat com a variable prèviament) i que utilitzarem per identificar-lo al llarg del programa. El segon paràmetre es fa servir per passar una variable del tipus pthread_attr_t que pot contenir atributs del thread, com pot ser per exemple si és joinable o no (ja veurem més endavant què és això). Si el deixem en NULL, com en el nostre cas, es crearà un thread amb els atributs per defecte. El tercer paràmetre és la funció d'inici per on deu de començar el thread quan s'iniciï. El quart paràmetre és una variable per ús del programador. Com es pot veure, en el nostre cas també hi hem posat un NULL. El valor de retorn serà zero en el cas de que tot hagi anat bé, i un número d'error en el cas que hagi hagut algun problema amb la creació del thread.

Quan cridem a aquesta funció, hem de ser conscients que a partir d'aquest moment el sistema operatiu ja comença a executar en paral·lel el thread que acabem de llençar amb la continuació del fil principal d'execució.

FINALITZEM EL THREAD

Per últim, el que he fet al meu programa és finalitzar el thread amb la instrucció següent:

```
pthread_exit(NULL)
```

El paràmetre (que en el nostre cas no hem utilitzat) que hi pot anar entre els parèntesis és per diferents modes de sortir, però habitualment no s'utilitza.

Com es pot veure, no és gens complicat el joc d'instruccions que he fet servir per implementar el tema dels thread al meu programa. Però, evidentment, no tot ha estat això. Quan vaig redissenyar lleugerament la meva aplicació per que funcionés amb threads, vaig estudiar molt curosament on arrencaria i on moriria el thread que correria paral·lel a l'execució principal del programa. Com es pot veure si analitzem el programa, bàsicament el que he fet és crear el thread en el moment que comença la captura, mentre l'aplicació principal se'n va cap el bucle d'espera "gtk_main()", que va repintant contínuament, i quan s'acaba la captura, el thread es mor allà mateix.

Però no tot va ser tant senzill. La teoria l'havia repassat cinquanta vegades, i no semblava haver errors de codi, i en canvi, algunes poques vegades l'sniffer m'havia funcionat satisfactòriament, i la majoria m'havia acabat donant un "segmentation fault". Així que hem vaig haver de submergir en les pàgines que tractaven de threads que vaig poder trobar per Internet. I en una de elles, en un mailing-list, vaig veure que parlava sobre els problemes que pot donar la llibreria GTK quan s'utilitza amb un programa amb multi-threading, i de com fer que una implementació d'aquest estil fos "thread-safe", o sigui, a prova de threads.

Bàsicament el problema resideix en que una implementació "normal" de la llibreria GTK no és "thread-safe", de manera que hi ha que afegir una sèrie d'instruccions apart de les habituals per evitar els errors de segmentació que es produeixen quan accedim a una zona de memòria que teòricament no hauríem d'estar utilitzant. Aquests errors, bàsicament se'm produïen per que mentre s'anava pintant per pantalla les dades dels nous paquets que anaven arribant amb un thread, el camí principal d'execució del programa ja havia anat cap al "gtk_main()", una de les funcions del qual és anar repintant la pantalla. El problema de l'ús de threads és que és difícil controlar en quin moment el sistema decideix aparcar l'execució d'un thread per passar a l'execució de l'altre, i llavors, si dues instruccions més enrere del bucle "gtk_main()" (que per nosaltres és totalment transparent) havia comprovat que hi havia dues línies de text per pintar, i quan les anava a pintar veia que son tres línies, per que mentrestant hi ha hagut un canvi de thread i l'altre thread (el que captura els paquets) ha afegit nova informació per pintar, es produeix una falta de coherència de la informació, de manera que es llavors quan es produïa el temut "segmentation fault".

Així doncs, el que calia fer era que la implementació de la llibreria GTK fos thread-safe per evitar els problemes de segmentació. La veritat és que tot això es fa amb una llibreria complementària a la GTK o a la Glib, i és la llibreria "lgthread-2.0", i de fet, es pot observar al fitxer "configure.in" que hi ha al directori principal del meu projecte com he hagut d'afegir aquesta llibreria per que també s'especifiqui a l'hora de cridar al compilador. Aquesta llibreria (la lgthread-2.0) ens proporciona una sèrie de mecanismes per implementar amb ella mateixa el tema dels threads sense haver de fer servir la llibreria lpthread, però en el meu projecte m'he estimat més no fer-ho així.

Si ens fixem, al fitxer "main.c" del meu projecte, veurem que hi ha dos defines al principi de tot del fitxer, que són:

```
#define     G_THREADS_ENABLED
#define     G_THREADS_IMPL_POSIX
```

Aquests dos defines el que fan és dir-nos que s'habilita el suport per al multi-threading, i després especificar quin estil de multi-threading està implementat en aquest programa. En el meu cas, com es pot veure, la implementació ha estat del tipus POSIX, que és el tipus al que pertany la llibreria lpthread.

Més endavant ens trobem una instrucció que diu:

```
g_thread_init(NULL);
```

El que estem fent amb aquesta instrucció és indicar al sistema de supervisió de threads que ens dona la biblioteca lgthread-2.0 que s'arrenqui, és adir, que a partir d'aquest moment es pot trobar amb que el programa llenci l'execució d'un thread.

La següent instrucció que ens trobem de la lgthread és la següent:

```
gdk_threads_enter()
```

El que fem amb aquesta instrucció és dir-li al sistema de supervisió de threads que crei els mecanismes necessaris entre aquesta instrucció i una altra que comentarem més avall, ja que segur que hauran de córrer en paral·lel amb altres instruccions de la llibreria gràfica.

La instrucció que tanca el grup d'instruccions "supervisades" és la següent:

```
gdk_threads_leave()
```

De manera que el que fan aquestes dues instruccions, d'alguna manera, és garantir l'atomicitat de les instruccions que es troben entre les dues, sense que hi hagin problemes de segmentació al programa. Veiem que al programa main.c el grup d'instruccions és el següent:

```
gdk_threads_enter();
gtk_main ();
gdk_threads_leave();
```

o sigui que el que fem és garantir l'atomicitat del "gtk_main". Val a dir que aquesta atomicitat no es pot garantir amb les eines com semàfors o mutexes que ens ofereix la llibreria lpthread, ja que recordem que quan s'entra al gtk_main no se'n surt fins que una de les senyals que el gtk_main capti (com per exemple, pitjar un botó) ens porti cap a una funció on hi hagi la instrucció "gtk_main_quit". És per això que s'ha hagut d'implementar una llibreria pel tema dels threads gràfics, ja que en realitat no és un tema en absolut trivial.

En diferents punts del programa (veure arxiu "callbacks.c") anirem veient una successió d'instruccions "gdk_threads_enter ()" i "gdk_threads_leave ()", que estaran envoltant les instruccions pertanyents a la llibreria GTK que s'hagin d'executar en paral·lel amb altres threads (o amb el fil principal d'execució en aquest cas) que també continguin instruccions de la llibreria GTK.

4. Conclusions

Com a conclusió principal val a dir que ha estat més divertit del que em pensava fer el projecte. Com és natural, hom té un cert respecte per aquelles coses que no ha fet mai, i la veritat és que jo, fins a dia d'avui, havia fet molt poques coses amb interfície gràfica (em sembla que dos exemples de l'estil "Hello World").

També val a dir que si aquesta experiència hagués sigut una experiència professional, potser hauria pressupostat menys hores de les que en realitat hauria emprat després, ja que haig de dir que en el projecte he invertit més hores de les que teòricament s'haurien d'invertir en una assignatura d'aquestes característiques. Això sí, també és veritat que la majoria de les hores han estat per haver de buscar informació per Internet, i sobretot pel tema del threading i del threading gràfic, ja que era la primera vegada que utilitzava aquesta tècnica. M'ha servit per haver-me de forçar a apuntar-me a un parell de "mailing-lists" d'aquells que quan els veus per primera vegada no enganxes res de res. Però de tot aprèn un, i al final inclòs resulta que he ajudat a un parell dels del mailing-list amb coses que jo ja m'havia trobat.

M'ha quedat molt clar que fer una estimació d'hores realista d'un projecte, has de tenir molt clar què faràs i quines eines utilitzaràs, i dominar-les, ja que a primera vista tot sembla molt senzill i després et trobes amb els disgustos de coses que no funcionen com voldries i que s'han de prendre decisions de si es modifica el disseny o no.

En fi, com a conclusió, trobo que ha estat una experiència altament positiva, i estic molt satisfet d'haver agafat la part de l'sniffer, ja que el tema de les xarxes sempre m'ha agradat molt.

5. Glossari

BPF : Berkley Packet Filter, sintaxi per especificar filtres de xarxa
FDDI : (Fiber Distributed Data Interface) protocol d'enllaç de dades que corre sobre una capa física formada per dos anells de fibra òptica. A nivell de trama, és gairebé idèntic a Ethernet.
glade : eina de desenvolupament d'interfícies gràfiques d'usuari, que genera codi que utilitza la llibreria GTK+
glib : llibreria gràfica feta en C que forma la base de GTK+ i de GNOME
gthread-2.0 : veure lgthread-2.0
GTK/GTK+ : Gimp Tool Kit llibreria per crear interfícies gràfiques
IP : Internet Protocol, protocol de xarxa de capa 3 que ha esdevingut és l'estàndard d'Internet i de les xarxes Ethernet
lgthread-2.0 : llibreria de C que serveix per a la implementació de threads relacionats a programes on tenim instruccions de les llibreries glib / GTK
lpcap : llibreria de C "paquet capture", especialitzada en la captura de paquets
lpthread : llibreria de C existent en sistemes UNIX/Linux, que ens serveix per a la implementació de threads
pcap : veure lpcap
pixmap : mapa de píxels, se'n diu d'una imatge que s'utilitza per ser incrustada en una interfície gràfica
pthread : veure lpthread
TCP : Transmission Control Protocol, protocol de capa 4 que és l'estàndard d'Internet i de les xarxes Ethernet
thread : fil d'execució d'un programa que corre paral·lelament al fil principal d'execució o a altres threads
UDP : User Datagram Protocol, protocol de capa 4 utilitzat dintre de la pila de protocols TCP/IP

6. Bibliografia

Cisco Systems, Inc.
Guía del primer año – *Pearson Educación, S.A.*, Madrid (2002)

Cisco Systems, Inc.
Guía del segundo año – *Pearson Educación, S.A.*, Madrid (2002)

Ian Main, Toni Gale
GTK Tutorial – *Tutorial publicado en Internet*, (2003)

Marcelo Pérez Medel
Manual de Glade – *Tutorial publicado en Internet*, (2001)

Francisco Domínguez-Adame
Desarrollo de aplicaciones científicas con Glade – *Tutorial pub. en Internet*, (2003)

Tim Carstens
Programming with pcap – *http://broker.dhs.org*, (2004)

The TCPDump Project
Pcap Man Pages – *http://www.tcpdump.org*, (2004)

Varios
Posix threads Programming – *http://www.lnll.org/tutorials/workshops*, (2004)