TREBALL DE FI DE CARRERA

TÍTOL: Sistema de llicència lliure d'arrencada remota

AUTOR: Arnau Güell Soler

DIRECTOR: Cristina Barrado Muxi

DATA: 3 d'octubre del 2006

Títol: Sistema de llicència lliure d'arrencada remota

Autor: Arnau Güell Soler

Director: Cristina Barrado Muxi

Data: 3 d'octubre del 2006

Resum

Amb l'expansió de l'informàtica des dels anys 80 a tots els entorns de la nostra societat (gracies a la sortida al mercat dels PCs), i la creació de llocs de treball interconectats entre si, van començar a sorgir problemes en l'administració d'aquests. Les desconfiguracions dels entorns de treball, el mal ús dels equips informàtics, els virus, van fer aparèixer la necessitat de crear copies de seguretat de cada equip per a la possible restauració en cas fallida o pèrdua d'informació.

Amb el creixement i la instauració paulatina en tots els entorns de xarxes locals per a compartir l'informació i optimitzar els recursos, van sorgir propostes des de l'entorn privat per a automatitzar les copies de seguretat i la restauració en cada equip d'aquestes copies. El gran incovenient d'aquestes propostes va ser el seu obscurantisme en la divulgació del codi emprat, a més a més del preu de configuració del sistema per a cada entorn en particular i el pagament de llicencies periòdicament per la utilització d'aquest sistemes per part de l'usuari.

No obstant, l'avenç que van fer aquestes empreses per l'automatització dels sistemes de restauració d'imatges i el fet de treure al mercat un producte vàlid per a qualsevol entorn sense importar ni el tipus de xarxa, ni el tipus d'equips informàtics (hardware dels PCs), ni el sistema operatiu utilitzat, van fer obrir un gran ventall de possibilitats i van marcar gran part del objectius d'aquest projecte.

Sota aquesta demanda s'ha creat un sistema de restauració d'imatges de sistemes operatius remots en software lliure. Així creant un sistema de restauració viable i sense cost addicional.

Aquesta solució que combina diversos elements del software lliure desenvolupat fins ara fa servir un servidor linux com a base del sistema i amb molts pocs recursos es capaç de satisfer les demandes dels usuaris.

Title: Free license system for remot booting

Author: Arnau Güell Soler

Director: Cristina Barrado Muxi

Date: October, 3rd 2006

Overview

With the expansion of the computer science from the 80s to all the environments of our society (thanks to the exit to the market of the PCs), and the creation of interconnected working places between itself, problems started arising in the administration from these. The desconfigurations of the environments of work, the bad use of the computer equipments, the virus, they made appear the need to create safety copies of every equipment for the possible restoration in case of break down or loss of information.

With the growth and the gradual restoration in all the environments of local nets to share the information and to optimize the resources, they arose proposed from the private environment to automate the safety copies and the restoration in every equipment of these copies. The great disadvantage of these offers was his obscurantism in the publication of the used code, also the price of configuration of the system for each environment and the payment of licenses dayly for the utilization of these systems .

However, the advance that these companies did for the automation of the systems of restoration of images and the fact of extracting to the market a valid product for any environment without to import either the type of net, or the type computer equipment (either hardware of the PCs), or the used operative system, they made open a great fan of possibilities and marked great part of the objectives of this project.

Under this demand a system of restoration image of remote operative systems has been created in free software. This way creating a restoration system viable without aftercost.

This solution that combines several elements of the free software developed until now, using linux server as base of the system and with few resources it is capable of satisfying the demands of the users.

ÍNDEX

INTRODUCCIÓ 1

CAPÍTOL 1. ELS SISTEMES OPERATIUS 3

1.1. Els serveis que proporciona un SO 4
1.1.1. Les crides de sistema 4
1.1.2. L'intèrpret de comandes 4

1.2. El nucli del SO 5

1.3. Comparativa dels SO existents 7

CAPÍTOL 2. EL SISTEMA D'ARRANC 9

2.1. Arrencada del sistema (Linux) 9
2.2.1. 1ª etapa del carregador d'arranc 10
2.2.2. 2ª etapa del carregador d'arranc 11
2.2.3. El nucli 11

2.3. Sistemes d'arranc múltiples 12
2.3.1. LILO 12
2.3.2. GNU GRUB 13
2.3.3. Conclusió 14

2.4. Arranc per xarxa 14
2.4.1. Procés d'arrencada 15
2.4.2. Serveis relacionats amb PXE 16

CAPÍTOL 3. NFS 21

3.1. El sistema de fitxers 21

3.2. Sistemes de fitxers distribuïts 22

3.3. NFS 22
3.3.1. Característiques de NFS 22
3.3.1. Protocols de NFS 23
3.3.3. Implementació de NFS 26

CAPÍTOL 4. KEXEC 29

4.1. Introducció a Kexec 29

4.2. Utilitzant kexec 30

4.3. Les operacions de kexec 31

4.4. Beneficis de kexec 32

CAPÍTOL 5. AUTOMATITZACIO I ÚS DEL SISTEMA DE RESTAURACIO REMOT 33

5.1. Sistema de restauració remot 33
5.1.1. El servidor 33
5.1.2. Transmissió i creació d'imatges 36
5.1.3. Configuració de GNU GRUB 39
5.1.4. Arrencada del nucli restaurat 39

5.2. Automàtització 40
5.2.1. Restaurar Ubuntu 40
5.2.2. Restaurar Windows 40
5.2.3. Arrencar Windows/Ubuntu 41

CAPÍTOL 6. PROVES EN EL PROCÉS DE RESTAURACIÓ 45

6.1. PXE 45

6.2. Creació i restauració d'imatges 46

CONCLUSIONS 49

BIBLIOGRAFIA 51

INTRODUCCIÓ

Amb l'expansió de l'informàtica des dels anys 80 a tots els entorns de la nostra societat (gracies a la sortida al mercat dels PCs), i la creació de llocs de treball interconectats entre si, van començar a sorgir problemes en l'administració d'aquests. Les desconfiguracions dels entorns de treball, el mal ús dels equips informàtics, els virus, van fer aparèixer la necessitat de crear copies de seguretat de cada equip per a la possible restauració en cas fallida o pèrdua d'informació.

Amb el creixement i la instauració paulatina en tots els entorns de xarxes locals per a compartir l'informació i optimitzar els recursos, van sorgir propostes des de l'entorn privat per a automatitzar les copies de seguretat i la restauració en cada equip d'aquestes copies. El gran incovenient d'aquestes propostes va ser el seu obscurantisme en la divulgació del codi emprat, a més a més del preu de configuració del sistema per a cada entorn en particular i el pagament de llicencies periòdicament per la utilització d'aquest sistemes per part de l'usuari.

No obstant, l'avenç que van fer aquestes empreses per l'automatització dels sistemes de restauració d'imatges i el fet de treure al mercat un producte vàlid per a qualsevol entorn sense importar ni el tipus de xarxa, ni el tipus d'equips informàtics (hardware dels PCs), ni el sistema operatiu utilitzat, van fer obrir un gran ventall de possibilitats i van marcar gran part del objectius d'aquest projecte. En resum els objectius d'aquest projecte són els següents:

- Crear un sistema de restauració remot d'imatges de sistemes operatius en programari lliure. Utilitzant eines ja existents en el programari o codi creat la conjunció dels quals ens permeti arribar al nostre objectiu final.

- Crear un sistema vàlid per a una xarxa heterogènia. On no importin ni el tipus d'equips ni els sistemes operatius emprats per cadascuna de les màquines.

- Crear un sistema eficient que pugui competir amb el programari privat actual.

- Crear des de l'inici fins al final del procés de restauracióun sistema consistent .

Per a assolir aquests objectius hem de tindre en compte molts factors que s'anirà explicant més extensament durant la lectura de la memòria. Per això la memòria s'ha estructurat de la forma següent:

- CAPÍTOL 1 ELS SITEMES OPERATIUS: Entendre el funcionament i funcionalitats d'un sistema operatiu es imperant per a poder tenir un concepte clar de la importància d'aquests en els nostres equips informàtics. I de la problemàtica que planteja el mal funcionament d'aquests. A més de facilitar-nos la comprensió de certs conceptes

durant la lectura de la memòria. També s'entendrà perquè s'ha decidit utilitzar linux davant altres opcions.

- CAPÍTOL 2 EL SISTEMA D'ARRANC: Per a poder dur a terme aquest projecte el primer que ens hem de preguntar és ¿ Com arranca un PC? Entendre el procés d'arranc dels sistemes operatius és de vital importància si es vol crear un sistema automatitzat per a la restauració d'imatges de sistema. A més d'ajudar-nos a comprendre els processos que intervenen durant el procés de restauració, com el procés d'arranc per xarxa.

- CAPÍTOL 3 Network File System: NFS és un sistema de fitxers distribuïts que ens ajudarà a centralitzar la informació i divulgació d'aquesta. Així tenint un paper clau en la possibilitat de transmetre les imatges dels diversos sistemes operatius d'un PC a un altre.

- CAPÍTOL 4 KEXEC: Com s'ha comentat al principi el nostre referent en aquest projecte es el programari privat existent per a la creació i restauració d'imatges de sistemes operatius, més en concret REMBO. Aquest programa no té la necessitat de reiniciar la màquina quan es descarrega en aquesta un nova imatge per a poder inciar-la. Kexec ens ajudarà a poder aconseguir aquesta tasca.

- CAPÍTOL 5 AUTOMATITZACIÓ: En aquest capítol s'explicarà la configuració de les eines utilitzades per a poder aconseguir el nostre fi. I el codi creat per a l'automatització del procés de restauració.

- CAPÍTOL 6 PROVES I RESULTATS: En aquest capítol s'explicaran les proves realitzades d'eficiència de les eines emprades i els resultats esperats. Així com la visualització real del funcionament dels protocols utilitzats.

CAPÍTOL 1. ELS SISTEMES OPERATIUS

En la figura següent es mostra un esquema general del sistema d'un computador:

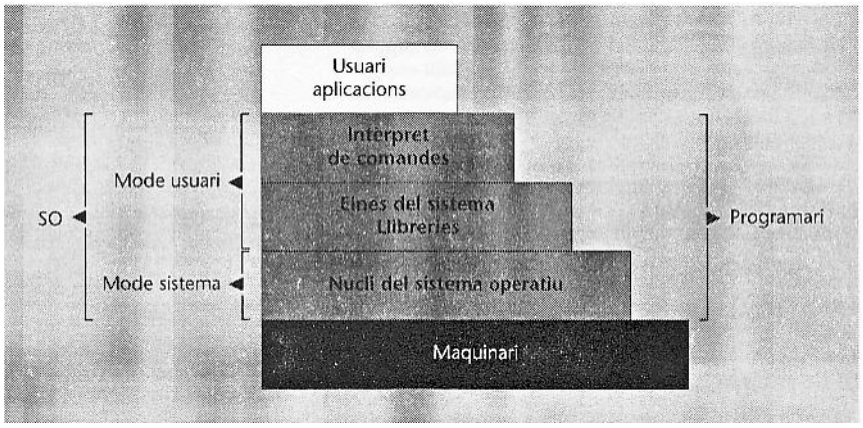

Fig. 1.1 Model per capes d'un PC extret del llibre Introducció als SO

La capa inferior correspon al maquinari del computador. Per sobre d'aquesta trobem el sistema operatiu. És molt difícil donar una definició exacta i acurada del terme sistema operatiu, el qual generalment es defineix a partir de les funcions que desenvolupa. Bàsicament són les dues funcions que es presenten a continuació:

a.-Gestió eficient dels recursos del sistema

El SO controla l'accés eficient als recursos del computador: la memòria principal, el temps de la unitat central de procés (CPU) i els dispositius. És a dir, el SO s'encarrega principalment de tasques de protecció i d'utilització eficient del sistema.

En general, en un sistema computador hi han molts programes que s'executen al mateix temps, són programes que poden pertànyer a un o diversos usuaris, els quals competeixen pels diferents recursos del sistema. El sistema operatiu reparteix el temps de CPU entre els diferents programes i aconsegueix una execució concurrent, protegeix la memòria i coordina l'accés als dispositius compartits (disc , memòria) i als no compartits (impresora).

b.- Presentació als usuaris d'una maquina virtual molt més senzilla d'utilitzar

El sistema operatiu proporciona un entorn de treball a l'usuari i als programes d'aplicació que permet emprar el computador (executar programes) de manera més fàcil i intuïtiva. Des d'aquest punt de vista, el sistema operatiu proporciona a l'usuari una màquina virtual molt més fàcil d'entendre i utilitzar pel fet que amaga la complexitat del maquinari.

Per sobre del nucli del sistema operatiu tenim més programari de sistema, el qual consta de l'intèrpret de comandes, compiladors, editors i, en general, programes que faciliten la comunicació entre el sistema operatiu i l'usuari.

Finalment, a sobre del programari de sistema hi ha el programari d'aplicació. Una part d'aquest programari la fa servir l'usuari sobretot per a resoldre problemes específics. En aquest nivell trobem fulls de càlcul, processadors de text, jocs, etc.

1.1. Els serveis que proporciona un SO

El sistema operatiu ofereix una gran varietat de serveis. Els podem agrupar en els dos blocs següents:

a) Les crides de sistema són els serveis més senzills, i permeten fer peticions directes al sistema operatiu. Les crides de sistema són la visió que té un programador dels serveis que pot oferir el sistema operatiu. Els llenguatges d'alt nivell en ultima instancia fan una crida de sistema.

b) A un nivell més pròxim a l'usuari, el programari de sistema ofereix un programa, l'anomenat intèrpret de comandes, gràcies al qual l'usuari pot dialogar amb el sistema operatiu sense necessitat d'escriure cap programa.

Fora d'aquests dos grans blocs també hem de tenir en compte les funcions d'arranc del sistema.

1.1.1. Les crides de sistema

Les crides de sistema ofereixen les funcions bàsiques per a poder utilitzar tots els recursos del sistema de manera correcta i controlada. Encara que els diferents sistemes operatius poden oferir diverses crides al sistema (des de el punt de vista sintàctic), en general en tots podem trobar una sintàctica equivalent.

Les crides de sistema es poden classificar en: gestió de processos, comunicació i senyalització de processos, gestió de dispositius de entrada/ sortida, gestió directa dels recursos del sistema (control dels usos de CPU i memòria per part dels processos), gestió del sistema d'arxius, proteccions i funcions de temps.

1.1.2. L'intèrpret de comandes

L'intèrpret de comandes és un programa encarregat d'interpretar i comunicar al SO el que vol fer l'usuari de sistema. Aquest programa reconeix un conjunt limitat de comandes que, bàsicament, permeten a l'usuari accedir, modificar, crear i protegir la informació.

1.2. El nucli del SO

El nucli és la part del sistema operatiu formada per un conjunt de rutines residents sempre a la memòria principal. És l'encarregat de gestionar directament el maquinari i d'implementar les funcions més bàsiques de la gestió de processos, memòria i entrada/sortida. En concret, la gestió del mecanisme que regula les interrupcions és una de les funcions més importants del nucli, tant pel que fa a les interrupcions pròpiament dites, com a les excepcions i les crides al supervisor (trap).

El nucli, que també rep el nom de kernel o core, només és una part petita de tot el sistema operatiu, però és el més utilitzat. Com s'ha esmentat, el nucli resideix a la memòria, en canvi les altres parts del sistema es carreguen a la memòria exclusivament quan es necessiten.

El nucli determina les maneres com es pot invocar el codi del sistema operatiu a fi d'obtenir un cert servei (ja sigui dels dispositius o dels processos). La següent figura mostra un diagrama de les maneres possibles en que es pot executar el nucli.

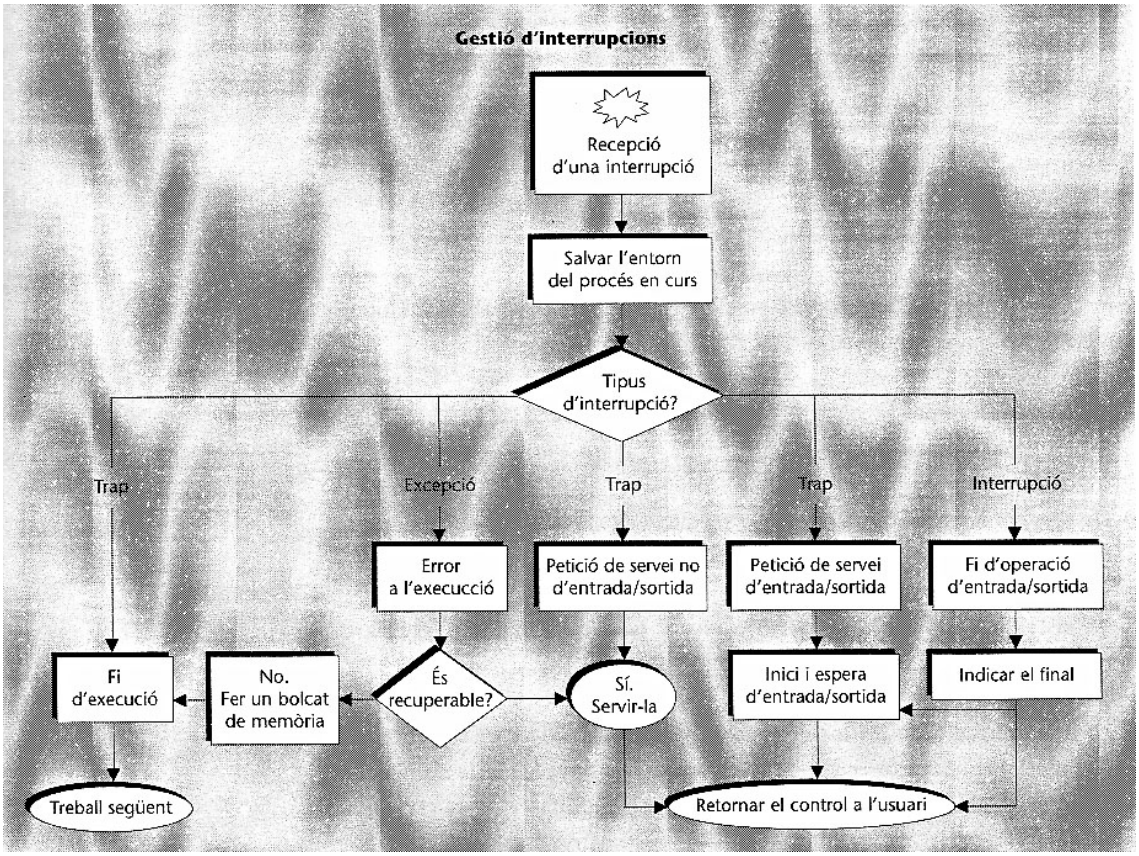

Fig. 1.2 Diagrama de la gestió d'interrupcions del nucli extret del llibre Introducció als SO

Per a poder garantir aquesta gestió el nucli utilitza totes les eines que ofereix el maquinari, les quals tenen una relació molt estreta amb la circuiteria involucrada en les interrupcions. Podem dir que els mecanismes més significatius són els següents:

a) L'execució en un mode privilegiat que li permet accedir a tots els registres de control de la màquina i també executar instruccions que en el mode normal no provocarien un interrupció.

b) La jerarquització de les interrupcions mitjançant un ordre de prioritats. Per a poder organitzar correctament la urgència dels serveis que se sol.liciten al nucli cal organitzar-los segons unes prioritats. La jerarquització d'aquests serveis mitjançant el maquinari simplifica i agilitza el funcionament del nucli.

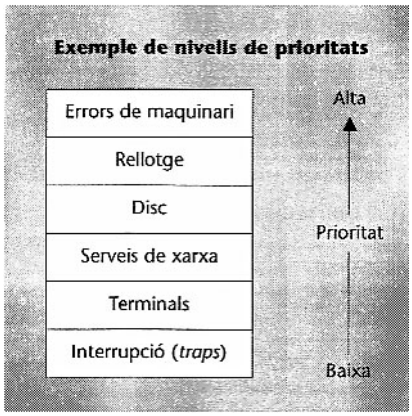

Fig. 1.3 Ordre de prioritats de les interrupcions (Introducció als SO)

1.3. Comparativa dels SO existents

	Unix	Windows	Mac OS	Linux
Versió considerada	Solaris 10	Windows XP Professinal Edition	OS X 10.4 (Tiger)	Gentoo 2.6.x
Preu mig d'adquisisció	Gratuït	130,00€	110,00€	Gratuït
Actualitzacions de seguretat gratuïtes	NO	SI	SI	SI
Codi font disponible	NO	NO	NO	SI
Propietat d'una empresa	SI	SI	SI	NO
Inclou aplicacions per a producció	NO	NO	NO	SI
Documentació técnica addicional gratuïta	NO	NO	NO	SI
Software servidor inclos	NO	NO	NO	SI
Tipus de hardware suportat	Intel/AMD i386 32/64bit, Sparc	Intel/AMD i386 32/64bit, PowerPC	Apple Mac	Intel/AMD i386 32/64bit, Sparc, Apple Mac, PowerPC y otros

Taula 1.1 Comparativa de diferents SO

Com podem observar a la taula anterior el fet de triar Linux per a execució d'aquest projecte no és un fet fortuït. No menystenint el fet que es gratuït, totes les distribucions Linux porten incloses el software de servidor. Com els serveis TFTP, NFS,etc. Fet necessari per a l'execució del projecte. A més a més, Les distribucions Linux suporten la majoria de hardware existent fet que pot fer implementar el projecte en qualsevol entorn de treball.

CAPÍTOL 2. EL SISTEMA D'ARRANC

2.1. Arrencada del sistema (Linux)

La etapa del sistema d'arranc depèn del hardware en el que Linux serà arrencat. Aquests programes resideixen en un regió especial de la memòria ROM. I ens proporciona les instruccions necessàries per a baixar i carregar una imatge d'un nucli de Linux i conseqüentment executar-la.

Quan un sistema operatiu es arrancat o reiniciat, el processador executa un programa situat a una localització coneguda. En un PC (Personal Computer), aquesta localització es troba en la BIOS (Basic input/output system). Aquesta esta emmagatzemada en una memòria ROM de la placa base. En un PC la característica diferencial respecte altres sistemes computacionals, es que la BIOS ofereix una major flexibilitat. Degut a que permet determinar quin dispositiu es el candidat per l'arrencada.

L'arrencada de Linux comença a la BIOS en l'adreça 0xFFFF0. El primer pas que fa la BIOS es encendre un auto-test (POST). La tasca de POST es la de comprovar el hardware. El segon pas de la BIOS es la enumeració i inicialització dels dispositius locals.

Donat els diversos usos de la BIOS, aquesta esta construïda en dos parts: el codi de POST i els serveis que ens dona en temps d'execució. Després de que el POST s'hagi completat, aquest s'esborra de la memòria. Però els serveis en temps d'execució romanen en la memòria i són disponibles per al SO.

Per a l'arrencada del SO, els serveis de la BIOS busquen els dispositius que estan actius i son arrencables en l'ordre de preferència definit al CMOS (Complementary Metal Oxide Semiconductor), memòria de lectura escriptura. El dispositiu d'arranc potser un disquet, un CD-ROM, una partició del disc, " un dispositiu de xarxa" o una memòria USB.

Quan es troba el dispositiu d'arranc, la primera etapa del CARREGADOR D'ARRANC es carrega dins de la RAM i s'executa. Aquest CARREGADOR D'ARRANC te menys de 512 bytes de longitud (un únic sector), i la seva feina es la de carregar la segona etapa del CARREGADOR D'ARRANC.

Comunment un PC es arrencat des del disc dur, on el MBR (Master Boot Record) conté el carregador d'arrencada primari. El MBR es un sector de 512 bytes, situat al primer sector del disc. Després el MBR es carregat a la RAM i la BIOS pren el control d'aquest.

Quan la segona etapa del CARREGADOR D'ARRANC es carregada en RAM i s'està executant, es carrega en memòria un sistema de fitxers temporal. Quan la imatge del nucli del sistema es carregada en memòria, la segona etapa del CARREGADOR D'ARRANC passa el control a la imatge del nucli. I aquesta es descomprimida i inicialitzada. Durant aquesta etapa , la segona etapa del

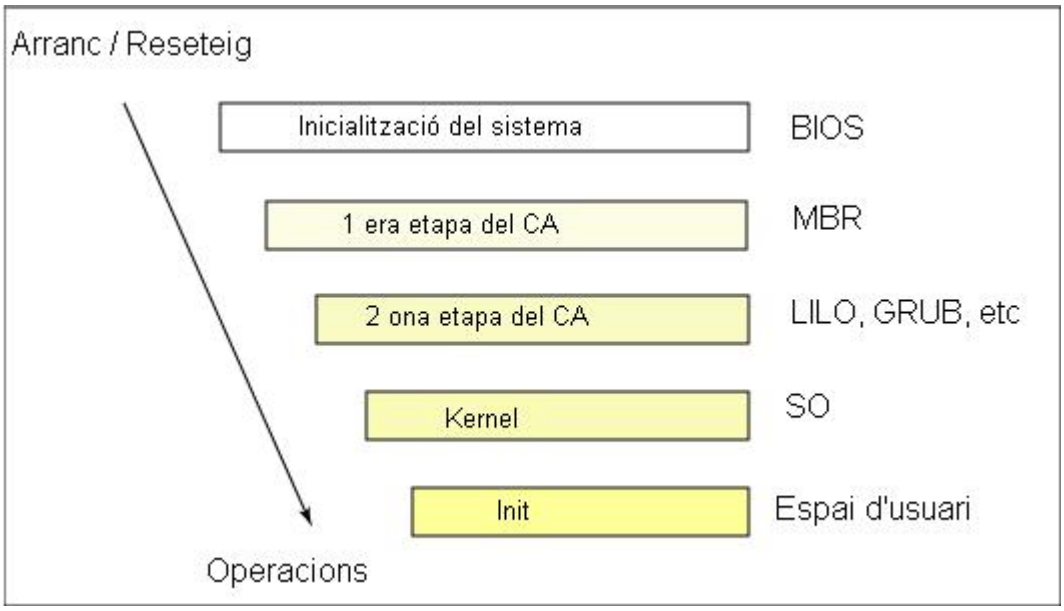

Fig. 2.1 Esquema d'arrencada d'un PC

2.2.1. 1ª etapa del carregador d'arranc

El carregador d'arranc primari resideix al MBR. Aquest es una imatge de 512 bytes que conte el codi de dos parts: una petita taula de particions i el carregador d'arranc primari (els primers 446 bytes), el qual conté un codi executable i un missatge d'error de text. La taula de particions conte una gravació de quatre particions. El MBR acaba en dos bytes que han estat definits com " el numero màgic ". Que serveixen per a validar el MBR.

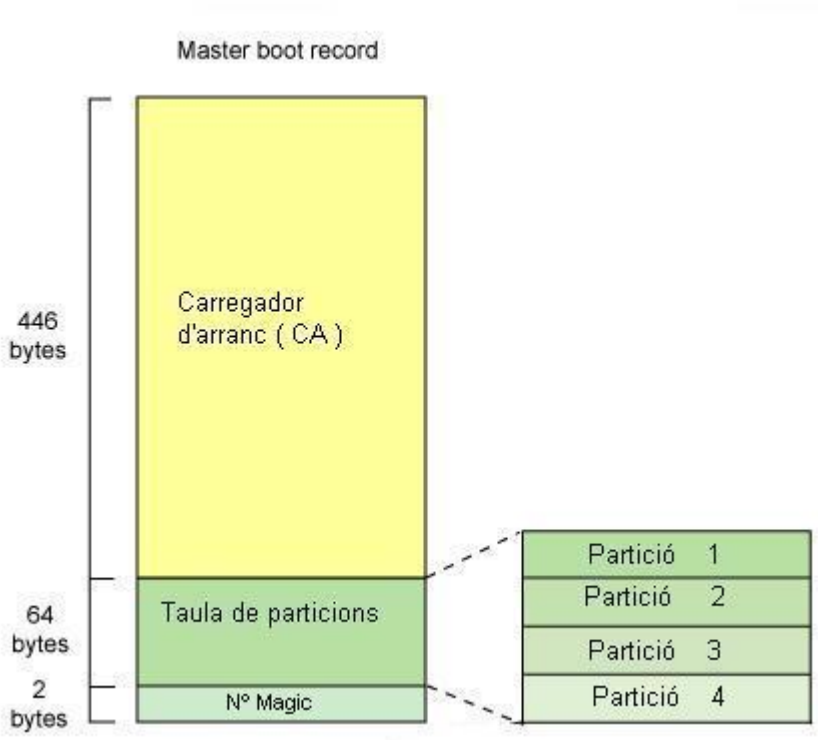

Fig. 2.2 Esquema del Master Boot Record

La feina del carregador d'arranc primari es la de trobar i carregar el carregador d'arranc secundari (2ona etapa). Quan aquest es trobat, s'escanegen les particions restants de la taula per a assegurar-se que estan inactives. Quan això es verifica, el sistema d'arranc de la partició activa es llegit i comprova el dispositiu RAM per a ser carregat i executat.

2.2.2. 2ª etapa del carregador d'arranc

El carregador d'arranc secundari, també anomenat carregador del nucli, té la tasca de carregar el nucli i opcionalment un disc de RAM.

Amb la segona etapa del carregador d'arranc en memòria, el sistema de fitxers es consultat, i la imatge del nucli a la vegada que la del initrd son carregades en memòria. Quan ja tenim les imatges preparades, la 2ª etapa del carregador d'arranc invoca la imatge del nucli.

2.2.3. El nucli

Amb la imatge del nucli en memòria i el control cedit per la segona etapa del carregador d'arranc, comença la etapa del nucli. La imatge del nucli no es un programa executable. Típicament aquesta es una zImage (imatge comprimida menor a 512 KB) o una bzImage (imatge comprimida major a 512 KB big zImage) que ha sigut prèviament comprimida amb zlib.

En la capçalera de la imatge del nucli resideix una rutina que fa algunes operacions mínimes de configuració del hardware i descompressió del nucli contingut dins la imatge, situant-lo a la memòria virtual. Si la imatge d'un disc RAM inicial es present, aquesta rutina el mou a la memòria del disc. Després la rutina crida al nucli i l'execució del nucli comença.

2.3. Sistemes d'arranc múltiples

Existeixen diferents sistemes d'arranc múltiple. Els que s'utilitzen més freqüentment són LILO i GRUB. Dels quals farem una breu descripció de les seves característiques principals i una comparativa entre ambdós.

La primera i segona etapa del carregador d'arranc combinades s'anomenen LInux LOader (LILO) o GRand Unified Bootloader (GRUB) en entorns PC x86. Degut a que LILO te algunes desavantatges que han estat corregides a GRUB (que explicarem més endavant) en aquest projecte s'utilitzarà GRUB com a mètode d'arranc de múltiples SO.

Com a gran avantatge en la utilització de sistemes d'arranc múltiples, esdevé el fet que quan la segona etapa es carregada es representa per pantalla un llistat dels nuclis disponibles (definits a /etc/grub.conf). A partir d'aquí nosaltres podem carregar el nucli i a demés enviar-li paràmetres addicionals. Opcionalment, podem utilitzar la línea de comandes per a tenir un millor control manual sobre el procés d'arranc.

2.3.1. LILO

LILO ve com a estàndard en totes les distribucions Linux. Com un dels carregadors d'arranc més antics de Linux, la ajuda continuada de la forta comunitat Linux ha permès que s'hagi desenvolupat en el temps i que sigui a dia d'avui un carregador d'arranc viable. Les noves funcionalitats inclouen una interfície gràfica a nivell d'usuari i una bona explotació de les noves funcionalitats de les BIOS que eliminen el límit de 1024 cilindres (existent en les antigues BIOS degut a una falta de previsió en el creixement del hardware informàtic).

Encara que LILO continua desenvolupant-se, els principis de base com les tasques de LILO segueixen sent els mateixos.

2.3.1.1. El procés inicial

Quan LILO es carregada inicialment, s'executa lexicogràficament seguint l'ordre de cadascuna de les seves lletres -L-I-L-O. Si totes les seves lletres són superades, la primera etapa haurà sigut carregada correctament. Qualsevol de les lletres no superades ens indica que hi ha hagut un problema:

- L: Ha sigut carregat (mogut a memòria) i executat el carregador d'arranc de la primera etapa. Si LILO s'atura en aquest punt ens indicaria que hi hauran problemes per a carregar la segona etapa del sistema d'arranc, es a dir, el carregador d'arranc de la primera etapa s'ha executat incorrectament.
- LI: S'ha carregat el carregador d'arranc de la segona etapa. Si LILO s'atura en aquest punt ens indica que el carregador d'arranc de la segona etapa no podrà ser executat.
- LIL: S'ha executat correctament el carregador d'arranc de la segona etapa.
- LILO: LILO s'ha carregat amb èxit i sense errors si s'atura en aquest punt el nucli no s'haura pogut carregar correctament.

2.3.2. GNU GRUB

Més recentment, el carregador unificat del sistema d'arranc (conegut comunment com a GRUB) sembla haver agafat algunes de les parts de la comunitat LILO. GNU GRUB esta desenvolupada per la fundació de software lliure i basada activament en el programari original de GRUB, creat originalment per Erich Stefan Boleyn.

Quan es carrega GRUB inicialment, igualment que LILO es carrega la primera etapa (MBR). Una vegada aquesta s'ha carregat, s'incorpora una etapa intermitja entre la 1ª i 2ª etapa. La etapa 1,5 esta present per a permetre l'accés regular al sistema de fitxers situat en /boot/grub. Això permet tenir accés als arxius de configuració de GRUB utilitzant els blocs del disc. Llavors s'incorpora la 2ªetapa del carregador d'arranc on GRUB carrega el archiu grub.conf (on es tria i s'executa el nucli).

2.3.2.1. GRUB vs LILO

Segons el que s'ha explicat anteriorment, tots els carregadors d'arranc treballen d'una manera similar per a satisfer un propòsit comú. Però LILO i GRUB presenten algunes divergències:

- LILO no te una interfície de comandes interactives, mentres que GRUB sí.
- LILO no suporta arrencades a traves de xarxa, mentres que GRUB sí.
- LILO emmagatzema informació respecte a la localització dels sistemes operatius que pot carregar físicament al MBR. Si es canvia el archiu de configuració de LILO, s'ha de reescriure el carregador d'arranc de la primera etapa de LILO al MBR. Això podria desconfigurar el MBR i deixa el sistema inarrencable. Amb GRUB si el archiu de configuració es configura incorrectament, simplement s'omitirà la línea de comanda de GRUB.

2.3.2.2. El futur de GRUB

GRUB esta essent substituïda per GRUB2. On a més a més de la fixació dels forats, s'està fent un desenvolupament actiu. GRUB2 serà una reescriptura completa del carregador d'arranc original. A dia d'avui, la base dels canvis realitzats es la següent:

- El reemplaçament de la etapa 1,5 amb la creació d'una imatge compacta de base (poder incloure el accés a grub.conf des del MBR).
- Carregar dinàmicament la imatge de base.
- Internacionalització, com jocs de caràcters non-ASCII
- Extensió per a diverses arquitectures de hardware i diverses plataformes (a excepció de Linux)

2.3.3. Conclusió

Com en tot el software, la millor opció per a un usuari no es sempre la millor per a tots. Dels carregadors d'arranc que hem tractat aquí el que per a mi es el millor es GNU GRUB. És un carregador versàtil que combina una interfície de usuari polida amb una gran quantitat de funcionalitats. Entre elles la capacitat d'arrancar des de la xarxa.

Amb respecte a la seguretat, qualsevol persona que tingui accés a un disc/ CD-ROM d'arranc pot desconfigurar l'arxiu de configuració de GRUB o LILO. Una manera senzilla per evitar aquest problema és inhabilitar l'arrencada des de disc o CD-ROM a la BIOS i protegir aquesta amb una contrasenya per a que ningú pugui canviar aquests ajustos.

2.4. Arranc per xarxa

Un dels problemes més comuns per als administradors de xarxes es tenir un sistema segur i apropiat per que els clients (màquines clients d'un servei) de la xarxa puguin arrencar les imatges dels seus SO amb la configuració correcte. El problema recau en que un mal ús de la màquina pot porta a desconfiguracions del sistema, entrades de virus o inclús a fallades del hardware instal.lat. Fins fa relativament poc temps, la solució a aquests problemes era fer una copia de seguretat de disc a disc dels sistemes dels clients. Això implicava una feina molt farragosa i tediosa per a l'administrador de la xarxa. De forma més còmoda i eficient aquestes imatges d'arranc poden ser adquirides pels clients mitjançant servidors de la xarxa local que ens proporcionen les imatges adequades per a cada entorn en particular. A més a més, els clients han de poder arrencar d'una forma consistent on no ha de tenir importància ni el software requerit ni el hardware tant dels clients com dels servidors.

Aquesta tasca pot ser complida només amb un protocol de prearrencada uniforme i consistent. Els serveis que ens proporciona la BIOS del client han d'assegurar que l'arranc basat en xarxa ha d'acomplir els estàndards del

protocols de la industria utilitzats per a la comunicació amb el servidor. Per assegurar la interoperatibilitat el NBP (Network Bootstrap Program) que ens descarreguem ha de ser un sistema de prearranc consistent i uniforme dintre del client d'arranc. Per a poder acomplir la seva tasca independentment del tipus de xarxa en el que s'implementa el sistema, aquesta capacitat és molt important per a la les diferents situacions on es pot trobar el client, per exemple:

- El client no té un SO instal·lat al seu disc dur, la descarregar del NBP ens ajudar a automatitza la instal·lació d'un SO.

- La màquina del client falla en l'arrencada, tant sigui per una fallada del software com del hardware, la descarrega d'una imatge executable on es pot proporcionar un diagnòstic del problema que ha sorgit.

- La maquina del client no té un sistema d'emmagatzematge local. El sistema pot descarregar una imatge executable (NBP) des del servidor com si fos una operació normal.

Per a tals finalitats PXE (Preeboot eXecution Enviroment) és un sistema que ens proporciona la fiabilitat necessària. PXE inclou tres tecnologies que ens permeten establir un servei consistent de prearrencada. En resum, fent ús de les característiques esmentades anteriorment, PXE ha de garantir que una màquina client sigui capaç de conectar-se a una xarxa heterogènia, adquirir per ella mateixa una adreça de xarxa des d'un servidor DHCP i descarregar el NBP i arrencar-lo.

2.4.1. Procés d'arrencada

El procés d'arrencada de PXE es pot dividir en sis etapes:

1. El PC s'encen, la BIOS consulta al CMOS on li es indicat l'arrencada per xarxa.

2. Executa el codi la ROM que conté carregador de PXE.

3 . El carregador de PXE envia una petició de DHCP a la xarxa.

4. El client espera fins a rebre la resposta DHCP que respon (si) ha de contenir una opció addicional de DHCP, amb les opcions del "nom de fitxer".

5 . El PC client de PXE procura descarregar l'arxiu del TFTP especificat. (si no esta especificat, el client intenta connectar amb la computadora que li va donar la renda de DHCP. Si s'especifica el paràmetre del "següent-servidor", en lloc d'un altre procurarà descarregar l'arxiu d'aquest servidor. S'ha d'observar que el DNS no està disponible aquí, així que s'ha de donar una adreça IP.

6 . El PC client descarrega l'arxiu, llavors executa aquest arxiu i el procés d'arranc de PXE s'acaba. Aquest és el procés sencer de PXE.

Si qualsevol dels passos anteriors fallen, la computadora ha de continuar amb el procés d'arranc. En cas que no rebi resposta del servidor DHCP o degut algun problema de transmissió no sigui capaç de adquirir una adreça o de descarregar el NBP, intentarà arrencar el sistema des del disc dur. En cas que no trobi cap dispositiu que pugui arrencar el sistema tornarà a intentar l'arrencada mitjançant PXE.

2.4.2. Serveis relacionats amb PXE

Com anteriorment s'ha comentat PXE fa us de tres tecnologies per a dur a terme la seva tasca. Aquestes tecnologies o serveis relacionats amb PXE son tres: DHCP, TFTP i NBP.

2.4.2.1. DHCP

Les sigles DHCP signifiquen Dinamic Host Configuration Protocol. Aquest protocol s'utilitza normalment en grans xarxes però és una eina útil per a qualsevol tipus de xarxa. Aquest protocol serveix per a conectar amb un servidor que es utilitzat per a donar informació a les estacions de treball de la xarxa, com la adreça IP, la mascara de la xarxa , el servidor DNS, la porta d'enllaç, etc…

DHCP ha sigut creat per el grup de treball Dynamic Host Configuration del IETF. I la seva definició es pot troba als RFC 2131, protocol DHCP, y al 2132, opcions de DHCP.

De forma similar a altres protocols similars, utilitza el paradigma client-servidor, per a que els nodes clients obtinguin la seva configuració del servidor.

El protocol de Configuració Dinàmica de Hosts, permet la transmissió de la configuració dels hosts sobre una xarxa TCP/IP. Aquest protocol s'encarrega de la configuració automàtica dels paràmetres de la xarxa. DHCP es una extensió de BOOTP i es compatible amb el mateix.

Un servidor DHCP esta format per dos bases de dades. La primera es estàtica (on hi ha un llistat de adreces MAC relacionades amb les seves adreces de xarxa), i la segona conte una pila amb el nombre de adreces disponibles (per a qualsevol MAC que no estigui dins de la base de dades estàtica). Aquesta segona base de dades es la que fa el DHCP dinàmic. Quan un client DHCP demana una adreça de forma temporal, DHCP agafa la pila d'adreces IP disponibles i li assigna una durant un període negociat.

El servidor DHCP admet tres tipus de configuració de adreces IP:

1. Estàtica. Es configura el PC amb l'adreça de xarxa que correspon amb la adreça LAN del client.
2. Dinàmica, per temps il·limitat. S'indica un rang d'adreces que s'assignen a cada client de manera permanent, fins que el client la allibera.
3. Dinàmica, llogada. Les adreces s'atorguen per un temps limitat. Un client ha de renovar la seva adreça per a poder segui utilitzant-la.

Quan el servidor DHCP rep una petició, primer comprova la seva base de dades estàtica. Si existeix una entrada per a aquella adreça física, es retorna la adreça IP estàtica corresponen. Si no es troba la entrada, el servidor selecciona una IP disponible de la base de dades dinàmica i afegeix una nova associació a la base de dades.

La adreça assignada des de la pila es temporal. El servidor DHCP emet un lloguer per un període determinat de temps. Quan el lloguer acaba, el client ha de deixar de fer servir aquesta adreça o renovar el lloguer. El servidor te la opció de renovar o negar la renovació.

Protocol. El client realitza els següents passos:

1. Envia un missatge *DHCPDISCOVER* broadcast utilitzant el port de destí 67.
2. Aquells servidors que puguin donar aquest tipus de Server envien un missatge DHCPOFFER, on s'ofereix una IP que serà bloquejada. En aquests missatges també es pot oferir la duració del lloguer que per defecte es una hora. Si els clients no reben aquest missatge, intenta establir connexió Quatre cops més,cada dos segons, si encara així no hi ha desposta, el client espera cinc minuts abans de tornar-ho a intentar de nou.
3. El client escull una de les adreces IP ofertades i envia un missatge DHCPREQUEST al servidor seleccionat.
4. El servidor respon amb un missatge DHCPACK i crea una associació entre l'adreça física del client i la IP assignada. Ara el client utilitza la adreça llogada fins que el temps expiri.
5. Abans de arribar al 50% del temps de lloguer, el client envia un altre missatge DHCPREQUEST per ha renovar el lloguer.
6. El client pot acabar amb el lloguer abans de que expiri el temps negociat enviant un paquet DHCPRELEASE al servidor.

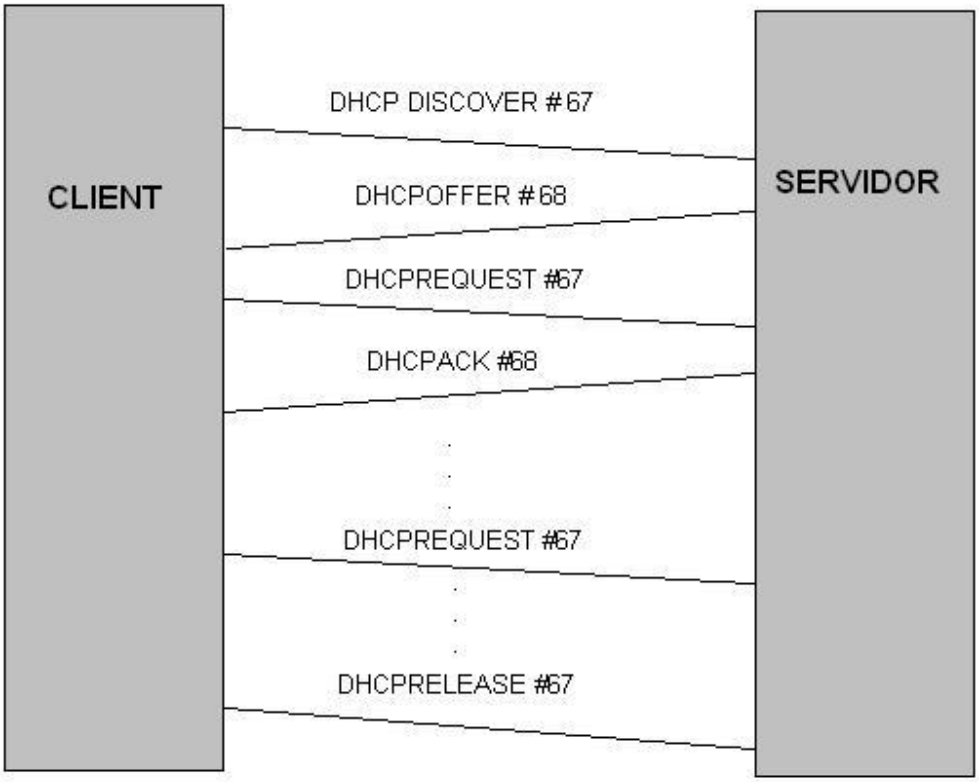

Fig. 2.3 Esquema DHCP

2.4.2.2. TFTP

TFTP son les sigles de Trivial File Transfer Protocol (Protocol de transferència trivial d'arxius). Es un protocol de transferència molt simple, similar a una versió bàsica de FTP. Un dels usos més comuns d'aquest protocol, es la transferència de petits archius entre ordinadors d'una mateixa xarxa. Com per exemple quan qualsevol thin client arranca des de un servidor de la xarxa, ja que es més ràpid que FTP (TFTP utilitza e protocol de transport UDP), al no tindre un control d'errors en la transmissió

Característiques de TFTP:

- Utilitza UDP (port 69) com a protocol de transport (a diferencia de FTP que utilitza el protocol TCP – port 21)
- No pot llistar el contingut dels directoris
- No existeixen mecanismes de autentificació o encriptació
- S'utilitza per llegir o escriure arxius de un servidor remot
- Suporta tres modes diferents de transferència, "netascii", "octet" i "mail", dels quals els dos primers corresponen a modes ASCII.

Degut a que TFTP utilitza UDP, no hi ha una definició formal de sessió, client servidor. En canvi, cada arxiu transferit via TFTP constitueix un intercanvi independent de paquets, i existeix una relació client-servidor informal entre la màquina que inicia la comunicació i la que respon.

- La màquina A, que inicia la comunicació, envia un paquet RRQ (read request, petició de lectura) o WRQ (write request, petició d'escriptura) a la màquina B, contenint el nom de l'arxiu i el mode de transferència.
- B respon amb un paquet ACK (acknowledgement / confirmació), que també serveix per a informar a A del port de la maquina B al que tindrà que enviar els paquets restants.
- La màquina origen envia els paquets de dades numerats a la màquina destí, tots excepte el últim contenen 512 bytes de dades. La màquina destí respon amb paquets ACK numerats tots els paquets de dades.
- El paquet de dades final ha de contenir menys de 512 bytes de dades per a indicar que es el últim. Si el tamany del archiu es múltiple exacte de 512bytes, el origen envia un paquet final que conte 0 bytes de dades.

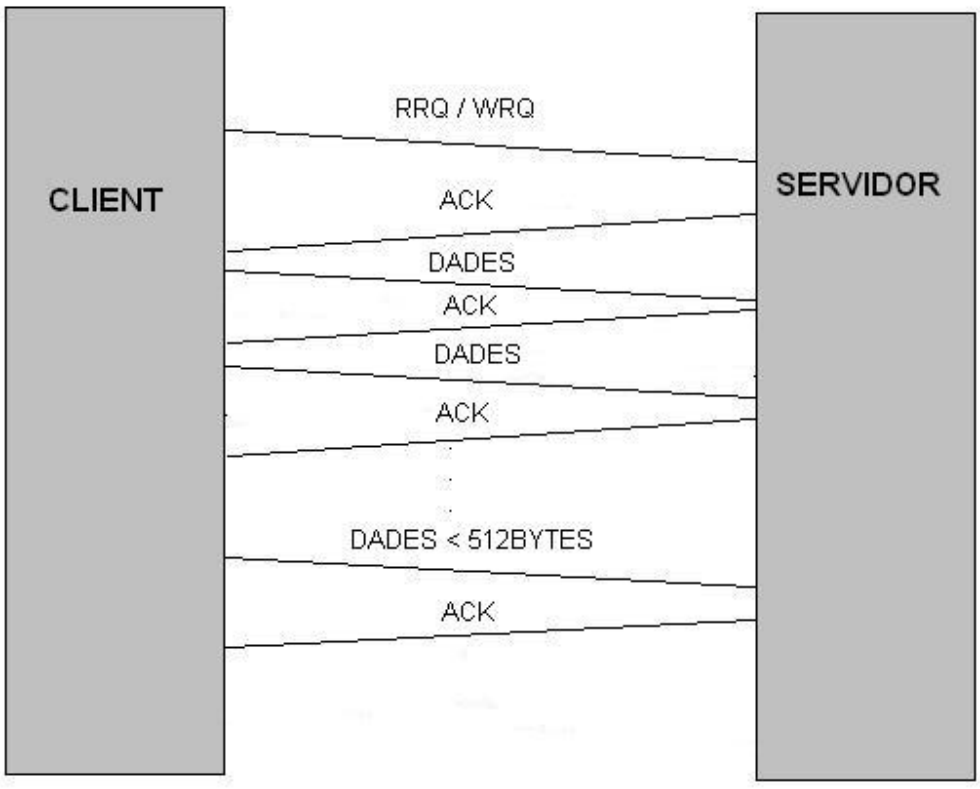

Fig. 2.4 Esquema transmissió TFTP

2.4.2.3. PXE LINUX

PXELinux és una distribució micro de Linux que permet construir thin-clients o llocs de treball diskless. Després d'arrencar el thin-client serà capaç de tenir accés a qualsevol servidor d'Unix/Linux XDM (encarregat de l'exhibició de X) que presenta la pantalla gràfica de la connexió o qualsevol Microsoft terminal-server amb RDP (protocol d'escriptori allunyat), servidor de Citrix ICA, servidor de VNC o NoMachine NX.

SYSLINUX és un carregador d'arranc del sistema operatiu Linux (segona etapa del carregador d'arranc) que opera dintre del sistema de fitxers FAT. Aquest serveix per a simplificar la primera instal·lació de Linux i la creació de un disc d'arranc de seguretat. SYSLINUX suporta la carrega del discs RAMs d'inici i el format del nucli bzImage.

PXELINUX és un derivat de SYSLINUX, per al booting de Linux en un servidor de la xarxa, usant una ROM de la xarxa que es conforma amb l'especificació de Intel PXE (entorn de la Pre-Execució). PXELINUX no és un programa pensat per a ser cremat en un PROM de la targeta de la xarxa.

CAPÍTOL 3. NFS

3.1. El sistema de fitxers

Dintre del sistema Unix tot són arxius: des de la memòria física de l'equip fins al ratolí, passant per mòdems, teclat, impressores o terminals. Aquesta filosofia de disseny és un dels factors que més èxit i potència proporciona a Unix, però també un dels que més perills entranya: un simple error en un permís pot permetre a un usuari modificar tot un disc dur o llegir les dades teclejades des d'una terminal. Per això, una correcta utilització dels permisos, atributs i altres controls sobre els fitxers és vital per a la seguretat d'un sistema.

En un sistema Unix típic existeixen tres tipus bàsics d'arxius: fitxers plans, directoris, i fitxers especials (dispositius). Generalment, al parlar de fitxers ens solem referir a tots ells si no s'especifica el contrari. Els fitxers plans són seqüències de bytes que a priori no posseïxen ni estructura interna ni contingut significant per al sistema: el seu significat depèn de les aplicacions que interpreten el seu contingut. Els directoris són arxius el contingut dels quals són altres fitxers de qualsevol tipus (plans, més directoris, o fitxers especials), i els fitxers especials són fitxers que representen dispositius del sistema; aquest últim tipus es divideix en dos grups: els dispositius orientats a caràcter i els orientats a bloc. La principal diferència entre ambdós és la forma de realitzar operacions d'entrada/sortida: mentre que els dispositius orientats a caràcter les realitzen byte a byte (això és, caràcter a caràcter), els orientats a bloc les realitzen en blocs de caràcters.

El sistema de fitxers és la part del nucli més visible pels usuaris; s'encarrega d'abstreure propietats físiques de diferents dispositius per a proporcionar una interfície única d'emmagatzematge: l'arxiu. Cada sistema Unix té el seu sistema d'arxius nadiu (per exemple, ext2 en Linux, UFS en Solaris o EFS en IRIX), pel que per a accedir a tots ells de la mateixa forma el nucli d'Unix incorpora una capa superior denominada VFS (Virtual File System) encarregada de proporcionar un accés uniforme a diferents tipus de sistema de fitxers.

Un inode o node índex és una estructura de dades que relaciona un grup de blocs d'un dispositiu amb un determinat nom del sistema de fitxers. Internament, el nucli d'Unix no distingeix als seus arxius pel seu nom sinó per un nombre de inodo; d'aquesta forma, el fitxer amb nombre de inodo 23421 serà el mateix tant si es denomina /etc/passwd com si es denomina /usr/fitxer. Mitjançant l'ordre ln(1) es poden assignar a un mateix inodo diversos noms de fitxer diferents en el sistema d'arxius.

3.2. Sistemes de fitxers distribuïts

Un sistema de fitxers distribuïts emmagatzema arxius en un o més ordinadors denominats servidors i els fa accessibles per a uns altres ordinadors anomenats clients, des de on es manipulen com si estiguessin en local. Existeixen moltes aventatges en el ús de servidors de fitxers:

- Els arxius són més accessibles si des de diversos ordinadors es pot accedir als servidors.
- La distribució dels arxius des d'una única localització es molt més senzilla que distribuir copies dels arxius a tots els clients.
- Les copies de seguretat són molt més manejables quan només s'han de tenir en compte els servidors.
- Els servidors poden oferir un gran espai d'emmagatzematge que seria poc pràctic i costos de subministrar al client.

3.3. NFS

El sistema d'arxius en xarxa de Sun Microsystems, conegut universalment com NFS va ser dissenyat e implementat per Sun en principi per al seu ús en les estacions de treball UNIX. Actualment altres fabricants el suporten tant per UNIX com per altres sistemes operatius. NFS suporta sistemes heterogenis; per exemple, clients MS-DOS que utilitzen servidors UNIX. Ni tant sols es necessari que totes les màquines utilitzin el mateix hardware.

Tres aspectes importants de NFS són d'interès: l'arquitectura, el protocol i la implementació, que seran analitzats a continuació.

3.3.1. Característiques de NFS

La idea bàsica que hi ha darrera NFS es permetre que una colecció arbitraria de clients i servidors comparteixin un sistema d'arxius comú. En la majoria de casos, tots els clients i servidors estan situats en la mateixa LAN, pero això no te per què ser necessari. Es possible executar NFS en una xarxa d'area amplia (MAN o WAN). Per a simplificar l'exposició es parlarà de clients i servidors com si estiguessin en màquines diferents, però de fet, NFS permet que cada màquina sigui client i servidor al mateix temps.

Cada servidor NFS exporta un o més dels seus directoris per al seu accés per part dels clients remots. Quan es disposa d'un directori, també es disposa de tots els seus subdirectoris, de fet, els arbres de directoris s'exporten com a una unitat. La llista de directoris que pot exportar un servidor es conserva a l'arxiu /etc/exports, de manera que aquests directoris es poden exportar de manera automàtica al arrencar el servidor de NFS.

Els clients NFS tenen accés als directoris exportats al muntar-los. Quan un client munta un directori (remot), aquest passa a forma part de la seva jerarquia de directoris. Moltes estacions de treball Sun no tenen disc. Si així es desitja, un client sense disc pot muntar un sistema d'arxius remot en el seu directori arrel, el que produeix un sistema d'arxius completament suportat en un servidor remot. Les estacions de treball que sí tenen discos locals poden muntar directoris remots en el lloc que desitgin de la seva jerarquia local de directoris, el que produeïx un sistema d'arxius parcialment local i parcialment remot.

Així, la característica bàsica de l'arquitectura de NFS és que els servidors exporten directoris i els clients els munten de manera remota. Si dos o més clients munten el mateix directori al mateix temps, es poden comunicar compartint arxius en els seus directoris comuns. Un programa en un client pot crear un arxiu, i un programa en altre client pot llegir l'arxiu. Una vegada realitzats els muntatges, no cal fer res especial per a assolir compartir els arxius. Els arxius compartits només estan aquí, en la jerarquia de directoris de diverses màquines i poden llegir-se i escriure en ells de la manera usual. Aquesta senzillesa és un dels punts forts de NFS.

3.3.1. Protocols de NFS

Un dels objectius de NFS és suportar un sistema heterogeni, amb clients i servidors que tal vegada executin diferents sistemes operatius amb un maquinari diferent i és essencial que la interfície entre els clients i els servidors estigui ben definida. Només llavors és possible que qualsevol pugui escriure una nova implantació client i esperi que funcioni de manera correcta amb els servidors existents i viceversa.

NFS assoleix aquest objectiu al definir dos protocols client-servidor. Un protocol és un conjunt de sol·licituds enviades pels clients als servidors, juntament amb les respostes enviades de retorn dels servidors als clients. Mentre un servidor reconegui i pugui controlar totes les sol·licituds en els protocols, no necessita saber gens dels seus clients. De manera anàloga, els clients poden tractar als servidors com "caixes negres" que accepten i processen un conjunt específic de sol·licituds. La forma com ho fan és assumpte d'ells.

3.3.1.1. Protocol d'anclatge

El primer protocol NFS controla l'anclatge. Un client pot enviar un nom de ruta d'accés a un servidor i sol·licitar que munti aquest directori en alguna part de la seva jerarquia de directoris. El lloc on es muntarà no està contingut en el missatge, ja que el servidor no es preocupa per aquest lloc. Si la ruta és vàlida i el directori especificat ha estat exportat, el servidor retorna un identificador d'arxiu al client. Aquest conté camps que identifiquen de manera única al tipus de sistema d'arxiu, el disc, el nombre de inode del directori, i informació de seguretat. Les crides posteriors per a la lectura i escriptura d'arxius en el directori muntat utilitzen l'identificador de l'arxiu.

Molts clients estan configurats de manera que muntin certs directoris remots sense intervenció manual. En general, aquests clients contenen un arxiu anomenat /etc/rc, que és un shellscript que conté les instruccions per al muntatge remot. Aquest shellscript s'executa de manera automàtica quan el client s'arrenca.

Una altre alternativa a la versió d'UNIX de Sun suporta també el automuntaje. Aquesta característica permet associar un conjunt de directoris amb un directori local. Cap dels directoris remots es munta (ni es realitza el contacte amb els seus servidors) quan arrenca el client. En comptes d'això, la primera vegada que s'obre un arxiu remot, el sistema operatiu envia un missatge a cadascun dels servidors. El primer a respondre guanya, i es munta el seu directori.

El automontaje té dos avantatges principals sobre el muntatge estàtic per mitjà de l'arxiu /etc/rc. La primera és que si un dels servidors NFS cridats en /etc/rc no està servint peticions, és impossible despertar al client, almenys no sense certa dificultat, retard i uns quants missatges d'error. Si l'usuari ni tan sols necessita aquest servidor de moment, tot aquest treball es transforma en una perdua de recursos. En segon lloc, al permetre que el client intenti comunicar-se amb un conjunt de servidors en paral·lel, es pot assolir cert grau de tolerància a fallades (ja que només es necessita que un d'ells estigui actiu) i es pot millorar l’eficiencia (al triar el primer que respongui, que suposadament té la menor càrrega).

D'altra banda, se suposa de manera implícita que tots els sistemes d'arxius especificats com alternatives per al automontaje són idèntics. Ja que NFS no dóna suport per a la rèplica d'arxius o directoris, l'usuari ha d'assolir que tots els sistemes d'arxius siguin iguals. En conseqüència, l’automontaje s'utilitza amb més freqüència per als sistemes d'arxius exclusius per a lectura que contenen binaris del sistema i per a altres arxius que rares vegades canvien.

3.3.1.2 Protocol d’accés

El segon protocol NFS és per a l'accés a directoris i arxius. Els clients poden enviar missatges als servidors perquè maneguin els directoris i llegeixin o escriguin en arxius. A més, també poden tenir accés als atributs d'un arxiu, com la manera, grandària i temps de la seva última modificació. La major part de les crides al sistema UNIX són suportades per NFS, amb la probable sorpresa de OPEN i CLOSE.

L'omissió de OPEN i CLOSE (En el servidor) no és un accident. És per complet intencionada. No és necessari obrir un arxiu abans de llegir-lo, o tancar-lo a l'acabar. En comptes d'això, per a llegir un arxiu, un client envia al servidor un missatge amb el nom de l'arxiu, una sol·licitud per a buscar-lo i retorna un identificador d'arxiu, que és una estructura d' identificació de l'arxiu. A diferència d'una crida OPEN, aquesta operació LOOKLJP no copia informació a les taules internes del sistema. La cridada READ conté a l'identificador d'arxiu que es desitja llegir, l'ajustament per a determinar el punt d'inici de la lectura i el nombre de bytes desitjats. Cadascun d'aquests missatges està autocontingut L'avantatge d'aquest esquema és que el servidor no ha de recordar el relatiu a

les conexions obertes entre les crides a ell. Així, si un servidor falla i després es recupera, no es perd informació sobre els arxius oberts, ja que no n'hi ha cap. Un servidor com aquest, que no conserva informació de l'estat dels arxius oberts es denomina sense estat.

Per contra, en el sistema V d'UNIX, el sistema d'arxius remots (RFS) requereix obrir un arxiu abans de llegir-lo o escriure en ell. El servidor crea llavors una entrada de taula amb un registre del fet que l'arxiu està obert i la posició actual del lector, de manera que cada sol·licitud no necessita un ajustament. El desavantatge d'aquest esquema és que si un servidor falla i torna a arrencar ràpidament, es perden totes les connexions obertes, i fallen els programes client. NFS no té aquesta característica.

3.3.1.3 Problemàtica de compatibilitat

Per desgràcia, el mètode NFS dificulta el fet d'assolir la semàntica d'arxiu pròpia d'UNIX. Per exemple, en UNIX un arxiu es pot obrir i bloquejar perquè altres processos no tinguin accés a ell. Al tancar l'arxiu, s'alliberen els bloquejos. En un servidor sense estat com NFS, els locks no es poden associar amb els arxius oberts, ja que el servidor no sap quins són els arxius estan oberts. Per tant, NFS necessita un mecanisme independent addicional per a controlar els bloquejos.

Els servidors d'informació de la xarxa es dupliquen mitjançant un ordre mestre/esclau. Per a llegir les seves dades, un procés pot utilitzar al mestre o qualsevol de les seves còpies (esclaus). No obstant això, totes les modificacions han de ser realitzades únicament en el mestre, que llavors les propaga als esclaus. Existeix un breu interval després d'una actualització en el qual la base de dades és inconsistent**.**

3.3.1.4 Problemàtica de seguretat

NFS utilitza el mecanisme de protecció d'UNIX, amb els bits rwx per al propietari, grup i altres persones. Al principi, cada missatge de sol·licitud només contenia els identificadors de l'usuari i del grup de qui va realitzar la crida, el que utilitzava el servidor NFS per a validar l'accés. De fet, confiava que els clients no mentissin. Diversos anys d'experiència han demostrat àmpliament que tal hipòtesi era una miqueta ingènua. Actualment, es pot utilitzar la criptografia de claus públiques per a establir una clau segura i validar al client i al servidor en cada sol·licitud i resposta. Quan aquesta opció s'activa, un client maliciós no pot personificar a un altre client, doncs no coneix la clau secreta del mateix. Per cert, la criptografia només s'utilitza per a autenticar a les parts. Les dades mai s'encripten.

3.3.1.5 Problemàtica d'autenticació

Totes les claus utilitzades per a l'autenticació, així com l'altra informació, són mantingudes pel NIS (servei d'informació de la xarxa). NIS es coneixia abans com el directori de pàgines grogues (yellow pages). La seva funció és la de guardar parelles (clau, valor). Quan es proporciona una clau, retorna el valor corresponent. No només controla les claus de xifrat, sinó també l'associació dels noms d'usuari amb les contrasenyes (xifrades), així com l'associació dels noms de les màquines amb les adreces de la xarxa, i altres elements.

3.3.3. Implementació de NFS

La capa superior és la capa de crides al sistema, la qual controla les crides com OPEN, READ i CLOSE. Després d'analitzar la crida i verificar els seus paràmetres, crida a la segona capa, la capa del sistema virtual d'arxius (VFS).

La tasca de la capa VFS és mantenir una taula amb una entrada per cada arxiu obert, anàloga a la taula de inodes per als arxius oberts en UNIX. En l'UNIX ordinari, un inode queda indicat de manera única mitjançant una parella (dispositiu, inode). En comptes d'això, la capa VFS té una entrada, anomenada vnode (inode virtual), per a cada arxiu obert. Els vnodes s'utilitzen per a indicar si un arxiu és local o remot. Per als arxius remots, es disposa de suficient informació per a tenir accés a ells.

Per a veure la forma d'utilitzar els vnodes, seguim una seqüència de crides al sistema MOUNT, OPEN i READ. Per a muntar un sistema remot d'arxius, l'administrador del sistema crida al programa mount amb la informació del directori remot, el directori local on serà muntat i algunes altres dades addicionals. El programa mount analitza el nom del directori remot per muntar i descobreix el nom de la màquina on es localitza aquest directori. Llavors, entra en contacte amb la màquina en la qual es localitza el directori remot. Si el directori existeix i està disponible per al seu muntatge remot, el servidor retorna un identificador d'arxiu per al directori. Finalment, crida a MOUNT per a transferir l'identificador de l'arxiu al nucli.

El nucli construeix llavors un vnode per al directori remot i demana el codi del client NFS per a crear un rnode (inode remot) en les seves taules internes, amb la finalitat de mantenir l'identificador de l'arxiu. El vnode apunta al rnode. Així, cada vnode de la capa VFS contindrà en última instància una referència a un rnode en el codi del client NFS o una referència a un inode en el sistema operatiu local. Així, és possible veure des del vnode si un arxiu o directori és local o remot i, si és remot, trobar el seu identificador d'arxiu.

Qui fa la crida rep un descriptor d'arxiu per a l'arxiu remot. Aquest descriptor d'arxiu s'associa amb el vnode mitjançant les taules en la capa VFS. Observi que no es creen entrades en les taules del costat del servidor. Encara que el servidor està llest per a proporcionar els identificadors d'arxiu que li sol·licitin, no manté un registre dels arxius que tenen identificadors actius i els quals no.

Quan se li envia un identificador d'arxiu per a l'accés a un arxiu, verifica l'identificador i, si aquest és vàlida, s'utilitza. El procés de validació pot incloure una clau d'autenticació continguda en les capceleres RPC, si la seguretat està activada.

Per raons d'eficiència, les transferències entre el client i el servidor es realitzen en blocs grans, en general de 8192 bytes, encara que se sol·licitin menys. Després que la capa VFS del client ha obtingut el bloc de 8Kbytes que necessitava, emet de forma automàtica una sol·licitud del següent bloc, pel que ho rebrà ràpidament. Aquesta característica es coneix com lectura avançada i millora en forma considerable el rendiment.

CAPÍTOL 4. KEXEC

4.1. Introducció a Kexec

Kexec és un grup de crides de sistema,creades per Eric Biederman l'any 2005, que ens permeten carregar un nucli des del nucli que s'està executant actualment en Linux. La implementació actual ha sigut únicament provada en arquitectures x86, però el codi genèric hauria de funcionar en qualsevol tipus d'arquitectura.

Abans d'entrar amb més profunditat en el funcionament de Kexec s'explicarà com funciona la familia de crides de sistema exec per a una més fàcil comprensió del funcionament de Kexec.

Cada una de les funcions dintre la família d'exec reemplaça la imatge del procés actual per una nova imatge de procés. La nova imatge és construïda des d'un fitxer regular i executable. Aquest fitxer a demés ha de ser un objecte executable. Es a dir, kexec és una crida que es fa des de un programa per a l'execució d'un altre dins d'aquest. Amb la conseqüència que la imatge del programa des d'on es fa la crida es reemplaçada pel nou procés. El nou procés no podrà retornar cap paràmetre al procés que fa la crida, per la senzilla raó que la imatge d'aquest queda solapada per la imatge del nou procés. A grans trets kexec fa una operació similar però en comptes d'imatges de processos, tracta amb imatges de nuclis.

L'execució de Kexec implica quasi bé una rearrancada del sistema amb l'exepció que no executa la part del firmware. Així doncs, la crida de sistema Kexec salta la primera part del carregador d'arranc i es situa en la segona etapa per a la del nou nucli. La part més sensible del procés d'arranc (la part del firmware) és totalment ignorada per part de kexec. El punt fort d'aquesta característica és, que ara reiniciar el sistema és extremadament ràpid. Per als desenvolupadors de software de sistema o de nucli, aquest procés fa guanyar molt de temps al reiniciar la màquina, al no tenir que passar constantment per l'etapa de firmware.

Com es pot intuir, Kexec toca moltes parts sensibles del sistema. Per això s'ha de tenir un compte extrem en el ús d'aquesta eina. Un dels grans avenços de Kexec, es que, en Linux el nou nucli ha d'estar situat en el mateix espai de memòria que el nucli que corre actualment. Reemplaçar el nucli existent pel nou, mentre encara esta corrent l'existent. Un altre avenç és l'omissió de l'estat dels dispositius, que durant la etapa de firmware son reiniciats per a assegurar el estat correcte d'aquests. Mentre Kexec fa un "bypass" total d'aquest procés. Així doncs això significa que l'estat dels dispositius és desconegut.

4.2. Utilitzant kexec

Kexec està compost per dos components. El primer component, és el component d'usuari conegut com a "kexec-tools". El segon component és una rutina del nucli. Ambdues parts són les que aconsegueixen les dues accions fonamentals de kexec: carregar el nou nucli en memòria i reiniciar cap a aquest.

Com hem esmentat anteriorment, la utilització de kexec consisteix en (1) carregar en memòria el nucli cap al qual reiniciarem el sistema (utilització de les kexec-tools) i (2) reiniciar aquest (invocació de la rutina del nucli de kexec mitjançant les kexec-tools). Per a la carregar del nucli en memòria, la sintaxis utilitzada és la següent:

kexec -l <kernel-image> --append="<command-line-options>"

A on <kernel-image> és l'arxiu que conté el nucli cap al qual reiniciarem i <command line options> conté els paràmetres de línea de comandes que necessitarà el nou nucli.

Degut a que una mala introducció dels paràmetres pot causar problemes durant el reinici, la manera més segura de passar els paràmetres que necessitarà el nou nucli es passar el contingut de /proc/cmdline.

Per exemple si volem reiniciar la imatge situada en /boot/BzImage i el contingut de /proc/cmdline es "root=/dev/hda1", la comanda per a la carregà del nou nucli serà:

kexec -l /boot/bzImage -append="root=/dev/hda1"

Ara, per a reiniciar el nucli que ja hem carregat hem de teclejar:

kexec -e

El sistema es reiniciarà immediatament. A diferència del procés normal de reinici, kexec no farà una apagada neta del sistema abans de reiniciar. És responsibilitat del usuari el fet de tancar (matar) totes les aplicacions i de desmuntar el sistema de fitxers abans de fer el reinici amb kexec. Sino kexec intentarà un reinici cap a un sistema mal definit amb el conseqüent mal funcionament d'aquest.

4.3. Les operacions de kexec

Un dels aspectes més importants del desenvolupament de kexec recau en el fet que els nuclis de Linux corren sobre una direcció fixa de la memòria. Això significa que el nou nucli necessita situar-se en la mateixa posició que el nucli que esta corrent. En els sistemes x86, el nucli es situa en la direcció física 0x100000 (direcció virtual 0xc0000000, coneguda com a PAGE_OFFSET). La tasca de sobreescriure el vell nucli amb el nou es fa en tres passes:

1. Copiar el nou nucli en memòria
2. Moure la imatge del nou nucli a la memòria dinàmica del nucli.
3. Copiar la imatge a la seva adreça real (sobreescrivint el nucli actual), i arrencar el nou nucli.

Els dos primers passos s'aconsegueixen durant la carrega del nucli. La primera tasca és interpretar els continguts del fitxer de la imatge del nucli. Les Kexec-tools han sigut construïdes per a aquest fi, en principi, es pot carregar qualsevol nucli (encara que no sigui Linux). Actualment, és possible arrencar qualsevol imatge en format elf32 (imatges executables compilades en 32 bits). El archiu es tractat i els "segments" del nucli son carregats als bufers que utilitza el codi de kexec. Aquests segments es categoritzen basant-se en la naturalesa del codi. Per exemple, en el cas d'utilitzar els arxius comunment utilitzats "bzImage", els segments típics són els 16-bit kernel code o 32 bit kernel code i el init ramdisk code. L'estructura utilitzada per aquests segments es coneguda com a kexec-segments i és una estructura bastant simple:

```
struct kexec_segment {
                        void *buf;
                        size_t bufsz;
                        void *mem;
                        size_t memsz;
};
```

Els primers dos elements de la estructura apunten al buffer del espai d'usuari i a la seva grandària . Els següents dos elements indiquen la destinació final del segment i la seva grandària.

Una vegada el arxiu del nucli amb el seu format específic carrega la imatge en memòria d'usuari, l'imatge es transferida a la memòria dinàmica del nucli per a ser utilitzada per la crida sys_kexec. La crida de sistema resitua les pagines del nucli dinàmic de cada un dels segments passats des de l'espai d'usuari i les copia a aquestes pagines del nucli.

Kexec situa les pagines del nucli fent servir un petit troç de codi en assemblador, conegut com a reboot_code_buffer. Aquest troç de codi fa la feina de sobreescriure el nucli actual pel que serà reiniciat i saltar cap a aquest. El reboot_code_buffer es l'únic en l'espai de reclinació final. En altres paraules,

aquest es ejecutat des del mateix lloc on inicialment s'havia carregat. La forma per a aconseguir això, en un sistema amb el MMU (Memory Management Unit) actiu, es que la pagina que mante el codi sigui mapejada idènticament. Això involucra la creació d'una entrada en la taula de paginat en init_mm (la taula de paginat de la estructura del nucli) amb la mateixa direcció física i virtual.

Una vegada la imatge del nou nucli ha sigut carregada, el sistema esta preparat per ha reiniciar-se cap a aquesta. La operació actual de reinici del nou nucli comença amb la comanda kexec –e. Aquesta comanda fa una crida al nucli per a que realitzi una rearrancada del sistema utilitzant la crida de sistema sys_reeboot, però amb un flag especial de LINUX_REBOOT_CMD_KEXEC.

La crida de sistema de rearrencada, utilitzant el flag especial, transfereix el control a la funció machina_kexec. Les funcions realitzades per machina_kexec son especifiques de l'arquitectura. En la implementació actual de x86, la seqüència d'accions es la següent:

1. Para les apics i deshabilitar les interrupcions (degut a que la rescripció és un procés molt sensible qualsevol interrupció del procés prodia fer perillar la integritat del sistema).
2. Copia el troç de codi assemblador (que es reedita durant la carrega de la imatge del nucli) al reboot_code_buffer. Aquest codi assemblador es troba a la rutina relocate_new_kernel.
3. Carrega tots els registres amb els segments d'informació dels valors del nucli (NUCLI_DS).
4. Salta al codi situat en reboot_code_buffer, i transfereix alguna informació vital, com els paràmetres del nucli. Així també la pagina que conté les adreces dels recursos i els destins de la imatge del nucli, la adreça de destí del nou nucli, la adreça de la pagina del reboot_code_buffer i un flag indican a on el sistema té la extensió de la adreça física activa.

Després de completar-se la seqüència, el nou nucli agafa el control i el nou sistema operatiu arrenca.

4.4. Beneficis de kexec

Una de les grans avantatges de Kexec és la seva velocitat al reiniciar el sistema. Algunes màquines tenen unes BIOS extremadament lentes a l'hora de reiniciar. En aquests casos kexec pot ser la única alternativa de reiniciar la màquina d'una manera confiable i oportuna. A demés del temps que ens estalvia en el reinici del sistema, Kexec es molt apropiat per al reinici des anomenats "kernel crash dump". Aquests són sistemes que salten d'un nucli a un altre en cas de que hi hagi un "kernel panic". Molts dels registres que es borren durant l'etapa de firmware ara es mantenen, així aportant una informació molt important a l'hora de detectar errors. Tot i això la utilització en aquest projecte de Kexec no respon a cap d'aquest avantatges sinó al fet que és l'única forma de no tindre que reiniciar el sistema per a iniciar el nou nucli.

CAPÍTOL 5. AUTOMATITZACIO I ÚS DEL SISTEMA DE RESTAURACIO REMOT

Durant els capítols anteriors s'han presentat les eines necessàries per a dur a terme aquest projecte. En conjunt han fet possible la creació del sistema de restauració remot d'imatges. Així com resultats de les probes que s'han fet amb aquestes.

5.1. Sistema de restauració remot

5.1.1. El servidor

El sistema de restauració remot esta compost per un servidor i diversos clients. En el servidor son necessaris els següents serveis: dhcpd,tftpd,PXELinux, NFS i el conjunt d'imatges que seran servides als clients. En resum el servidor tindrà actius els serveis de PXE.

El servei dhcpd s'arrenca amb la comanda "/etc/init.d/dhcp start" i fa servir el fitxer "/etc/dhcp/dhcpd.conf". A continuació es mostra la configuració del nostre fitxer amb els canvis efectuats marcats amb negreta

DHCPd:

```
# DHCP configuration file for DHCP ISC 3.0

ddns-update-style none;

# Definition of PXE-specific options
# Code 1: Multicast IP address of boot file server
# Code 2: UDP port that client should monitor for MTFTP responses
# Code 3: UDP port that MTFTP servers are using to listen for MTFTP
requests
# Code 4: Number of seconds a client must listen for activity before
trying
#         to start a new MTFTP transfer
# Code 5: Number of seconds a client must listen before trying to
restart
#         a MTFTP transfer

option space PXE;
option PXE.mtftp-ip               code 1 = ip-address;
option PXE.mtftp-cport            code 2 = unsigned integer 16;
option PXE.mtftp-sport            code 3 = unsigned integer 16;
option PXE.mtftp-tmout            code 4 = unsigned integer 8;
option PXE.mtftp-delay            code 5 = unsigned integer 8;
option PXE.discovery-control      code 6 = unsigned integer 8;
option PXE.discovery-mcast-addr   code 7 = ip-address;
```

```
// Subxarxa entorn de probes
subnet 192.168.1.0 netmask 255.255.255.0 {


  class "pxeclients" {
    match if substring (option vendor-class-identifier, 0, 9) =
"PXEClient";
    option vendor-class-identifier "PXEClient";
    vendor-option-space PXE;
    option PXE.mtftp-ip 0.0.0.0;
 //Adreça del servidor TFTP
    next-server 192.168.1.1;
  }

#  pool {
#    max-lease-time 86400;
#    default-lease-time 86400;
    # This prevents unlisted machines from getting an IP
#    deny unknown-clients;
#  }

  host test {
       # Use your slave's MAC address
// Aquí donem les ips estàtiques als clients dhcp, en les proves
realitzades només hem utilitzat la següent IP associada a la MAC del
client
       hardware ethernet                00:15:F2:90:44:AC;
       # Give your slave a static IP
       fixed-address                    192.168.1.4;
       server-name                      "master";
       # Use your gateway IP, if required
       option routers                   192.168.1.1;
       # Use your DNS IP, if required
       option domain-name-servers       192.168.1.1;
       option domain-name               "mydomain.com";
       # Use your slave hostname
       option host-name                 "test";


       # Etherboot and pxe boot with a mac specific image
//Aquí especifiquem la situació del arxiu que s'ha de descarregar
option root-path                 "/diskless/192.168.1.4";



//Aquí indiquem el nom del arxiu a descarregar si volguessim donar
NBP's diversos s'hauria de indicar als diferents rangs d'ips diferents
arxius

         if    substring    (option    vendor-class-identifier,    0,9)
="PXEClient" {
                        filename "/pxelinux.0";
        }

  }
}
```

Seguint el procediment del protocol PXE, la primera acció que realitzarà el client serà fer enviar un paquet DHCPDISCOVER així començant el procés anteriorment explicat per a obtenir la adreça IP.

Durant el procés d'obtenció de la adreça IP, en un dels paquets DHCP se li informarà al client de la adreça IP del servidor TFTP a on es podrà descarregar el NBP, el micro nucli (el nucli que enviem es una mica més gran de l'habitual ja que conté els drivers de diverses targetes de xarxa), per a iniciar el sistema de restauració d'imatges o arrencar algun dels nuclis prèviament descarregats en la màquina. En el nostre cas, no se li indicarà cap nova adreça al client ja que el servidor TFTP esta situat en la mateixa màquina que el servidor DHCP.

Així doncs quan el client arrenqui tindrà al CMOS la prioritat d'arrencar per xarxa. Llavors buscarà un servidor DHCP per a que li assigni una adreça i li indiqui des d'on s'ha de baixar el nucli per a arrencar.

NBP:

Tot i el tamany lleugerament superior del nucli, el nucli preparat és suficientment petit per a que el temps de descarrega d'aquest sigui imperceptible per a l'usuari. Les opcions de configuració bàsica que hauriem de marcar al make menuconfig de gentoo per a compilar el kernel són les següents:

*Code maturity level options --->
[*] Prompt for development and/or incomplete code/drivers*

*Device Drivers --->
[*] Networking support
Networking options --->
<*> Packet socket
<*> Unix domain sockets
[*] TCP/IP networking
[*] IP: multicasting
[*] IP: kernel level autoconfiguration
[*] IP: DHCP support (NEW)*

*File systems --->
Network File Systems --->
<*> file system support
[*] Provide NFSv3 client support
[*] Root file system on NFS*

Com aquest nucli esclau es genèric per tots els clients, s'ha de donar suport també per totes les targetes de xarxa i plaques base. Això es podria solucionar tenint un detector de hardware que demanes els mòduls necessaris al servidor. També es pot fer una configuració de pxe per cada client, i llavors un kernel per cada client.

Un cop descarregat el nucli, aquest muntarà com arrel del sistema, mitjançant NFS, un directori del servidor on hi haurà descomprimida un stage3 de Gentoo que li permetrà tindre al seu abast una gran quantitat d'eines i serveis. Les

quals ens permetran restaura les imatges.

5.1.2. Transmissió i creació d'imatges

Per a la transmissió i creació d'imatges utilitzarem tres eines que ara s'exposaran:zsplit,unzsplit i netcat.

Zsplit i unzsplit són utilitats de línia de comanda de linux que funcionen com el popularment conegut software Symantec Norton Ghost o Acronis true Image que permeten la creació d'imatges exactes de disc per a un backup complet del sistema o el clonatge d'un disc. En canvi, diferint d'altres solucions de software zsplit i unzsplit son lliures per al seu ús i distribució.

Zsplit i Unzsplit utilitzen la llibreria zlib que implementa les funcions de compressió i descompressió, incloent-hi funcions de comprovació de les dades descomprimides.

5.1.2.1. Creació d'imatge sobre dos discs

Per a entendre millor el ús d'aquestes eines primerament exposarem la situació hipotètica de crear una copia de seguretat des d'un disc a un altre de la mateixa màquina. En la computadora tenim un dispositiu anomenat hda amb una partició arrencable de Linux i la eina zsplit (Hem de considerar que arrenquem des de aquesta partició). A demés tenim un altre dispositiu anomenat hdb amb altres sistemes operatius en aquest, per exemple windows XP. S'ha de tenir en compte que el dispositiu hda a de tenir espai adicional suficient com per a poder contindre la totalitat de la imatge del dispositiu hdb.

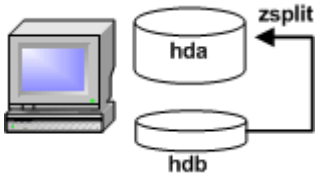

Fig. 5.1 Esquema de la creació d'imatges en un mateix PC

Asumint que arrenquem des de hda, seleccionem el directori on la imatge serà guardada i des d'aquest escrivim la següent comanda:

zsplit -N WinXP_backup -d /dev/hdb

5.1.2.2. Creació d'imatges sobre la xarxa

Suposant que volguessim crear una imatge de backup de un dispositiu de sistema arrencable i el sistema del qual volem fer la imatge no tingui un dispositiu de backup, però tinguem una segona computadora i aquestes estiguin conectades entre si en xarxa. Zsplit ajudat per netcat farà la feina de crear una imatge de backup des d'un dispositiu fins a l'altre mitjançant la xarxa.

Abans d'entrar en més detall sobre com fer aquesta operació explicarem les funcionalitats de netcat. Netcat és una utilitat que ens ofereix la possibilitat de treballar sobre la xarxa. Amb la qual podem llegir i escriure dades mitjançant coneccions de xarxa, utilitzant el protocol TCP/IP. Aquesta és una eina pensada per a la utilització directa del usuari. Així aquesta pot ser utilitzada fàcilment per altres programes o scripts. Al mateix temps és una eina rica per a l'exploració i correcció, ja que pot també crear qualsevol tipus de conecció que es necessiti.

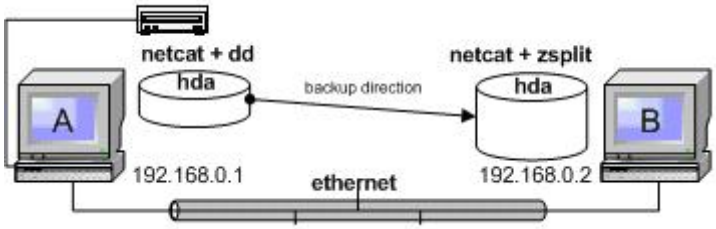

Fig. 5.1 Esquema de la creació d'imatges de un PC a un altre

La imatge de backup ha de ser creada des del dispositiu de la computadora A a través de la xarxa fins a algun dispositiu de la computadora B. Després de engegar ambdós computadores, hem de conèixer les adreces IP d'aquestes.En l'exemple assumim que la computadora A te l'adreça 192.168.0.1 i la computadora B te la adreça IP 192.168.0.2. Ara s'exposarà pas a pas les comandes per a la realització d'aquest procés :

Pas 1: A la màquina B introduïm les següent línia de comandes:

nc –l –p 9000 | zsplit –s 4.5G –N Linx_bkp –d

Aquesta comanda conectarà netcat en mode de escolta. Netcat estarà escoltant pel port 9000, si alguna dada arriba a aquest port, aquest enviarà la dada cap a zsplit. Zsplit fara la seva feina i comprimirà les dades en segments de 4,5 Gibibyte (Gygabyte binari) i guardarà les dades en el directori actual(els segments es guardaran de la forma següent: : Linx_bkp_0.spl.zp, Linx_bkp_1.spl.zp, Linx_bkp_2.spl.zp ...). En el directori també es podrà veure el arxiu debug-log amb el temps i la mida de la informació.

Pas 2: En la màquina A utilitzarem la següent comanda:

dd if=/dev/hda bs=1024 | nc 192.168.0.2 9000

Aquesta inicialitzarà el torrent de dades per la xarxa cap al port 9000 de la màquina amb IP 192.168.0.2. Depenen de la velocitat de la xarxa i la capacitat del disc dur de la màquina A variarà el temps de transferència de tota la informació cap a la màquina B. Com a resultat d'aquesta operació tindrem una imatge dividida en segments del dispositiu hda (de la computadora A) en la computadora B, ja que la feina de la comanda dd és la de enviar el arxiu cap a netcat. S'ha de dir que aquesta operació també es podria realitzar d'un altre manera. En la computadora B només s'hauria de activar el netcat en escolta redireccionat al directori on volem situar l'imatge. I en la A s'hauria de clickar la següent comanda: zsplit –s 4.5G –N –d /dev/hda | nc 192.168.0.2 9000. Així comprimint la imatge en el client i després enviant-la al servidor de backup.

5.1.2.3. Restauració d'imatges sobre la xarxa:

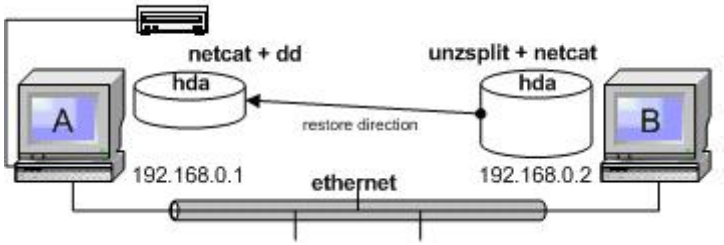

Fig. 5.3 Esquema de la restauració d'imatges d'un PC a un altre

Per a començar el procés de restauració s'han de seguir les següents passes.

Pas 1: En la màquina A executarem la següent comanda

nc -l -p 9000 | dd of=/dev/hda bs=1024

Netcat escoltarà per el port 9000 i aquest enviarà totes les dades a dd. I dd restaurarà totes les dades al dispositiu /dev/hda.

Pas 2: En la màquina B, dins el directori que conte la imatge teclegem la següent comanda:

unzsplit -c -d Linx_bkp | nc 192.168.0.1 9000

Unzsplit trobarà tots els arxius amb el nom comú Linx_bkp, determinarà el ordre correcte i la seqüència de lectura dels segments. Els descomprimirà i els enviarà a netcat. Netcat conectarà pel port 9000 amb la maquina la adreça de la qual es 192.168.0.1 i transferirà totes les dades a la màquina amb adreça IP

192.168.0.1. dd tornarà a restaura totes les dades al dispositiu hda. La opció –d controla la sortida del tamany i temps del procés de restauració en el arxiu debug.log.

5.1.3. Configuració de GNU GRUB

Les passes que necessitarem per a la realització de la eina GRUB seran: Posar GRUB com ha carregador actiu de l'arranc. Depenen de si hem instal·lat un nou sistema operatiu o hem instal·lat Linux i ens hem traslladat a GRUB farem diferents passos. Si comencem de nou, podem anar directament a la configuració de GRUB. Si hem instal·lat una distribució Linux haurem de instal·lar GRUB. Un cop hem instal·lat GRUB haurem de fer que assumeixi el control del MBR.

Com ha usuari de l'arrel, haurem de escriure: /boot/grub/grub. Aquesta comanda carregarà un BASH amb l'avís que actualment estem en GRUB:

grub> install (hd1,2)/boot/grub/stage1 (hd1) (hd1,2)/boot/grub/stage2 p (hd1,2)/boot/grub/menu.conf

Aquesta comanda utilitza la comanda de instal·lació de GRUB, que requereix la localització de la imatge de la 1ª etapa i la localització del MBR (install (hd1,2)/boot/grub/stage1 (hd1)). També la localització de la imatge de la segona etapa (install (hd1,2)/boot/grub/stage1 (hd1)) es requerida. També un paràmetre opcional p (hd1,2)/boot/grub/menu.conf informa a GRUB de la localització del archiu de configuració del menú de la GUI.

5.1.4. Arrencada del nucli restaurat

Un cop descarregada la imatge desitjada ens trobem que hem de arrencar aquesta sense reiniciar el sistema. S'ha de dir que un cop restaurada la imatge, si es treu el cable de xarxa del pc (per a evitar l'arranc per xarxa) i es reinicia la maquina. El nou nucli passaria automàticament a obtenir el control d'aquesta. Però com em d'evitar el màxim possible els inconvenients del sistema per al usuari utilitzarem kexec per a aquesta tasca. La utilització de kexec variarà depenen de la opció triada. En el cas que el sistema restaurat sigui Linux s'haura primerament de munta la imatge sobre el disc amb les comandes:

mount /dev/hda3 /mnt/otro
mount /dev/hda1 /mnt/otro/boot

Aquestes dues comandes el que fara es munta sobre el dispositiu /mnt/otro la imatge del sistema operatiu i el carregador d'arranc a /mnt/otro/boot. Quan acabem de muntar els dispositius exdecutarem la comanda:

Kexec –l /mnt/otro/boot/bzImage –append="root=/dev/hda3"

Així li passem els paràmetres de on esta la imatge del nucli i on esta l'arrel del sistema d'aquest i finalment executarem la comanda kexec –e per a carregar la nova imatge.

En cas que vulguem restaurar windows el procés serà una mica més complicat degut a que la localització del seu carregador d'arranc es desconeguda. Per això necessitarem fer un pas intermig que és la carrega de grub.exe per kexec. Perquè aquesta ens ajudi a poder carregar Windows. Així doncs haurem d'executar: kexec –l grub.exe ; kexec –e. Aquestes comandes carregaran grub, des de la qual podrem carregar windows.

5.2. Automàtització

Un cop carregada la "micro imatge" del nucli s'han creat varies aplicacions, de les quals en destacarem dues: una per al client i una per al servidor que donar lloc a l'obtenció d'un sistema flexible e interactiu per part de l'usuari final. Aquest programa en cas del client estar situat dins del procés init, és a dir, serà executat com a primer programa dins l'espai d'usuari.

Aquest programa mostrarà per pantalla un petit menu des d'on es dóna al usuari la possibilitat de triar entre quatre opcions: Restaurar Ubuntu, Restaurar Windows, Arrencar Ubuntu, Arrencar Windows.

5.2.1. Restaurar Ubuntu

Quan es tria aquesta opció per part de l'usuari, el programa que corre en la màquina del client executa un programa (fet amb scripting) que executa la comanda: nc –l -p 9000 | unzslip –D /dev/hda –d -. Aquesta comanda el que fa es copiar tota la informació que arriba per la xarxa a una pipe i tot seguit descomprimir-la al disc. Al mateix temps que executa aquesta comanda, envia un paquet TCP al servidor indicant-li la opció triada (en aquest cas la opcio 1). Llavors el servidor agafa la imatge del client corresponent a Ubuntu i li envia al client. Per a fer aquesta operació, un cop el programa resident al servidor hagi rebut l'opció triada pel client, executa un script. Aquest executa la comanda zsplit -0 –c –l /dev/hda | nc –l –p 192.168.1.4 9000. per a enviar la imatge. Un cop acabada de rebre la imatge i haver sigut descomprimida. S'interromp el procés de descarrega al client. Es munta la imatge al dispostiu corresponent. I s'executa la comanda kexec amb els paràmetres necessaris i el nou nucli agafa el control del sistema.

5.2.2. Restaurar Windows

Quan es tria aquesta opció per part de l'usuari, en la màquina client s'executa la comanda: nc –p -l 9000 | unzsplit –D /dev/hdb. Aquesta comanda el que fa es copiar tota la informació que arriba per la xarxa a una pipe i tot seguit

descomprimir-la. Alhora que fa això, paral.lelament envia un paquet al servidor indicant-li la opció triada. Llavors el servidor agafa la imatge del client corresponent a Windows i li envia al client. Un cop acabada de rebre la imatge i haver sigut descomprimida. S'executarà la comanda kexec amb els paràmetres de la imatge per a grub.exe. En aquest punt hem d'esmentar que l'automatització del programa acaba en aquest punt. Degut a que grub té de tenir accés al arxiu grub.conf per poder automatitzar la carrega del nucli i al utilitzar kexec no disposem d'aquesta informació, al només poder executar arxius en format ELF32. Al no tenir accés a l'arxiu de configuració s'han de introduir els paràmetres manualment. Aquests són els següents:

root (hd0,0)

rootnoverify (hd0,0)

#makeactive

#chainloader +1

#boot

Aquest problema es solventarà amb l'aparició de GRUB2 que permetra carregar dinàmicament els sistemes operatius dels dispositius. Així autodetectant de quin tipus de SO es tracta.

5.2.3. Arrencar Windows/Ubuntu

Quan es triï alguna d'aquestes opcions el programa client envia al servidor la opció triada i en vés de executar els scripts de restauració. El client executa els scripts de execució corresponents a cada SO. I el servidor tanca la conexió.

Els programes del client i del servidor son bàsicament un servidor tcp concurrent dissenyat inicialment per atendre a deu clients simultanis. I un client TCP. Per a entendre millor el funcionament del servidor aquí es mostra el diagrama d'estats de la interactuació bàsica del servidor-client TCP.

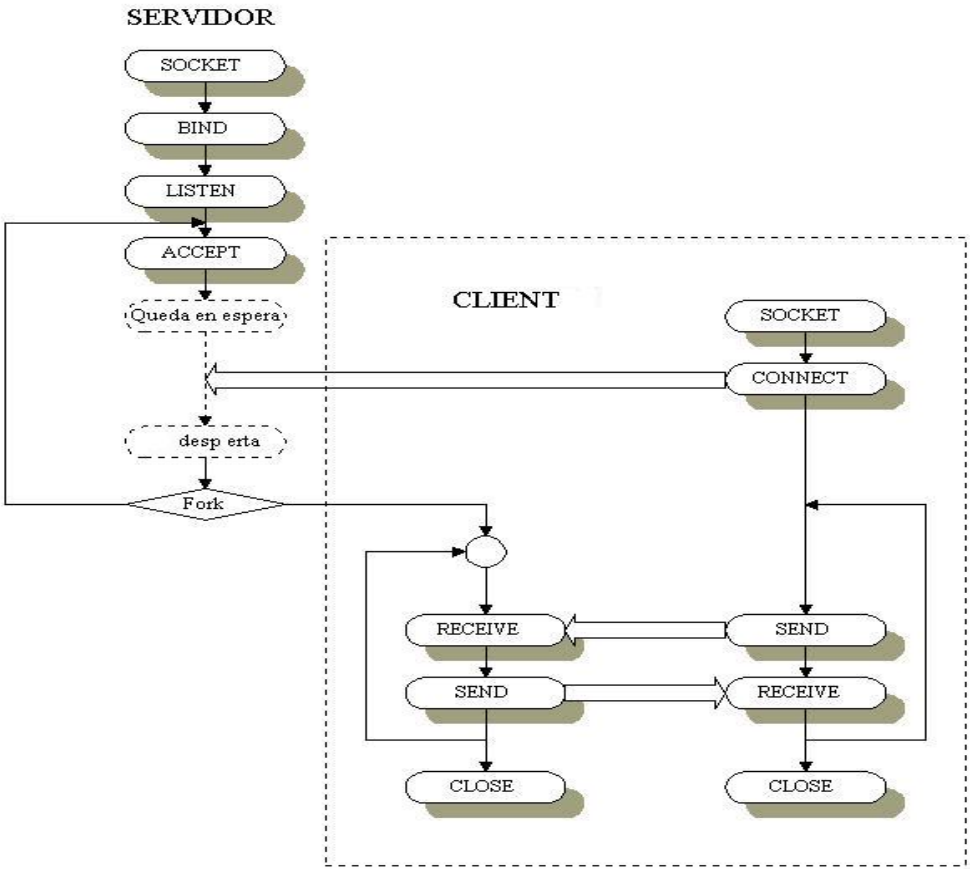

Fig. 5.4 Diagrama d'un servidor TCP concurrent

Per a entendre millor com funcionen des de dins els programes client servidor (El codi dels quals es pot veure a l'anex). Aquí es mostra un diagrama de l'estructura lògica d'aquests:

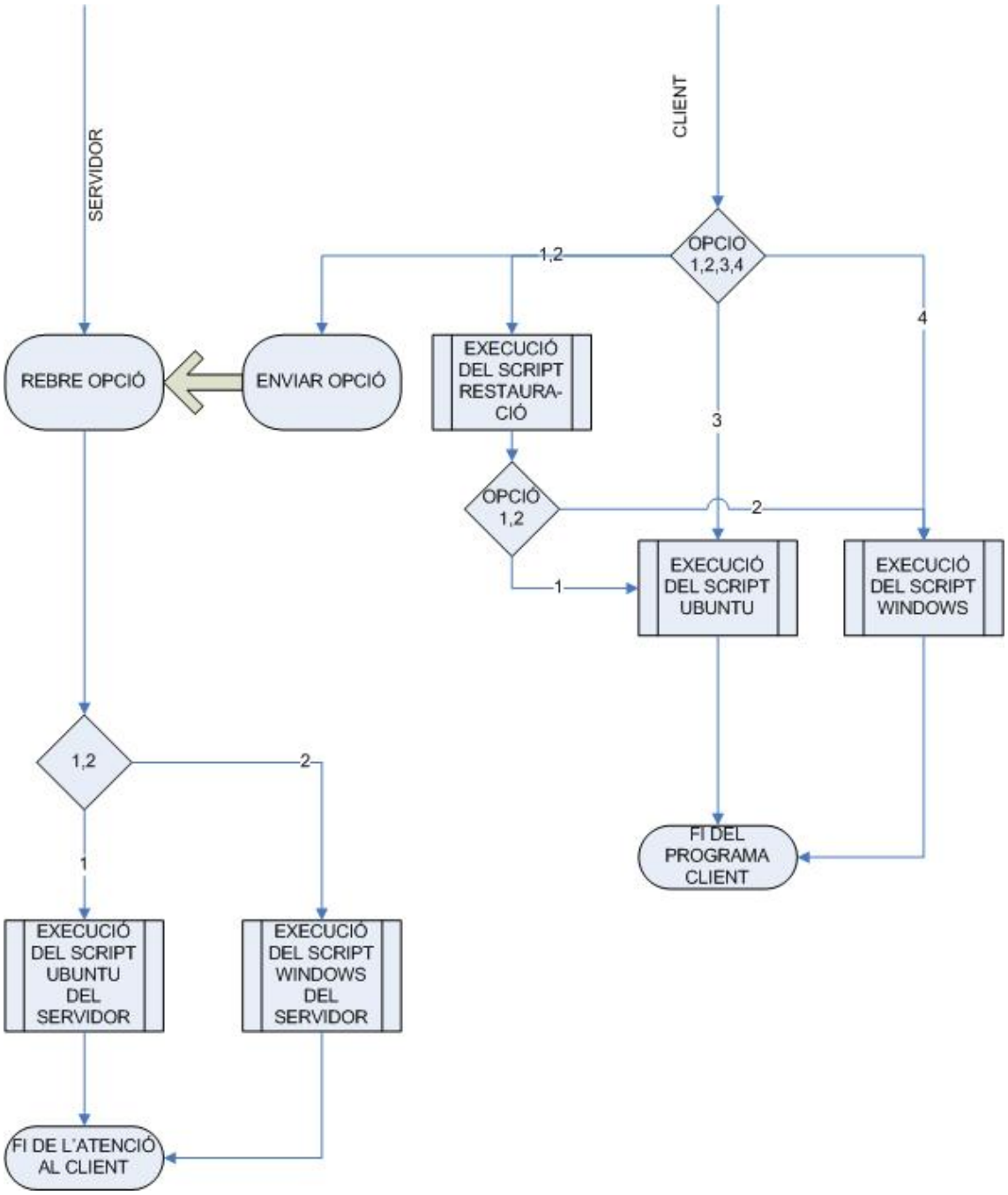

Fig. 5.5 Diagrama de l'estructura lògica del client - servidor

Per a aclarir el procés d'arranc del client a continuació es mostrà un diagrama de les fases per les que passa el client:

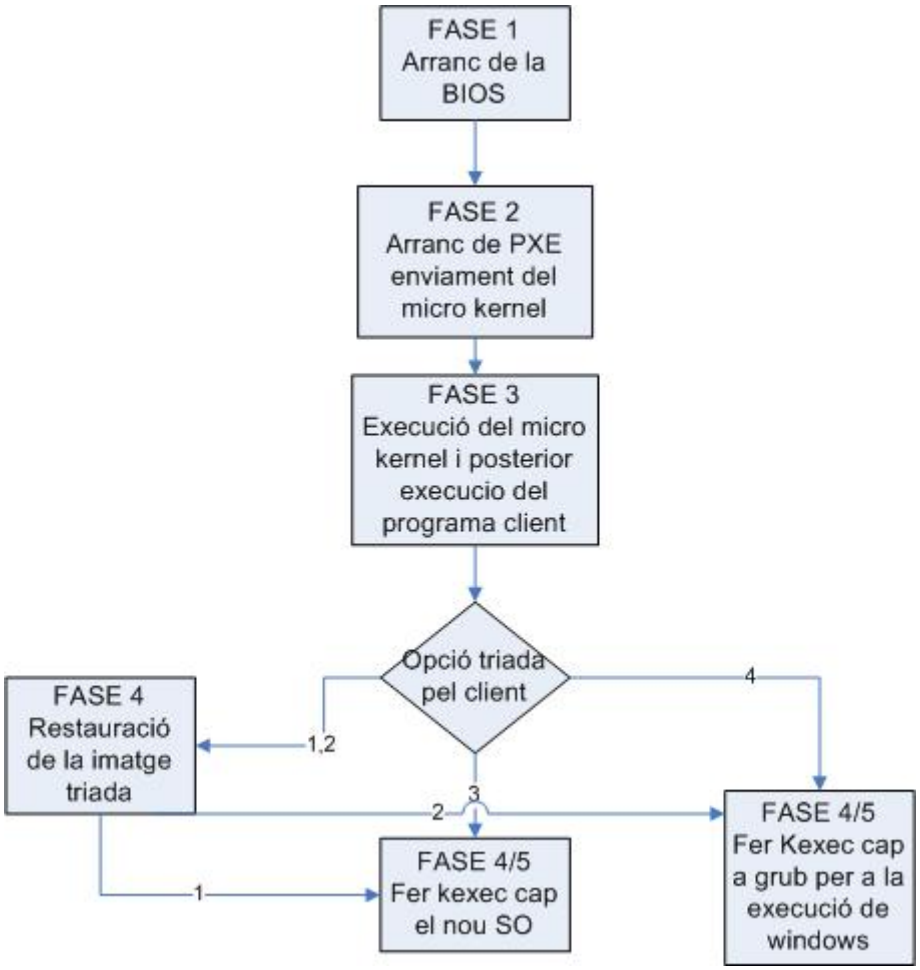

Fig. 5.6. Diagrama de les Fases del client

CAPÍTOL 6. PROVES EN EL PROCÉS DE RESTAURACIÓ

6.1. PXE

Durant la fase d'implementació de les tres tecnologies de PXE es van haver de variar alguns aspectes a mida que s'anava avançant. En el que correspon al ús de DHCP i TFTP només es va tenir que configurar correctament l'arxiu de configuració de DHCP. Que es va tindre que limitar a respondre a les adreces només si la MAC corresponia, per a no interrompre l'execució normal del servei DHCP de la universitat. I en el corresponent al TFTP només s'havia d'activar el servei que ve integrat en tots els nuclis de linux. Aquí veiem una captura del funcionament del servei DHCP:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xf39044ac
2	0.001116	192.168.1.1	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0xf39044ac
3	3.981088	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xf39044ac
4	3.982139	192.168.1.1	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0xf39044ac
5	3.982799	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xf49044ac
6	3.983547	192.168.1.1	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0xf49044ac
7	5.024955	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xf49044ac
8	5.026490	192.168.1.1	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0xf49044ac
9	5.027736	0.0.0.0	255.255.255.255	DHCP	DHCP Discover - Transaction ID 0xf59044ac
10	5.028374	192.168.1.1	255.255.255.255	DHCP	DHCP Offer - Transaction ID 0xf59044ac
11	9.034763	0.0.0.0	255.255.255.255	DHCP	DHCP Request - Transaction ID 0xf59044ac
12	9.035789	192.168.1.1	255.255.255.255	DHCP	DHCP ACK - Transaction ID 0xf59044ac
13	9.038186	AsustekC_90:44:ac	Broadcast	ARP	Who has 192.168.1.1? Tell 192.168.1.4
14	9.038265	BellTech_d2:ef:36	AsustekC_90:44:ac	ARP	192.168.1.1 is at 00:00:1c:d2:ef:36
15	9.038317	192.168.1.4	192.168.1.1	TFTP	Read Request, File: /pxelinux.0, Transfer type: octet

Fig. 6.1 Captura de pantalla de les transaccions inicials de DHCP

Com s'observa en la figura anterior, veiem com el client dhcp envia un paquet DHCPDISCOVER a la xarxa. El servidor DHCP al veure'l li respon amb un DHCPOFFER (oferint-li una adreça), el client li respon que accepta l'adreça assignada amb un DHCPREQUEST i finalment el servidor li respon amb un ACK per a confirmar-li. També observem que després de acceptar la adreça, el client envia un paquet ARP per a saber qui es el servidor TFTP en aquest cas el mateix servidor que el DHCP. Un cop el servidor li respon qui és, el client envia una petició de lectura del fitxer pxelinux.0 (El nucli per l' arranc).

En el que correspon a PXE es va tenir que recopilar diversos cops el nucli per a depurar el funcionament de l'aplicació, com compilar els drivers de la majoria de targes de les diferents targetes de xarxa de la universitat. O, i de forma molt important (fet que ja es veura més endavant), compilar els drivers de les

controladores DMA (són els drivers que faciliten l'accés d'escriptura lectura als disc-durs).

6.2. Creació i restauració d'imatges

Per a fer les probes de creació i restauració. Primer hem mesurat els temps de compresió i descompresió de la imatge en una mateixa màquina. S'ha de tindre en compte que el tamany de l'informació que enviem des d'un disc dur a un altre és de 10 GBytes. El processador que utilitzem es un Pentium III a 500 Mhz .

Prova de compresió realitzada cap al dispositiu /dev/null, amb factor de compresió 0.

#zsplit -0 -c –l -d /dev/hda >/dev/null

Tc = 21:09s
Velocitat mitja del procés = 443.32 MB/min
Prova de compressió / descompressió sobre discs diferents del mateix PC:

Factor de compressió : 0
Tt (Temps total)=34min:33s
Tc(Temps de compressió)=1530s=25min:30s
Td(Temps de descompressió) =543s = 9min : 3s
Velocitat mitja del procés = 282.732MB/min

Al comparar els resultats obtinguts entre la compressió i descompressió sobre disc amb la compressió sobre /dev/null, s'observa que el temps de compressió és una mica més elevat. Sabent que el temps d'escriptura sobre disc no fa de coll d'ampolla en el procés (Ja que aquest es superior a 50MB/s). Observem que al escriure sobre dos discs en el mateix PC les interrupcions d'escriptura lectura augmenten el temps de l'operació.

Prova de compressió/descompressió sobre discs de diferents PC(sobre xarxa):

```
19-09-2006 18:03:20 zsplit: version: 1.2.0
19-09-2006 18:03:21 zsplit: start of operation
19-09-2006 18:03:21 zsplit: read/write buffer size: 8 kibibyte (8192 bytes).
19-09-2006 18:03:21 zsplit: compression level: 0.
19-09-2006 18:03:21 zsplit: file/device-image cannot be splitted, size of chunk
is unknown, one chunk image will be created
19-09-2006 18:39:01 zsplit: bytes read from /dev/hda: 10242892800; written to :
10244976736; elapsed time: 00hr:35min:40sec
19-09-2006 18:39:02 zsplit: total sum of  bytes read: 10242892800;
written: 10244976736; elapsed time: 00hr:35min:40sec
19-09-2006 18:39:02 zsplit: average speed of imaging: 273.88MiB/min
19-09-2006 18:39:02 zsplit: Success!...
19-09-2006 18:39:02 zsplit: end of operation
```

Fig. 6.2 Fitxer debug.log de zsplit

Client:

#nc –l –p 9000 | unzsplit –c-l /dev/hda –d -

Servidor:

#zsplit -0 –l –c /dev/hda | nc 192.168.1.4 –p 9000

Tt= 35min:40s
Tc=26min:1s
Td=9min:31s

Aquí es pot observar que al transmetres per la xarxa s'introdueix una mica de retard en la finalització de l'operació. Això es degut a que la transmissió per TCP introdueix cert retard al tindre que confirmar els paquets rebuts.

Prova de compressió / descompressió sobre discs diferents del mateix PC. Amb factor de compressió 3.

#zsplit -0 -c –l -d /dev/hda >/dev/hdb

Tt= 1h:12min:54s
Tc=1h:01min:40s
Td= 11min:14s

Com s'ha pogut veure al tindre un processador massa lent per a fer la compressió de manera eficient s'ha pres la decisió de no comprimir les dades per a enviar-les. També observem que el temps de descompressió es relativament petit. Així doncs si disposéssim de dispositius d'emmagatzematge per a poder tindre les imatges dels SO, seria factible enviar les dades comprimides.

S'ha de dir que al principi aquesta operació durava més de cinc hores. La dràstica reducció d'aquest temps es va solucionar compilant de nou el nucli que enviem per arrencar la màquina client i donant-li suport per a la controladora DMA de la seva placa base.

Com podem observar a les proves realitzades hi ha una gran divergència entre el temps de compressió i descompressió fet que resulta inusual ja que els processos corren en paral.lel a la CPU. Això es degut a que les mesures es fan sobre el temps total de procés i no corresponen al temps real de CPU consumit pels processos. La major part del temps que identifiquem als resultats com a temps de compressió es realment temps de lectura/escriptura al disc (factor de compressió 0). Així doncs la estimació que s'extreu d'aquests resultats seria un temps de compressió similar al de descompressió (9 min) i el temps afegit seria el de lectura/escriptura (14 min).

CONCLUSIONS

A data d'avui,i des de que es va plantejar la consecució del sistema de restauració remot a mitjans de febrer d'aquest mateix any, s'han pogut assolir els objectius marcats i definits en la introducció de la present memòria:

- S'ha implementat un sistema de restauració remot amb els mateixos serveis per al client que les actuals versions de software privat de restauració d'imatges.

- El disseny d'un sistema de restauració remot basat en la utilització d'eines GNU amb la conseqüent reducció de costos. Gracies a les llicencies GPL de Linux es possible introduir al mercat una solució amb les mateixes característiques que les existents.

- Fer un sistema vàlid per a una xarxa heterogènia. On no importin ni el tipus d'equips ni els sistemes operatius emprats per cadascuna de les màquines.

- Crear des de l'inici fins al final del procés de restauració un sistema consistent .

- El sistema dissenyat no suposa cap repercussió mediambiental, excloent-hi el consum d'energia elèctrica per part del servidor.

-

El sistema de restauració remot assolit es per tant una solució robusta,flexible, econòmica i amb un gran potencial. Encara que actualment es podrien introduir unes millores per a poder posar-lo al mercat. Aquestes serien les següents:

- Introduir un entorn gràfic per a la presentació del menu.

- Dintre de les funcions d'administrador. Poder tindre un recull de les restauracions fetes per part dels usuaris.

- Introduir un Wins proxy als servidor per a poder variar el nom dels clients i poder validar els clients dins un domini.

- Amb els nous avenços en l'escriptura sobre NTFS, millora el temps de restauració de imatges, fent comprovació i enviament de petits troços de la imatge en vés de una imatge completa.

BIBLIOGRAFIA

[1] http://www.arcovia.com/es/comparativalinux/

[2] El tutorial "Getting to know GRUB" (developerWorks, January 2001)

[3] El tutorial "Build a Linux test network" (developerWorks, May 2003)

[4] "Boot Linux from a FireWire device" (developerWorks, July 2004)

[5] "Tip: Dual-booting Linux" (developerWorks, April 2002)

[6] "Lightweight Linux, Part 1" (developerWorks, October 2002)

[7] http://www.xmission.com/~ebiederm/files/kexec/

[8] http://www.xmission.com/~ebiederm/files/kexec/linux-2.5.48.x86kexec.diff

[9] http://www.xmission.com/~ebiederm/files/kexec/linux-2.5.48.x86kexec-hwfixes.diff

[10] http://www.xmission.com/~ebiederm/files/kexec/kexec-tools-1.8.tar.gz

[11] http://www.gentoo.org

[12] http://www.starlinux.net

[13] http://www.ibm.com

[14] http://www.device-image.de/main_docu.htm

[15] http://www.linuxinfor.com/spanish/man4/initrd.html

[16] Introducció als sistemes operatius – Teodor Jové Lagunas, Josep Lluís Marzo Lázaro, Dolors Royo Vallés

[17] http://www.arquired.es/users/aldelgado/doc/nfscoda/nfscoda.DOC-000.htm

[18] http://es.tldp.org/Manuales-LuCAS/doc-unixsec/unixsec-html/node53.html

[19] Manual de programació en c http://www.modelo.edu.mx/univ/virtech/prograc/clecesc.htm

[20] Manual de programació en c http://www.maxitrucos.com/Manuales/M_programacion_c_2.htm

Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXES

TÍTOL DEL TFC/PFC

TITULACIÓ:

AUTOR: Manuel Pérez Pérez

DIRECTOR: José González González

DATA: 25 de febrer de 2003

ANEXE 1. CODI DEL SERVIDOR TCP CONCURRENT

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/wait.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<errno.h>

#define MYPORT 3490 //port al que es conectaramn els clients
#define BACKLOG 10 // Grandaria de la cua al servidor

main(){

      int sockfd;
      int newfd;
      int n;
      char buffer[256],ip[256];
      int opcio = 0;
      struct sockaddr_in my_addr;
      struct sockaddr_in their_addr;
      int  sin_size;
      if (( sockfd = socket(AF_INET,SOCK_STREAM,0)) == -1){
            perror("socket");
            exit(1);
      }
      my_addr.sin_family = AF_INET;
      my_addr.sin_port = htons(MYPORT);
      my_addr.sin_addr.s_addr = INADDR_ANY;
      bzero( &(my_addr.sin_zero),8);
      if(bind(sockfd,(struct sockaddr *)&my_addr,sizeof(struct
sockaddr)) == -1){
            perror("bind");
            exit(1);
      }
      if ( listen(sockfd,BACKLOG)==-1){
            perror("listen");
            exit(1);
      }
      while(1){
            sin_size = sizeof(struct sockaddr_in);
            if (( newfd = accept(sockfd,(struct sockaddr
*)&their_addr,&sin_size)) == -1){
                  perror("accept");
                  continue;
            }
            printf("server: conexion desde:
%s\n",inet_ntoa(their_addr.sin_addr));
            if(!fork()){
                  //ip = inet_ntoa(their_addr.sin_addr);
                  printf("forkeando\n");
                  bzero(buffer,256);
```

```
                  n=read(newfd,buffer,256);
                  printf("%s\n",buffer);
                  if(n<0)
                  printf("No es pot llegir l'opcio del client\n");
                  n=strcmp(buffer,"1");
                  printf("%d\n",n);
            while(opcio<=0){
                  if(strcmp(buffer,"1")==0)
                        opcio=1;
                  else if(strcmp(buffer,"2")==0)
                        opcio=2;
                  switch(opcio){
                        case 1:
                              printf("1\n");

      execlp("/root/scripts/servidor/gentoo","gentoo",NULL);
                              break;
                        case 2:
                              printf("2\n");

execlp("/root/scripts/servidor/windows","windows",NULL);
                              break;

                        default:
                              printf("Adeu\n");
                              break;
                  }

            }
            close(newfd);
            while(waitpid(-1,NULL,WNOHANG)>0);
      }
}
```

ANEXE 2. CODI DEL CLIENT TCP

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<sys/wait.h>
#include<netinet/in.h>
#include<netdb.h>
#include<arpa/inet.h>
#include<stdlib.h>
#include<string.h>
#include<unistd.h>
#include<pthread.h>
#include<errno.h>


#define PORT 3490
#define MAXDATASIZE 100

int main (int argc,char *argv[]){

      FILE *fd;
      int sockfd,numbytes;
      int n,opcio=0,largo,result,op,ii=0;
      char *c,buf[MAXDATASIZE];
      char buffer[256],nombre[256],paraula[256];
      char host[256]="192.168.1.1";
      struct hostent *he;
      struct sockaddr_in their_addr;

      //Obrim el socket
      sockfd= socket(AF_INET, SOCK_STREAM, 0);
      //Verifiquem que s'ha obert correctament
      if(sockfd<0)
            error("No hem pogut crea el socket");
      he = gethostbyname(host);

      if (he==NULL)
      {
            error("No es pot trobar el servidor");
            exit(0);
      }

      //Inicialitzem la estructura i copiem la info adecuada
      bzero((char *) &their_addr,sizeof(their_addr));
      their_addr.sin_family = AF_INET;
      bcopy((char *)he->h_addr,(char *)&their_addr.sin_addr.s_addr,
he->h_length );

      their_addr.sin_port = htons(PORT);
      //conectem els sockets
      if(connect(sockfd, &their_addr, sizeof(their_addr)) <0){
                        error("No es pot conectar amb el servidor\n");
                        exit(0);
                  }
                  result = gethostname(&nombre,&largo);
                  if(result!=0)
                  {
                  error("NO es pot obtenir el nom del host");
```

```
                exit(0);
                }


        while(opcio <= 0  ){
                printf("Escull una de les seguents opcions(1,2,3,4)\n");
                printf("1.-Restaura Ubuntu\n");
                printf("2.-Restaura Windows\n");
                printf("3.-Arrenca Ubuntu\n");
                printf("4.-Arrenca Windows\n");
                bzero(buffer,256);
                scanf("%d",&op);
                printf("opcio %d\n",op);
                opcio=op;
                printf("opcio %d\n",opcio);
                if(opcio==1)
                        strcpy(buffer,"1");
                else if(opcio==2)
                        strcpy(buffer,"2");
                else if(opcio==3)
                        strcpy(buffer,"3");
                else if(opcio==4)
                        strcpy(buffer,"4");
                printf("%d\n",opcio);
        }

        switch(opcio){
                case 1:
                        printf("Restauracio Ubuntu ( Temps estimat 8 min
)\n");
                        if(!fork()){
                                if(!fork()){
                                        while(ii<=8){
                                                ii=0;
                                                sleep(10);
                                                fd = fopen("debug.log","r");
                                                printf("ii=%d\n",ii);
                                                do{
                                                        printf("ii=%d\n",ii);
                                                        c = fgets(paraula,256,fd);
                                                        if(c!=NULL)
                                                                printf("%s",paraula);
                                                        ii++;

                                                }while(c!=NULL);
                                                fclose(fd);

                                        }
                                        printf("gentoo\n");

        execlp("/scripts/gentoo1","gentoo1",NULL);

                                }
                                system("rm /debug.log");

        execlp("/scripts/restauracio2","restauracio2",NULL);
                        }
                        n=write(sockfd,buffer,strlen(buffer));
                        if(n<0){
                                printf("Error al escriure\n");
                                exit(1);
```

```
                }
                wait(NULL);
                break;
        case 2:
                printf("Restauracio Windows ( Temps estimat 18 min
)\n");
                if(!fork()){
                        if(!fork()){
                                while(ii<=8){
                                        ii=0;
                                        sleep(10);
                                        fd = fopen("debug.log","r");
                                        printf("ii=%d\n",ii);
                                        do{
                                                //printf("ii=%d\n",ii);
                                                c = fgets(paraula,256,fd);
                                                if(c!=NULL)
                                                        //printf("%s",paraula);
                                                ii++;
                                        }while(c!=NULL);
                                        fclose(fd);

                                }

        execlp("/scripts/windows","windows",NULL);

                                }
                                system("rm /debug.log");

        execlp("/scripts/restauracio1","restauracio1",NULL);
                }
                n=write(sockfd,buffer,strlen(buffer));
                if(n<0){
                        printf("Error al escriure\n");
                        exit(1);
                }
                wait(NULL);
                break;
        case 3:
                printf("Arrancada Ubuntu\n");
                printf("Muntant els dispositius...\n");
                //system("./scripts/gentoo1");
                break;
        case 4:
                printf("Arrancada Windows\n");
                system("./scripts/windows");
                break;
        default:
                break;
    }
    printf("FINAL\n");
    close(sockfd);
    return 0;
}
```