Grupa 441A

Temă de casă

Software de sistem şi Comparaţie Linux-Windows

Implementarea Mecanismelor de I/E

Cuprins :

Bibliografiile aferente se găsesc la sfârşitul fiecărui capitol si/sau în subsolul paginilor

1.Introducere

Diferiţi specialişti privesc mecanismele de I/E (intrare/ieşire) în diferite moduri. Inginerii care au terminat specializarea de Microelectronică le privesc ca un asamblu de cip-uri, conexiuni, alimentări şi alte componente fizice din care e creat hardware-ul. Studenţii care termină specializarea Ingineria Informaţiei se uită dupa comenzile pe care partea de hardware le acceptă, la funcţiile care sunt îndeplinite de aceasta şi la eventualele erori ce pot apărea. În acest referat se va pune accent pe programarea dispozitivelor de I/E şi nu pe proiectarea fizică a lor.

Dispozitivele I/E pot fi împarţite în 2 categorii: dispozitive care lucrează cu blocuri de date adresabile de dimensiuni fixe (de genul harddisk-urilor) şi dispozitive care lucrează cu şiruri de date care nu sunt adresabile (de genul mouse-ului,imprimantelor,interfeţelor de reţea, etc). Acestea se mai numesc în limbaj de specialitate ca: „block devices” şi „character devices”. Lucrul cu dispozitivele din ultima categorie este o problemă de programare care nu este de obicei rezolvată în limbaje de programare înalte; sistemul de operare prin intermediul sistemului de fişiere şi al unor locaţii din memorie (în care se vor pune datele ce vin de la dispozitivele care lucrează cu şiruri) va furniza datele diferitelor programe care rulează la un moment dat. Practic şi aceste dispozitive se abstractizează ca fiind tot dispozitive care lucrează cu blocuri de date. La ambele categorii de dispozitive se vor folosi pentru acestă „abstractizare” software-uri speciale numite *drivere dispozitiv.* Acestea se interpun între software-ul sistemului de operare şi dispozitivele hardware şi sunt de obicei scrise chiar de producătorul dispozitivului. [1]

1 Andrew S. Tanenbaum, Operating Systems - Design and Implementation, 3rd Edition, P.Hall 2006

2.Colecţia de drivere dispozitiv la Linux

Linux suportă mii de dispozitive hardware dar totuşi reuşeşte să păstreze dimensiunea kernel-ului destul de mică deoarece păstrează încărcat în memoria kernel-ului „activ" decât un mic set de drivere. Dacă sunt necesare conectarea altor dispozitive I/E se folosesc aşa zisele *module încărcabile.* Acestea se pot încărca şi rula la cerere (sau automat în funcţie de complexitatea distribuţiei pe care o avem instalată). [4]

Un driver dispozitiv (în Linux) e definit ca fiind codul scris permanent în kernel (şi deci care rulează mereu la fiecare bootare a kernel-ului) al cărui rol este de a intermedia interacţia între aplicaţii şi o anumita componentă hardware(sau virtuală). Un modul încarcabil are acelaşi rol ca un driver dar este încarcat şi rulat decât la cerere. [4]

Spre deosebire de Windows sistemele de operare bazate pe arhitectura Unix se bazează mult mai mult pe manipularea fişierelor. De aceea şi dispozitivele I/E vor fi abstractizate tot ca nişte fişiere în spaţiul de lucru. Un exemplu legat de acest lucru ar fi faptul că prin comanda write() se poate scrie atât într-un fişier normal dar se poate şi trimite la o imprimantă dacă se scrie în fişierul special /dev/lp0 (dacă se afla o imprimantă la portul la care e asignat lp0). [2]

Fişierele de drivere dispozitiv au 4 atribute importante:

- Numele – numele fişierului din sistemul virtual de fişierul; a nu se confunda cu pid (numele procesului)
- Tipul – fie pentru dispozitive ce lucrează cu blocuri de date, fie pentru dispozitive care lucrează cu şiruri de date.
- Număr Major – un număr între 1 şi 255 prin care se poate identifica tipul dispozitivului controlat; mai multe fişiere pot avea acelaşi număr major
- Număr Minor – un număr ce diferenţiază un dispozitiv atunci când există mai multe dispozitive care au acelaşi numar major (sunt de acelaşi tip)

Comanda mknod( ) este folosită ca să creeze un fişier device. Ea trebuie să primească ca parametri: numele fişierului, tipul, numarul major şi cel minor. Ultimii 2 parametri sunt regăsiţi în acelaşi număr pe 16 biţi notat „dev_t", primii 8 biţi sunt rezervaţi pentru numărul major iar ceilalţi 8 pentru cel minor. Există algoritmi scrişi pentru extragerea numerelor macro şi minor iar comanda „mkdev" e folosită ca să unească cele 2 numere. Numarul dev_t este folosit pentru identificarea cererilor făcute de către aplicaţiile utilizatorului; sistemul de operare foloseşte pentru

42 Daniel Bovet şi Marco Cesati ,Understanding the Linux Kernel, Editura Oreilly , 2000

4 Chritopher Negus, Linux Bible 2007 Edition, Editura Wiley Publishing, 2007

apelare numărul kdev_t(dev_t la care se ataşează şi micro-pid-ul procesului driver). Din Linux 2.2 kdev_t este folosit în ambele cazuri. [2]

Fişierele device sunt de obicei incluse în directorul /dev. În tabelul următoare sunt câteva exemple de fişiere device care vin cu Linux. Cu C s-au notat dispozitivele care lucrează cu şiruri de date iar cu B s-au notat dispozitivele care lucrează cu blocuri de date. [2]

Nume	Tip	Nr.Major	Nr.Minor	Descriere
/dev/fd0	B	2	0	floppy-disk
/dev/hda	B	3	0	hard-disk 1
/dev/hda1	B	3	2	Prima parţie al hard-disk-ului 1
/dev/hdb	B	3	64	hard-disk 1
/dev/hdb1	B	3	67	Prima parţie al hard-disk-ului 2
/dev/ttyp0	C	3	0	Terminalul principal
/dev/console	C	5	1	Consola sistemului
/dev/lp1	C	4	1	Imprimantă pe port paralel
/dev/rtc	C	10	135	Ceas-ul sistemului
/dev/ttyS0	C	6	64	Primul port serial
/dev/null	C	1	3	Dispozitiv nul

De obicei un fişier device este asociat unui dispozitiv fizic sau unei subunităţi al acestuia (de exemplu `/dev/hda1` e asociat primei partiţii a unui hard). Dar în unele cazuri un fişier nu e ataşat unui dispozitiv real ci unui dispozitiv fictiv, care nu există decât pe plan logic. De exemplu /dev/null este un device care va corespunde unei „găuri negre", adica toate fişierele trimise la el nu sunt luate în considerare şi fişierul /dev/null pare a fi mereu gol. [2]

Merită de asemenea de menţionat că tipul fişierelor device se poate afla din atributele fişierului. Aşa cum se ştie comanda ls ne arată conţinutul directorului curent şi atributele fiecăruit fişier afişat:

```
#ls /root
drwx------    2 dee      users      4096       Jul 29 07:48 .
drwxr-xr-x    5 root     root       4096       Jul 27 11:57 ..
-rw-r--r--    1 dee      users      24         Jul 27 06:50 .bash_logout
```

2 Daniel Bovet şi Marco Cesati ,Understanding the Linux Kernel, Editura Oreilly , 2000

```
-rw-r--r--    1 dee    users    230    Jul 27
06:50 .bash_profile
-rw-r--r--    1 dee    users    124    Jul 27
06:50 .bashrc
-rw-rw-r--    1 dee    users    0      Jul 29
07:48 lsfile
.......
```

Zamfir Narcis 441A

Prima literă ne arată tipul fişierului. Următoarele (rw-r--r--) ne arată permisiunile diferiţilor utilizator asupra acestui fişier. Astfel avem:

- \- = Fişier normal
- b = Fişier device ataşat unui dispozitiv care lucrează cu blocuri de date
- c = Fişier device ataşat unui dispozitiv care lucrează cu şiruri de date
- d = Director
- i = Legătura la un fişier [4]

Atributele unui fişier device ataşat unui dispozitiv care lucrează cu blocuri de date sau şiruri de date ar arăta aşa:

#ls root/mnt/

```
br--r--r--    1 dee    users    0      Jul
29 07:55 cdrom
crw-rw-r--    1 dee    users    0      Jul 29
07:43 net_2pci
....
```

Mai sus prin mnt/cdrom se pot citi date de pe un disc din unitatea CDROM a sistemului iar prin placa de reţea (mnt/net_2pci) se pot primi sau scrie date de la un alt calculator din LAN.[5]

După cum se observă din tabelul de pe pagina precendetă denumirile asociate dispozitivelor sunt oarecum tipizate. Un hard-disk pe interfaţa IDE are asociat un driver al cărui nume începe cu /dev/hd; primul hard-disk detectat are asociat litera a, al doilea litera b şi aşa mai departe. De exemplu al 3-lea hard-disk IDE ar avea asociat /dev/hdc. Dacă se foloseşte un „Controller IDE" separat (pe interfaţă PCI, etc) atunci hard-disk-urile controlate de el vor fi puse după cele detectat de placa de bază şi cu denumiri de la litera e încolo. [Ex: Dacă avem 3 hdd-

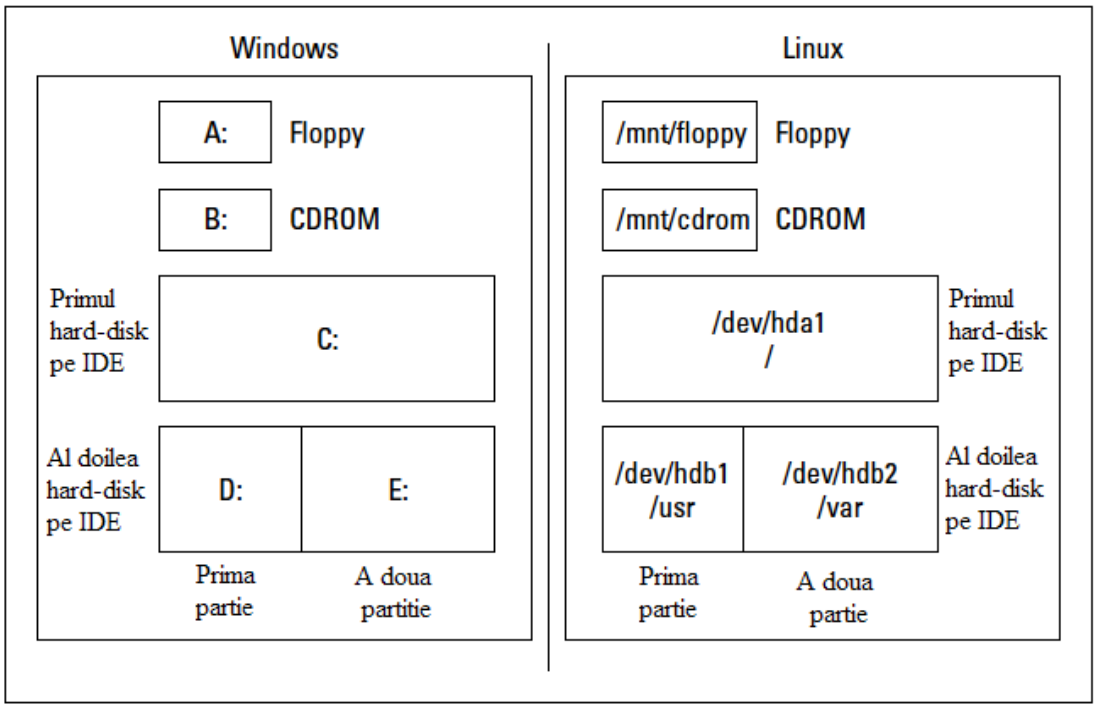
4 Chritopher Negus, Linux Bible 2007 Edition, Editura Wiley Publishing, 2007

5 Dee-Ann LeBlanc şi Richard Blum, Linux for Dummies, Ed Wiley, 2007

uri detectate de placa de bază acestea vor avea denumirile: hda,hdb,hdc; şi alte 2 hdd-uri detectate de controller vor avea denumirile: hde, hdf. Dacă avem 5 hdd-uri pe placa de bază: hda, hdb, hdc, hdd, hde atunci cele 2 hdd-uri de la controlor vor fi: hdf, hdg.]. Un hard-disk pe interfaţa SCSI are o denumirea asociată care începe cu /dev/sd; la fel ca la IDE, primul hard detectat are urmată litera a, al doilea b, etc; de exemplu al 4-lea hard SCSI detectat are asociat driver-ul cu denumirea dev/sdd. De asemea majoritatea distribuţiilor de Linux detectează dispozitivele de stocare pe USB ca fiind tot hdd-uri SCSI. De aceea dacă se conectează un memory stick pe usb la un calculator cu linux aceasta ar putea fi accesat de la /dev/sda1 (dacă nu există hdd-uri SCSI deja montate în sistem). Revenind la denumirea driverelor de hdd-uri: după cele 3 litere urmează un numar care reprezintă partiţia care este accesibilă. La majoritatea distribuţiilor

moderne de Linux dacă se cere să se scrie la /dev/sda/ atunci automat se scrie în prima partiţie cu destul spaţiu al partiţia respectivă, deci în /dev/sda1. [5]

Mai jos este o comparaţie între cum s-ar vedea diferite componente ce păstrează date în Linux şi Windows.



Deoarece s-a folosit Fedora (distribuţie destul de nouă de Linux) dev/fd0 a fost montat automat în mnt/floppy; analog dev/cdrom0 a fost montat automat în mnt/cdrom. Primul hard nu are decât o singură partiţie şi automat aceasta a fost alocată drept partiţia /root. Al doilea hard are

5 Dee-Ann LeBlanc şi Richard Blum, Linux for Dummies, Ed Wiley, 2007

două partiţii ale caror nume au fost alocate de către utilizator sau de către sistemul de operare ca: prima partiţie cu /usr iar a doua partiţie cu /var. [5]

Tot prin drivere dispozitiv se pot accesa şi porturile seriale. La Windows porturile seriale sunt denumite prin „COMx" (unde x este un număr). Această denumire se regăseşte mai nou şi în BIOS-ul plăcii de bază. În Linux fiecarui port serial îi corespunde o intrare în directorul /dev; primul port serial este /dev/ttys0, al doilea port serial este /dev/ttys1, etc. Trebuie ţinut cont că deşi în directorul /dev pot fi foarte multe porturi seriale nu înseamnă ca ele şi există fizic; acest lucru e necesar la crearea de maşini virtuale şi nu numai. Pentru legarea virtuală a unor porturi seriale se foloseşte comanda *setserial.* Aceasta nu face decât conexiunea virtuală, în interiorul sistemului de operare, nu şi cea fizică prin domeniu de adrese şi IRQ(aceasta se face fie fizic pe placa de bază, fie virtual prin software şi drivere specializate). Unele distribuţii de Linux creează un

device virtual /dev/modem; acesta ar trebui folosit pentru al lega la un port serial fizic (cu *setseriall /dev/ttysX*), pentru a uşura munca utilizatorului (această nu trebuie să ţină minte mereu, ce numar X are portul cu care lucrează). Acelaşi lucru se poate spune şi de portul serial de mouse : sistemul de operare Linux identifică primul mouse serial detectat şi leagă portul serial aferent lui la /dev/mouse. [6]

Configurarea porturilor seriale este puţin mai complicată în Linux decât în Windows unde acest lucru se făcea complet automat. Spre deosebire de majoritatea distribuţiilor noi de Linux care ataşează adresele şi setările IRQ corect tuturor porturilor seriale, versiunile mai vechi de Linux nu aveau acest lucru implementat. Pentru afişarea listelor de IRQ şi porturilor aferente se foloseşte comanda *dmesg*. Ex:

```
#dmesg
.....
Serial driver version 4.27 with HUB-6 MANY_PORTS
MULTIPORT SHARE_IRQ
ttyS00 at 0x03f8 (irq = 4) is a 16550A
ttyS01 at 0x02f8 (irq = 3) is a 16550A
.....
```

Managementul întreruperilor este făcut de sistemul de operare şi nu de driverele dispozitiv ale porturilor seriale. Linux poate acceta până la 16 dispozitive care cer întreruperi simutan. Astfel la un calculator normal avem următoarele setări pentru lista de IRQ:

---

6 Michael J. Tobler, Inside Linux, Ed. New Riders, 2004

- IRQ 0: Ceas(canal 0)
- IRQ 1: Tastatură
- IRQ 2: Controller în „cascadă” 2
- IRQ 3: Port Serial 2
- IRQ 4: Port Serial 1
- IRQ 5: Port Paralel 2, placa de sunet
- IRQ 6: Cititor de dischete (FDD)
- IRQ 7: Port Paralel 1
- IRQ 8: Ceas(-ul sistemului)
- IRQ 9: Redirecţionat la IRQ2
- IRQ 10: Nefolosit
- IRQ 11: Nefolosit
- IRQ 12: Nefolosit
- IRQ 13: Coprocesor aritmetic
- IRQ 14: Controller de hard-disk 1
- IRQ 15: Controller de hard-disk 2 [6]

De obicei porturile seriale ttyS0 şi ttyS2 folosesc IRQ4 iar ttyS1 şi ttyS3 folosesc IRQ3. În cazul de mai sus dacă vine semnal de întrerupere

de la IRQ 10,11,12 atunci acesta va fi ignorat. O listă cu IRQ-urile curente se găseşte afişând conţinutul directorului */proc/interrupts/*.

De exemplu (distribuţie Fedora):

#cat /proc/interrupts

```
IRQ   CPU0
0:       5461793 XT-PIC  timer
1:        103302 XT-PIC  keyboard
2:             0 XT-PIC  cascade
3:        270309 XT-PIC  serial
8:             2 XT-PIC  rtc
10:        76802 XT-PIC  ide0, ide2
12:       183043 XT-PIC  PS/2 Mouse
13:            1 XT-PIC  fpu
15:            4 XT-PIC  ide1
NMI:           0
```

[6]

Pentru dispozitive care nu sunt detectate sau care nu au drivere dispozitiv instalate se folosesc module încarcabile. Acestea sunt create din coduri sursă de drivere. Codurile sursă se găsesc în subdirectorul /usr/src/linux/drivers/ sau pe cd-ul distribuţiei. Dacă totuşi

6 Michael J. Tobler, Inside Linux, Ed. New Riders, 2004

distribuţia instalata nu are drivere compatibile coduri sursa de drivere se găsesc pe diferite site-uri de pe internet, Linux având o foarte mare comunitate de programatori amatori care îşi creeaza drivere proprii „open-source”. Nu în ultimul rând şi producătorii dispozitivelor respective oferă coduri sursă de drivere iar creatorii de distribuţii adaugă noi drivere dispozitiv kernel-ului de la o versiune la alta. După ce modulele sunt create (prin comanda *build* sau *make xconfig*) ele sunt instalate în directorul /lib/modules/. [4]

Pentru a vedea ce module sunt încarcate şi rulează acuma în kernel se foloseşte comanda lsmod. Un exemplu de rulare a comenzii lsmod în distribuţia Knoppix:

lsmod

Module	Size	Used by
snd_seq_oss	38912	0
snd_seq_midi_event	9344	1 snd_seq_oss
snd_seq	67728	4 snd_seq_oss
snd_seq_oss	8328	2 snd_seq
.		
autofs	16512	0
ne2k_pci	9056	0
8390	13568	1 ne2k_pci
ohci1394	41860	0
ieee1394	284464	1 ohci1394
floppy	65712	0
sg	36120	0
scsi_mod	124600	1 sg

Se observã cum deşi user-ul (terminalul 0) foloseşte doar modulul snd_seq_oss (placã de sunet de tip Alsa) acesta rulează în mod automat modulul snd_seq_oss care la rândul lui mai încarcã 2 module. Se mai observã ºi încãrcarea modului pentru conexiunea Firewire (ohci1394, ieee1394), placa de reþea (ne2k_pci, 8390) , control de harddisk-uri scsi (sg, scsi_mod) . Pentru aceastã versiune de Linux (Knoppix , 2007) cititorul de dischete nu are driver dispozitiv şi de aceea a fost necesarã încarcarea unui modul “floppy”. Pentru a afla informaþii despre un modul încãrcat se poate folosi comanda *modinfo.* Modulele se încarcã folosind comenzile *modprobe* şi *insmod* iar pentru a le opri se foloseşte comanda *rmmod* . Este recomandat să se oprească procesele(prin comanda *kill*, etc) care folosesc date care provin de la un modul care urmează sa fie oprit. [4]

Linux implementează driverele dispozitiv ca nişte procese independente care lucrează în spaţiul dedicat utilizatorului. Fiecare driver rulează folosind un spaţiu de adresare intern (de exemplu driver-ul de hard are un sistem de adresare de magistrală intern care nu este vizibil

4 Chritopher Negus, Linux Bible 2007 Edition, Editura Wiley Publishing, 2007

4 Chritopher Negus, Linux Bible 2007 Edition, Editura Wiley Publishing, 2007

sistemului de operare); acesta poate fi extins pentru a include mai multe dispozitive odată(de exemplu conexiunile seriale şi de cele de reţea sunt controlate de un singur proces de driver). În fişierele de bază ale Linux-ului se găsesc un set de dll-uri (librarii software = dynamic link libraries) care conţin un set de drivere generice pentru RAM, hard-disk şi dischetă (precum şi pentru alte dispozitive în funcţie de distribuţia de Linux folosită). Dacă este necesar ca pentru mai multe dispozitive de acelasi fel să se folosească acelaşi tip de driver atunci dll-ul respectiv este încărcat încă o dată în memorie şi va rula ca un proces independent. De obicei procesele de drivere în linux au privilegii limitate şi nu pot manipula întreruperi sau deschide porturi noi de I/E. Sistemul de operare procesează întreruperile şi trimite procesului driver un mesaj de notificare a apariţiei întreruperii. [3]

Lăsând la o parte ultimele distribuţii de Linux care conţin tehnologii la fel de avansate ca windows-urile(„semnarea digitală" a driverelor, downloadarea driverelor în mod automat de pe net, etc) versiunile mai noi de kernel de linux oferă destule îmbunătăţiri asupra sistemului de I/E. Dar de la Linux 2.6 deja se poate vorbi de un model de driver unificat. În versiunile anterioare modelele de drivere care venea cu Linux nu făceau decât cele mai de bază operaţii: alocă memorie în mod dinamic pentru dispozitiv, rezervă un anumit număr de adrese I/O şi verifică o linie IRQ pentru identificarea şi implementarea întreruperilor.

Zamfir Narcis 441A

De aceea programarea driverelor dispozitiv era anevoioasă şi astfel se obţineau perioade lungi între doua update-uri. [7]

Astfel cu rezultate similare ca cele din windows-uri Linux 2.6 are integrat drivere dispozitiv care au:

- control asupra managementului puterii (kernel-ul poate forţa dispozitivele într-un mod low-power; lucru ce este necesar pentru a ţine calculatorul în standby sau în hibernare.)
- plug and play (alocarea resurselor aferente dispozitivului în timp real şi într-un mod transparent proceselor necesare)
- hot-plugging (asemănător plug and play dar această tehnologie permite introducerea sau scoaterea unui dispozitiv în sistem când acesta funcţionează.)

Deşi directorul /dev se păstrează şi el driverele dispozitiv folosite de sistem sunt acuma păstrate în directorul /sys. În interiorul lui se regăsesc subdirectoare de clasificare a driverelor:

- /sys/block – dispozitive care lucrează cu blocuri de date, independent de bus-ul pe care se află acestea
- /sys/devices – toate dispozitivele recunoscute de kernel, organizate după bus-ul la care sunt conectate

3 Andrew S. Tanenbaum , Modern Operating Systems, 2nd Edition, , Editura Prentice Hall, 2002

- /sys/bus/ - bus-urile detectate de sistem
- /sys/drivers – driverele dispozitiv înregistrare în sistem (care sunt încarcate în memorie şi rulează în acel moment)
- /sys/class – clasele de dispozitive care se regăsesc în sistem (de ex. plăci audio,plăci video, plăci de reţea , etc)
- /sys/power – driverele dispozitiv necesare pentru managementul puterii
- /sys/firmware – fişierele necesare pentru a controla firmware-urilor anumitor dispozitive [7]

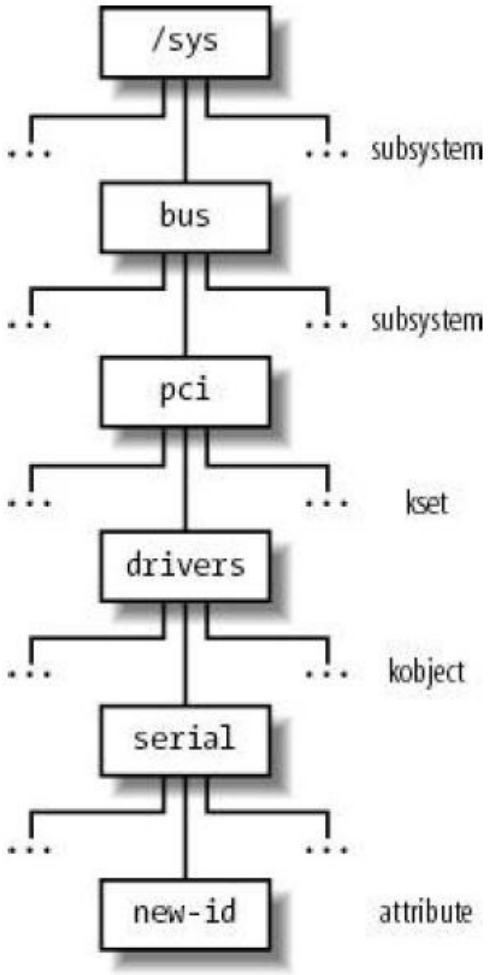
Se folosesc link-uri simbolice pentru a nu ocupa spaţiul într-un mod ineficent. De exemplu dispozitivul /sys/block/sda/device poate fi legat prin link direct la /sys/devices/pci0000:00 care reprezintă un controller SCSI conectat la primul slot Pci.

Structura unui driver dispozitiv a fost puternic tipizată din Linux 2.6 şi de aceea structura oricărui nucleu de driver poartă acum denumirea de „Kobject". Aceleaşi kobject poate fi instanţiat de mai multe ori dacă se doreşte controlarea unor dispozitive similare. [7]

Zamfir Narcis 441A

Mai jos este prezentată ierarhia pentru un dispozitiv serial de pe bus-ul pci denumit „new-id":

7 Daniel Bovet şi Marco Cesati ,Understanding the Linux Kernel 3rd Edition, Editura Oreilly , 2005



Fiecare dispozitiv este definit de cel puţin un obiect „device”. Aceasta este o structură care conţine informaţiile necesare pentru descrierea comportamentală (şi dacă e cazul şi funcţională) a dispozitului. Detalii referitoare la construcţia lui efectivă se pot găsi în diferite cărţi de specialitate. Pentru creearea lui sunt necesare atât cunoştinţe de programare obiect orientată avansate cât şi înţelegerea funcţionării dispozitivului în respectivul sistem. [7]

Fiecare driver este definit de un obiect „device_driver”. Din nou această structură este tipizată începând cu Linux 2.6 şi în mod obligatoriu conţine unul sau mai multe obiecte de tip „device”. Linux permite scriarea şi modificarea driverelor existenţi în sistem. Pentru introducerea în sistem al unui driver nou se foloseşte funcţia *driver_register( )* iar pentru a scoate un driver din driverele dispozitiv acceptate de sistem se foloseşte comanda *driver_unregister( ).* [7]

Zamfir Narcis 441A

Într-un mod similar se defineşte şi „bus-ul”. Obiectul care îl defineşte este de tip standardizat „bus_type” şi conţine obiecte „device” şi „device_driver”. În continuare în ierarhie se află clasa de drivere al

7 Daniel Bovet şi Marco Cesati ,Understanding the Linux Kernel 3rd Edition, Editura Oreilly , 2005

cărui obiect descriptiv „class_device” conţine obiecte „device” şi „device_driver” dar poate conţine şi obiecte „bus_type”. [7]

Pentru utilizarea eficientă a sistemului se recomandă folosirea dispozitivelor din directorul /dev/. Directorul /sys nu trebuie modificat decât de către administratori de sistem sau utilizatori cu pregătire avansată în Linux. Chiar şi versiunile cele mai noi de Linux nu oferă drivere complete pentru toate dispozitivele de I/E existente pe piaţa. Acelaşi driver poate controla diferite dispozitive mai mult sau mai puţin eficient. În general sunt 3 categorii la care un dispozitiv (detectat cu succes) poate fi suportat de Linux:

- Nesuportat – Aplicaţiile care rulează pot totuşi interacţiona cu porturile de intrare/ieşire ale dispozitivului (dacă acestea au fost detectate cu succes)
- Suport minim – Kernel-ul nu recunoaşte dispozitivul hardware dar recunoaşte interfaţa. Aplicaţiile care rulează pot interacţiona cu dispozitivul respectiv prin porturile de intrare/ieşire; de obicei într-un mod secvenţial.
- Suport extins – Kernel-ul recunoaşte dispozitul hardware şi poate controla operaţiile I/E chiar el. Acest lucru de obicei înseamnă ca dispozitivul are drivere compatibile printre driverele nucleu care se încarcă mereu cu linux.

Un alt aspect ce merită menţionat este îmbunătăţirea sistemului virtual de fişiere(VFS-ului). Acesta are din Linux 2.6 posibilitatea de a crea „cache-uri temporare” în ram/hdd/etc şi lucrează mult mai eficient cu driverele pentru dispozitive ce lucrează cu blocuri de date. În Linux aceste drivere (driverele pentru dispozitive ce lucrează cu blocuri de date) sunt considerate ca fiind elementul de bază(cea mai joasă componentă) în subsistemul de control al blocurilor de date(şi deci al VFS-ului). [7]

Mai jos este un tabel cu unele din driverele care vin cu Linux şi dispozitivele care sunt controlate de acestea.

root\drivers	Director ce contine drivere în Linux (2.6+)
acorn	dispozitive Acorn
acpi	Advanced Configuration Power Interface (managementul puterii)
atm	drivere pentru reţele ATM
block	drivere pentru dispozitive ce lucrează cu blocuri de date
paride	drivere pentru accesarea dispozitivelor IDE devices de la portul paralel
bluetooth	drivere pentru dispozitive Bluetooth
cdrom	drivere CD-ROM simple (nu ATAPI sau SCSI)
char	drivere pentru dispozitive ce lucrează cu şiruri de date
agp	drivere pentru placi video pe AGP
drm	driver pentru acceleratoare grafice compatibile XFree86
drm-4.0	driver(mai nou) pentru acceleratoare grafice compatibile XFree86

7 Daniel Bovet şi Marco Cesati ,Understanding the Linux Kernel 3rd Edition, Editura Oreilly , 2005

ftape	driver pentru dizpozitive de stocare pe benzi
ip2	controller serial Computone Intelliport II
joystick	drivere ptr Joystick-uri
mwave	driver ptr modem IMB (Winmodem-Linux)
pcmcia	drivere ptr dispozitive conectate pe pcmcia
rio	Driver for the Specialix Rio multiport serial card
dio	driver ptr Hewlett-Packard's HP300
fc4	drivere ptr controllere de fibra optica
hotplug	suport ptr hot-plugging pe diferite bus-uri
i2c	driver ptr bus-ul Philips i2c
ide	driver ptr bus-ul şi hdd-urile pe IDE
ieee1394	driver ptr bus-ul Firewire
input	Input layer module for joysticks, keyboards, and mouses
isdn	driver ptr modem isdn
usb	suport ptr bus-ul USB
video	drivere ptr acceleratoare grafice

8

Bibliografie:
(ptr Colecţia de drivere dispozitiv la Linux – Zamfir Narcis 441A)

1. Andrew S. Tanenbaum, Operating Systems ,Design and Implementation, 3rd Edition, Editura Prentice Hall 2006 , Capitolul 3: Input/Output
2. Daniel P. Bovet şi Marco Cesati ,Understanding the Linux Kernel, Editura Oreilly , 2000, Capitolul 13: Managing I/O Devices
3. Andrew S. Tanenbaum , Modern Operating Systems, 2nd Edition, , Editura Prentice Hall, 2002 , Capitolul 5: Input/Output
4. Chritopher Negus, Linux Bible, 2007 Edition, Editura Wiley Publishing, 2007, PartII,Capitolul4: Learning Basic Administration, şi alte cap.
5. Dee-Ann LeBlanc şi Richard Blum, Linux for Dummies 8th Edition, Editura Wiley Publishing, 2007, Part II şi III
6. Michael J. Tobler, Inside Linux, Editura New Riders, 2004, Part I şi II
7. Daniel P. Bovet şi Marco Cesati ,Understanding the Linux Kernel 3rd Edition, Editura Oreilly , 2005 , Capitolul 13: I/O Architecture and Device Drivers; Capitolul 14: Block Device Drivers
8. Daniel P. Bovet şi Marco Cesati ,Understanding the Linux Kernel 2nd Edition, Editura Oreilly , 2003, Anexa C

Bălan Alexandru 441A

3.Colectia de drivere dispozitiv la Windows

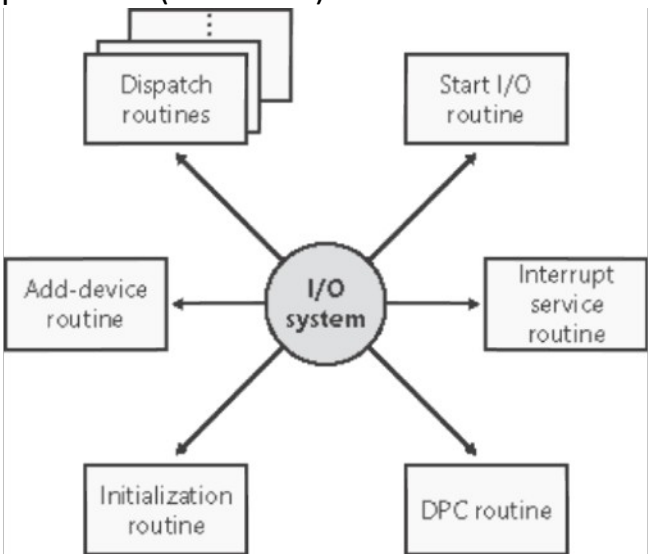
Driver dispozitiv - este o componenta pe care windows-ul o utilizeaza pentru a interactiona cu dispozitive externe ca modemul sau placa de retea. In loc sa acceseze direct dispozitivul, windows-ul incarca

8 Daniel Bovet şi Marco Cesati ,Understanding the Linux Kernel 2nd Edition, Editura Oreilly , 2003

driverele dispozitiv, apeleaza functiile driverului pentru a trimite dispozitivului sarcinile ce trebuiesc indeplinite. Functiile driverului contin codul specific dispozitivului.

Structura unui driver

Sistemul I/O conduce la executia driverelor dispozitiv. Driverele dispozitiv sunt alcatuite dintr-un set de subrutine care sunt apelate ptr a procesa diferitele etape ale unei cereri I/O. Principala procedura(subrutina) a unui driver:



1. Initializarea rutinei: se incarca driverul in sistemul de operare;
2. Adaugarea rutinei dispozitivului: este implementata printr-un driver ce sustine Plug&Play-ul; ori de cate ori un dispozitiv ce foloseste driverul este detectat P&P ii trimite driverului un semnal;
3. Rutine de executie: functiile principale pe care un driver dispozitiv le asigura(deschidere, inchidere, citire, scriere);
4. Start I/O routine: este utilizata ptr a initia un transfer de date de de la/catre un dipsozitiv;
5. Rutina de intrerupere a serviciilor: cand un dispozitiv este intrerupt, kernel-ul transfera controlul acestei rutine;

6. Rutina DPC: executa cea mai mare parte a instructiunilor implicate in manipularea intreruperii dispozitivului, dupa ce rutina de intrerupere a serviciilor a fost excutata;

Driverele pot fi clasificate astfel:

1. **Driverele modului utilizator: drivere virtuale:**

Virtualizare, masina virtuala, drivere virtuale

Virtualizarea: este o noua tehnologie softwere ce a dus la dezvoltarea domeniului IT. In esenta virtualizarea ne lasa sa transformam dispozitivul hardwere in softwere. Folosind diferite programe putem sa transformam resursele hardwere ale unui calculator (unitate centrala de prelucrare, RAM, hard disk, placa de retea) pentru a crea o masina virtuala care poate rula propriul sistem de operare si aplicatii asemenea unui calculator real.

Masina virtuala: reprezinta un calculator virtual. Ea se comporta exacat ca un calculator real. Sistemele de operare si celelalte programe nu pot face diferenta intre masina virtuala si calculator. Chiar si masina virtuala "crede" ca este un calculator. Ea este compusa in intregime din programe softwere, fara a avea o componenta hardwere. Benefiiciile masinilor virtuale sunt: compatibiliatea, izolarea, incapsularea si independenta hardwere. **Compatibilitatea:** dat fiind faptul ca masina virtuala are toate caracteristicile hardwere ale unui calculator, toate sistemele de operare si aplicatii softwere ce sunt folosite de calculatoare pot fi folosite si de masina virtuala. **Izolare:** atat timp cat masinile virtuala dispun de resuresele hardwere ale aceluiasi calculator, ele raman izolate. Daca 3 masini virtuale sunt pe acelasi server si una se defecteaza, celelalte doua functioneaza normal. **Incapsularea:** o masina virtuala este in esenta o librarie de programe softwere ce incapsuleaza un set virtual de resurse hardwere, cat si sisteme de operare si aplicatii softwere ceea ce o face portabila si usor de manevrat. **Independenta harwere:** masini virtuale aflate pe acelasi server pot rula diferite sistme de operare.

Driverele virtuale(VxD): sunt cheia unei virtualizari hardwere de success. Stiind ca programele DOS "cred" ca detin

totul in sistemul de operare, atunci cand ruleaza pe o masina virtuala, Windows-ul trebuie sa le asigure substitutiile pentru dispozitivele reale. Aceste substitutii sunt driverele virtuale. De obicei driverele virtuale virtualizeaza dispozitive hardwere. Cand

un program DOS crede ca interactioneaza cu tastatura, el de fapt interactioneaza cu tastatura virtuala.

Comunicatiile intre driverele virtuale se fac prin trei mecanisme: mesaje de control, servicii API si functii de raspuns. **Mesajele de control:** administatorul masinii virtuale trimite mesaje de control catre toate driverele virtuale incarcate in sistem cand apare un eveniment. Fiecare driver are o functie care se ocupa cu mesajele de control numita procedura de control al dispozitivului. **Servicii API:** driverele exporta deobicei functii publice ce pot fi cerute si de alte driveer virtuale. Aceste functii se numesc servicii API. Fiecare driver ce exporta astfel de servicii are un numar unic de identificare ID. Cand acest lucru se intampla driverul copiaza intr-un tabel si adresa de servicii. **Functii de raspuns:** sunt functii care exista pentru a fi cerute de alte drivere virtuale. Nu trebuie sa confundam aceste functii cu serviciile API. Aceste functii nu sunt publice ca serviciile.

2. **Driverele modului kernel[1]:**
    - Drivere de Bus;
    - Drivere de functionare;
    - Drivere de filtrare;

Drivere de filtrare: primesc unul sau mai mute tipuri de cereri I/O specifice dispozitivelor, iau cateva decizii cu privire la cerere si apoi trimit cererea catre urmatorul driver din stiva. Driverele de filtrare nu inlocuiesc dispozitivele I/O, ci ele modifica si inregistreaza o cerere pe care un alt driver o rezolva.

Drivere de functionare: este driverul care exporta interfata operationala a dispozitivului sistemului de operare. In generala, este driverul cu cele mai multe informatii despre operatiile pe care le fac dispozitivele.

Drivere de Bus: opereaza ca un driver de functionare pntru un dispozitiv parinte, care mai are dipozitive copii. Dispozitivul parinte poate fi un BUS, sau poate fi un dispozitiv multifunctional ce are copii cu functii ce necesita diferite tipuri de drivere.

Bălan Alexandru 441A

Suportul pentru piese hardwere individuale este de cele mai multe ori impartite intre mai multe drivere. Pe langa cele trei tipuri de drivere enumerate mai sus, mai apar trei componente ce asigura suportul harwere:

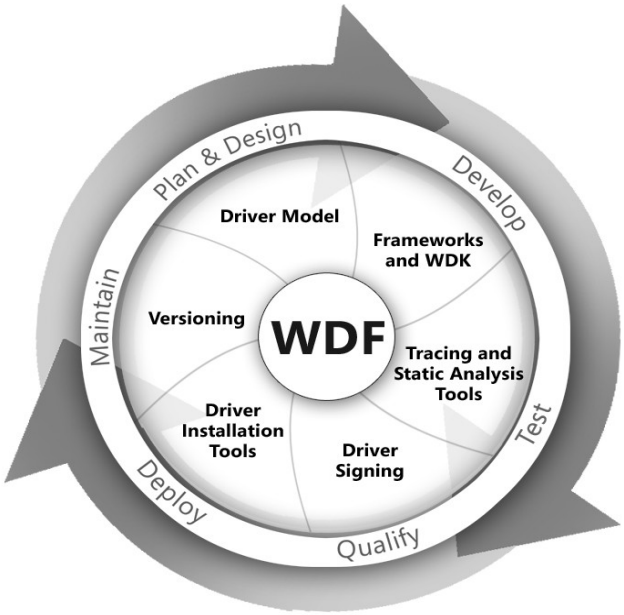
---

1 **Architecture of the Kernel-Mode Driver Framework,Microsoft**

- **Drivere de clase**: implementeaza procesul de intrare/iesire pentru anumite clase de dispozitive, unde interfata hardwere a fost standardizata astfel incat driverele sa poata sa serveasca dispozitive de la diferiti producatori;
- **Drivere de porturi:** implementeaza procesarea unei cereri de intrare/iesire specific unui tip de port I/O, si sunt implementate ca librarii ale modului kernel, sau functii decat ca drivere dispozitiv;
- **Drivere de miniporturi:** sunt drivere dispozitiv ce importa functii furnizate de un driver de porturi;

Dezvoltarea driverelor[2]:

Cei de la Windows au creat noul model de dezvoltare a driverelor, numit Windows Driver Foundation(WDF). Acesta contine componente ce suporta dezvoltarea si intretinerea driverelor modurilor kernel si utilizator.



Windows Driver Foundation si ciclul de viata al driverelor:

Bălan Alexandru
441A

Modelul driverului: modelul driverului WDF suporta crearea de obiecte. Folosind WDF programatorul se poate concentra pe dispozitiv si nu pe sistemul de operare. Acesta include:

2 **Architecture of the Windows Driver Foundation ,Microsoft**
Architecture of the User Mode Driver Framework,Microsoft

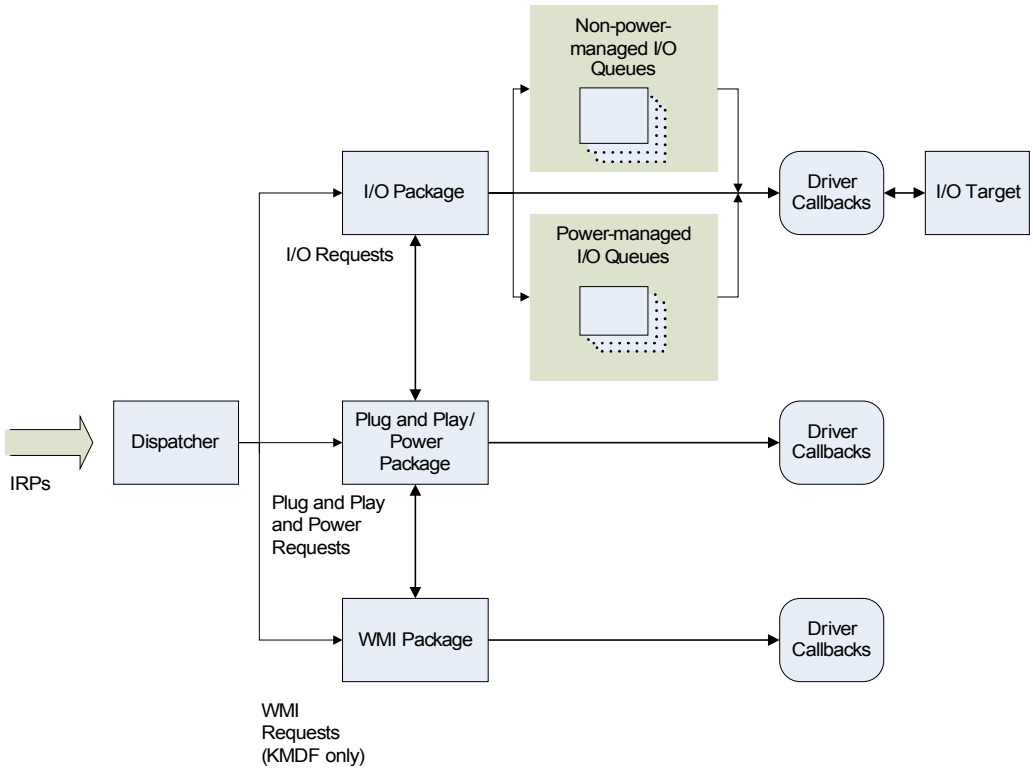
- Un obiect model ce este implementat de ambele cadre kernel si utilizator: obiectele sunt asemenea unor blocuri pentru drivere. Cadrele modifica aceste obiecte cu ajutorul unuor interfete bine definte. Obiectele au un ciclu de viata bine defint; setul de evenimente poate afecta toate obiectele. Cadrele precizeaza un comportament predefinit pentru fiecare eveniment. Suportul pentru comportamentul specific al dispozitivului include subrutine ce inlocuiesc comportamentul predefinit;

- Dispozitive Plug&Play si Power management pe care le folosesc ce le doua cadre;
- Model I/O: In windows pachetele de cereri I/O(IRP) fac mai mult decat o simpla prezentare a cererilor I/O catre drivere. Functioneaza ca un sistem de comunicatii intre sistemul de operare si drivere, si intre drivere. Administratorul I/O al windows-ului trimite IRp-uri sa instiinteze driverele de cererile Plug&Play, de cele ale managementului consumului de putere si schimbarilor statutului dispozitivului. Cadrele completeaza sau anuleaza cererile IRP in folosul driverelor. Ele cer rutinele de raspuns ale driverelor pentru a anunta o cerere pe care driver-ul trebuie sa o rezolve. Dupa ce primeste o cerere, cadrele inregistreaza informatii despre cerere, creaza un obiect ce reprezinta cererea, si cere una sau mai multe rutine raspuns pentru a o rezolva. Driverele pot crea diferite cozi in care intra obiectele, si le configureaza astfel incat sa primeasca anumite tipuri de cereri. Depinzand de mecanismul de trimitere pe care driverul il desemneaza pentru fiecare coada, cadrele fie trimit cererea imediat driverului fie o trimite mai tarziu.

Bălan Alexandru 441A

Fluxul de cereri I/O:

Ambele moduri, nucleu si utilizator, folosesc aceeasi arhitectura de I/O chiar daca sunt implementate cu componente diferite.



- Daca driverul nu a configurat coada sau nu a deschis rutina raspuns pentru tipul cererii, cadrul ia o decizie bazata pe tipul driverului.
- Daca driverul a configurat coada sau a deschis rutina-raspuns corespunzatoare tipului de cerere, cadrul creaza un obiect ce contine informatii din structura originala a IRP-ului dar si informatii despre starea driverului
- Daca driverul este setat automat pe managementul consumului de putere, cadrul determina daca dispozitivul este in starea de putere corecta. Daca nu, dispozitivul de P&P seteaza starea de functionare corecta;

- Dupa ce dispozitivul a intrat in starea de lucru, cadrul repartizeaza cererile I/O in functie de specificatiile driverelor, invocand rutine de raspuns specifice cererilor;
- Cand driverul a terminat de procesat cererea, o poate rezolva sau o poate trimite dispozitivului extern;

- Tehnici de instalare pentru ambele tipuri de drivere

Scheletul si kitul driverelui de windows: WDF defineste un singur model de driver ce include scheletul driverelor modurilor utilizator si Kernel. Scheletul acestor drivere este infrastructura de baza al modelului WDF. Acestea implementeaza caracteristicile comune si manevreaza toate operatiile cu sistemul de operare. Scheletul driverului modului kernel implementeaza caracteristicile de baza ale acestui driver ce sunt necesare Windowsului si care sunt comune tuturor driverelor modului Kernel. Scheletul driverelor modului utilizator face acelasi lucru, insa mai permite driverelor unor tipuri de dispozitive sa functioneze in modul utilizator si nu in modul kernel. Aceste structuri realizeaza urmatoarele servicii pentru driverele WDF:

- Definesc obiecte WDF pe care driverele le instantiaza;
- Administrarea obiectelor
- Expune un set de interfete ale driverelor dispozitiv pe care driverele le cheama pentru a manipula obiecte
- Asigura o implemenatre comuna a caracteristicilor driverelor
- Coorodoneaza fluxul de cereri I/O, Plug&Play de la sistemul de operare la driver

Instrumente de dezvoltare si testare: testarea unui driver este la fel de complexa ca crearea unui driver datorita a doua motive:

- Observarea erorii este dificila. In cele mai multe cazuri o eroare a driverului apare dupa mult timp de cand aceasta s-a produs. Daca un driver kernel foloseste incorect o interfata, sistemul nu pica in acel moment ci atunci cand un alt driver incerca sa faca o operatie ce se bazeaza pe aceasta eroare.
- Driverele ce pot lucra corect in situatii normale pot avea erori subtile in situatii exceptionale.

WDF au mai multe caracteristice de testare si urmarire ce usureaza munca programatorului in gasirea din timp a erorilor. Aceste caracteristici includ:

- **Dispozitiv de verificare a cadrelor:** WDF-ul are un dispozitiv de verificare cadrelor ce furnizeaza informtii despre caracteristicile specific ale cadrelor ce nu sunt disponibiledispozitivului de verificare a driverului. Acest dispozitiv asigura si urmarirea mesajelor ce aduc informatii detaliate despre lucrul cu cadrele. El urmareste referintele la fiecare obiect WDF si construieste o adresa(drum) pe care o trimite debugger-ului.
- **Dispozitiv de inregistrare a drumului**
- **Extensii ale debugger-ului:** ofera informatii despre locatia unde a aparut o eroare;

Incarcarea driverelor:

Config.sys este un fisier de configurare alcatuit din comenzi de DOS pe care acesta le cauta in directorul radacina la incarcarea sistemului de operare. Acesta este folosit pentru a incarca driverele si pentru a schimba setarile sistemului de operare.

Exemplu de cum ar arata fisierul config.sys:

DEVICE=C:\Windows\HIMEM.SYS	Aceasta linie este foarte importanta, pentru ca ea ne permite sa incarcam driverele in High Memory(primi 64K din memoria extinsa); Daca linia nu exista atunci windosul nu incarca driverele.
DOS=HIGH,UMB	Aceasta linie incarca DOS-ul in High Memory, intr-un bloc de memorie
DEVICE=C:\Windows\EMM386.EXE NOEMS	Aceasta linie incara administratorul memorie extinse
FILES=30	Permite Windows-ului sa incarce 30 de fiesiere in acelasi timp.
STACKS=0,0	Schimba stiva ori de cate ori se produce o intrerupere asincrna a dispozitivului
BUFFERS=20	Se incaraca bufere in memorie ptr a permite Windosului sa incarce memorie
DEVICEHIGH=C:\Windows\COMMAND\ANSI.SYS	Incarca driverul pentr DOS ansi.sys, ce permite sa avem in DOS caractere speciale de culori si marimi diferite.
DEVICEHIGH=C:\MTMCDAI.SYS /D:123	Incarca driverul pentru CD-ROM

- Alte drivere ce pot sa apara in liniele de cod al fisierului config.sys: display.sys,driver.sys,mouse.sys,printer.sys,ramdrive.sys,smartdrv.sys.

Bibliografie

Architecture of the Windows Driver Foundation ,Microsoft

Architecture of the Kernel-Mode Driver Framework,Microsoft

Architecture of the User Mode Driver Framework,Microsoft

Bălănica Victor 441A

4.Interfata driver – nucleu la Linux

Driver – Kernel Interface (DKI)

Linux – Kernel

Kernelul este centrul oricarui sistem de operare si cuprinde toate utilitatile necesare pentru ca acel computer sa functioneze. Este un fisier executabil care se incarca la bootare si se executa.

Linux, ca notiune stricta, se refera la kernelul creat de Linus Torvalds in 1991. Deci Linux este numele unui kernel din spatele sistemului de operare. Definirea intergului system de operare ca Linux este un abuz de limbaj; un nume corect pentru a defini sistemul de operare este GNU/Linux.

La pornirea computerului, BIOS-ul incarca si executa boot loader-ul care deschide la randul lui kernelul. Kernelul se ocupa de lansarea si oprirea din executie a tuturor proceselor. Orice proces necesita resurse din parea sistemului, cum ar fi timp de executie din partea procesorului si spatiu de executie din partea RAMului, kernelul se ocupa de alocarea de resurse pentru aceste procese si de eliberarea acestora in momentul opririi executiei procesului.

Kernelul, odata incarcat, se ocupa de urmatoarele functii :

- se ocupa de buna functionare (managementul) a dispozitivelor, a memoriei si a proceselor.
- controleaza fluxul de informatii dintre aplicatiile si hardware-ul sistemului.
- controleaza functii cum ar fi : spatiul de swap – partea a spatiului hard-disk-ului pe care kernelul il foloseste; programele demon – procese speciale ce sunt pornite dupa incarcarea sistemului de operare, asemanatoare serviciilor din Windows; sistemul de fisiere – sistemul de ierarhizare al directoarelor, al fisierelor pe disk.

DKI

Interfata driver – nucleu cuprinde toate serviciile pe care kernelul le pune la dispozitia compnentelor. Toate aceste servicii sunt apelate de drivere si sunt executate in kernel.

Interfata driver – kernel cuprinde 2 tipuri de servicii :

- servicii generale – sunt serviciile accesibile oricarui driver indiferent de procesor. Acestea cuprind : servicul de

- sincronizare, serviciul de inregistrare al dispozitivelor si al driverelor, serviciile de alocare a memoriei, serviciul de intretinerea a evenimentelor, serviciul de interuperi, serviciile de intrare/iesire s.a.m.d.
- servicii specifice unor familii de procesoare,acestea cuprinzand: servicii specifice de mamagementul intreruperilor, servicii specifice de intrare/iesire si altele.

Sincronizare

Serviciile de sincronizare se ocupa de managementul apelurilor catre acelasi serviciu din diferite fire de executie si sunt intretinute de firul de executie al DKI. Firul de executie al DKI este lansat de kernel la pornirea acestuia.

Firul de executie al DKI ruleaza ca un mecanism de functionare in urmatoarele cazuri:

- normale – toate apelurile catre porniri sau opriri de drivere se fac din interiorul firului de executie al DKI. Ceea ce echivaleaza cu faptul ca driverele nu sunt “incarcate” cu probleme de sincronizare.
- speciale – cazuri in care pornirile sau opririle de drivere nu fac parte din procesul de initializare al driverelor din kernel. In acest caz, driverele folosesc threadul DKI pentru a se ‘sincroniza’ cu alte drivere care deja ruleaza.

Alocarea memoriei

Kernelul asigura serviciul de alocare a memoriei driverelor care folosesc alocarea/eliberarea dinamica de memorie. Memoria alocata de drivere care folosesc acest serviciu este necunoscuta pentru kernel, asadar eliberarea memoriei folosite va trebui facuta chiar de driverul care a alocat acea unitate de memorie inainte de oprirea acestuia.

Sintaxa functiei de alocare/eliberare de memorie:

```
void *kmalloc(size_t nbytes, int type) ;
void  kfree(const void *ptr) ;
void *vmalloc(unsigned long size);
void  vfree(void *ptr) ;
```

Mai exista un serviciu mai special de alocare/eliberare de memorie. Este folosit de magistralele de intrare/iesire, care se folosesc de anumite

limitari ale memoriei folosite de anumite componente pentru accesarea directa a memoriei(DMA), cum ar fi dimeniune maxima sau o anumita locatie dintr-un spatiu de memorie.

Timeout

Serviciile de pauzare (timeout) pot fi folosite de diverse drivere sa verifice daca exista sau nu activitate pe un anumit dispozitiv sau sa verifice daca o anumita actiune inceputa se va termina inainte de o anumita perioada data.

Asteptare exacta

Driverele folosesc serviciul de asteptare exacta pentru a astepta pentru o scurta perioada de timp, de exemplu anumite permisiuni sau resurse necesare.

```
void udelay(long micro_secs) ;
```

Gestiunea evenimentelor

Serviciul de management al evenimentelor este asigurat de kernel tuturor driverelor de prioritate mare. Ele trebuie sa se ocupe de inregistrarea de handlere de evenimente tuturor driverelor care ruleaza. Propagarea evenimentului de la un driver de prioritate mare incepe odata cu apelul unui driverului respectic catre handlerul unui driver de prioritate mai mica.

Mascarea intreruperilor

Aceste serviciu este apelat din firul de executie al DKI si este folosit de un driver pentru a impiedica intreruperea unei anumite portiuni din executie.

Dezactivarea ‘preemptiva’ a firelor de executie

Algoritmul de programare al firelor de executie este “preemptive” – executia unui thread este intrerupta la aparitia altuia cu prioritate mai mare. Threadul din urma poate rula pana la sfarsitul executiei (poate duce la “starvation”-ramanerea in stare de asteptare pe perioade lungi de timp a celorlalte threaduri) sau numai un anumit interval de timp – time-slicing.

Acest serviciu ofera posibilitatea unui anumit driver sa activeze/dezactiveze aceasta optiune.

I/E

Acest serviciu asigura functii optimizate pentru transferul de biti.

LINUX STREAMS

Linux Streams (LiS) este o arhitectura care ofera o alta interfata intre kernel si anumite drivere numite Streams Drivers.

Streams defineste un model de comunicare intre servicii pentru sistemele Linux/Unix. Streams defineste o modalitate de schimb informational in interiorul kernelului si intre kernel si restul sistemului.

Idea de baza pentru Streams este stream-ul – adica crearea unui flux de date. Streams da posibilitatea de construire a unor module si de a face apoi conexiunea intre diferite servicii cu ajutorul fluxurilor de date (streams); informatia fiind impartita in module, care sunt mult mai usor selectate si interconectate.

Un flux de date (stream) se compune din 3 parti: un varf al streamului, unul sau mai multe module si un driver. Functionarea acestuia poate fi prezentata astfel : o anumita aplicatie deschide un dispozitiv Streams, actiune in urma careia se formeaza un varf al streamului pentru a putea accessa driverul respectiv. Varful fluxului primeste datele de la aplicatia deschisa si o transmite sub forma unui mesaj stream kernelului. Apoi modulele specificate sunt introduse in stream pentru a executa sarcinile specificate de aplicatia respectiva.

Cateva dintre driverele Streams sunt urmatoarele :

- clone-drvr – acest driver ajuta LiS la implementarea functiei de deschidere clona (‘clone open’ ajuta la deschidrea unui dispozitiv dintr-un fond comun de drivere). Se afla in /dev/clone_drvr
- link-drvr – creaza o legatura intre driverele Stream si driverele de retea. Se afla in /dev/ldl
- loop-around – realizeaza o conexiune intre 2 fluxuri de date astfel incat datele inscrise intr-un flux pot fi accesate din celalalt flux. Se afla in /dev/loop.1, /dev/loop.2.
- mini-mux – driverul este folosit de LiS pentru a isi autotesta procedurile. Se afla in /dev/minimux.1, /dev/minimux.2.

- printk – driverul primeste mesaje scrise si le printeaza cu ajutorul functiei printk din kernel. Se afla in /dev/printk.
- sad – driverul realizeaza o gestiune a functiilor LiS. Se afla in /dev/sad.

Alocarea memoriei prin LiS

Sunt mai multe metode de alocare a memoriei prin LiS toate separate de kernel. Avantajul acestor metode nu provine din faptul ca alocarea se facea prin mia multe metode ci din faptul ca toate aceste metode sunt compatibile cu noile variante de kernel fara a avea nevoie de recompilare. Pentru a beneficia de metodele alocarii memoriei prin LiS terbuie inclus `<sys/lismem.h>` in codul sursa al driverului Stream.

Primul grup de metode de alocare de memorie sunt secventele ce realizeaza operatii asemanatoare cu “malloc” si “free”. Aceste secvente de cod formeaza o lista inlantuita ce cuprinde toate elementele de memorie allocate, astfel incat fiecare locatie alocata are incastrata un nume de fisier si un numar de linie din codul care a generat alocarea.

Sintaxa functiilor de alocare/eliberare de memorie este :

```
void *ALLOC(int nrbytes) ;
void *ALLOCF(int nrbytes, char *tag) ;
void  FREE(void *ptr) ;
```

O alta varianta de alocare a memoriei este folosind alocatorii de memorie din kernel. Aceasta varianta ofera o mai mare flexibilitate in alegrea optiunilor de alocare de memorie din interiorul kernelului. Totusi toate exemplele ajung intr-un apel al functiei “kmalloc” cu argumentele respective. (aici GFP_ATOMIC, GFP_KERNEL, GFP_DMA)

```
void    *lis_alloc_atomic(int nrbytes) ;
void    *lis_alloc_kernel(int nrbytes) ;
void    *lis_alloc_dma(int nrbytes) ;
void    *lis_free_mem(void *mem_area) ;
```

Crearea de fire de executie

Un driver Stream poate crea si fire de executie, compactand tot codul necesar crearii unui thread intr-o singura expresie.

Sintaxa:

```
pid_t lis_thread_start(int (*fcn)(void *),void *arg,
                                        const char *name);
int   lis_thread_stop(pid_t pid);
```

lis_thread_start este functia care initializeaza noul fie de executie dandu-i anumite argumente: fcn va functiona ca si main thread (fir de executie principal) si name va fi numele aplicat noului thread. Va returna numarul de identificare al noului thread in caz de reusita sau un numar negative de eroare in caz opus. Acest numar de identificare va ajuta la oprirea firului de executie respective, fiind singurul argument al lis_thread_stop.

Accesarea porturilor de I/E

Functiile care permit driverului sa acceseze porturile de intrare/iesire sunt:

```
int  check_region(unsigned int from, unsigned int extent)
;
void request_region(unsigned int from,
                    unsigned int extent,
                    const char  *name) ;
void release_region(unsigned int from,unsigned int
extent);
```

Secventele de printare

Cele mai folosite functii de printare ale kernelului sunt:

```
int printk(const char *fmt, ...) ;
int sprintf(char *bfr, const char *fmt, ...) ;
int vsprintf(char *bfr, const char *fmt, va_list args) ;
```

Functii unitare prin LiS

```
void  lis_atomic_set(lis_atomic_t *atomic_addr,int val) ;
int   lis_atomic_read(lis_atomic_t *atomic_addr) ;
void  lis_atomic_add(lis_atomic_t *atomic_addr,int amt) ;
void  lis_atomic_sub(lis_atomic_t *atomic_addr,int amt) ;
void  lis_atomic_inc(lis_atomic_t *atomic_addr) ;
void  lis_atomic_dec(lis_atomic_t *atomic_addr) ;
int   lis_atomic_dec_and_test(lis_atomic_t *atomic_addr);
```

Bibliografie :

- Linux for Dummies 8th edition 2008
- Linux from scratch, versiune 2.2 2002
- http://ro.wikipedia.org/linux
- Linux bible 2007 edition

Bogdan Andra 441A

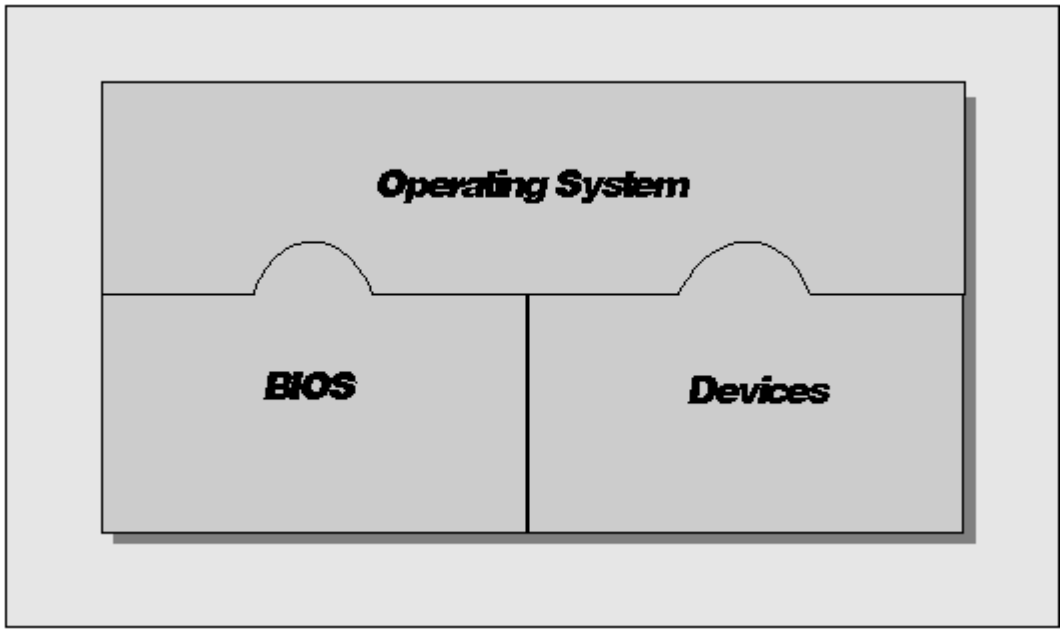
5.Gestionarul Plug and Play la Windows:

Plug and Play(PnP) a fost conceput pentru a permite adaugarea de noi dispozitive periferice fara sa mai fie nevoie de interventia utilizatorului in problemele de configurare si de instalare.
Un sistem Plug And Play are o serie de caracteristici printre care se numara :instalarea se realizeaza automat, sistemul determina configuratia optima si aplicatiile se ajusteaza automat noilor modificari, utilizatorii nu trebuie sa modifice configuratia fisierelor sistemului de operare .

Procesul Plug and Play este accesat la momentul de boot: cand porneste calculatorul, dispozitivele PnP sunt identificate si se stabilesc adrese neocupate si numere IRQ.
Tehnologia Plug and Play a fost folosita prima data de Microsoft la Windows 95.

Un calculator compatibil cu PnP trebuie sa indeplineasca trei cerinte:

1)un sistemul de operare compatibil cu PnP
2)BIOS trebuie sa suporte PnP
3) dispozitivul care urmeaza sa fie instalat trebuie sa fie un dispozitiv de tip Pnp



[1][2] Functiile de baza pe care cele trei componente Plug and Play trebuie sa le coordoneze si sa le realizeze includ urmatoarele:

- **Identificarea dispozitivelor instalate**
- **Determinarea resurselor de care dispozitivul are nevoie**
- **Crearea unei configuratii complete a sistemului, eliminand conflictele dintre resurse**
- **Incarcarea dispozitivelor pentru drivere**
- **Notarea schimbarilor de configuratie**

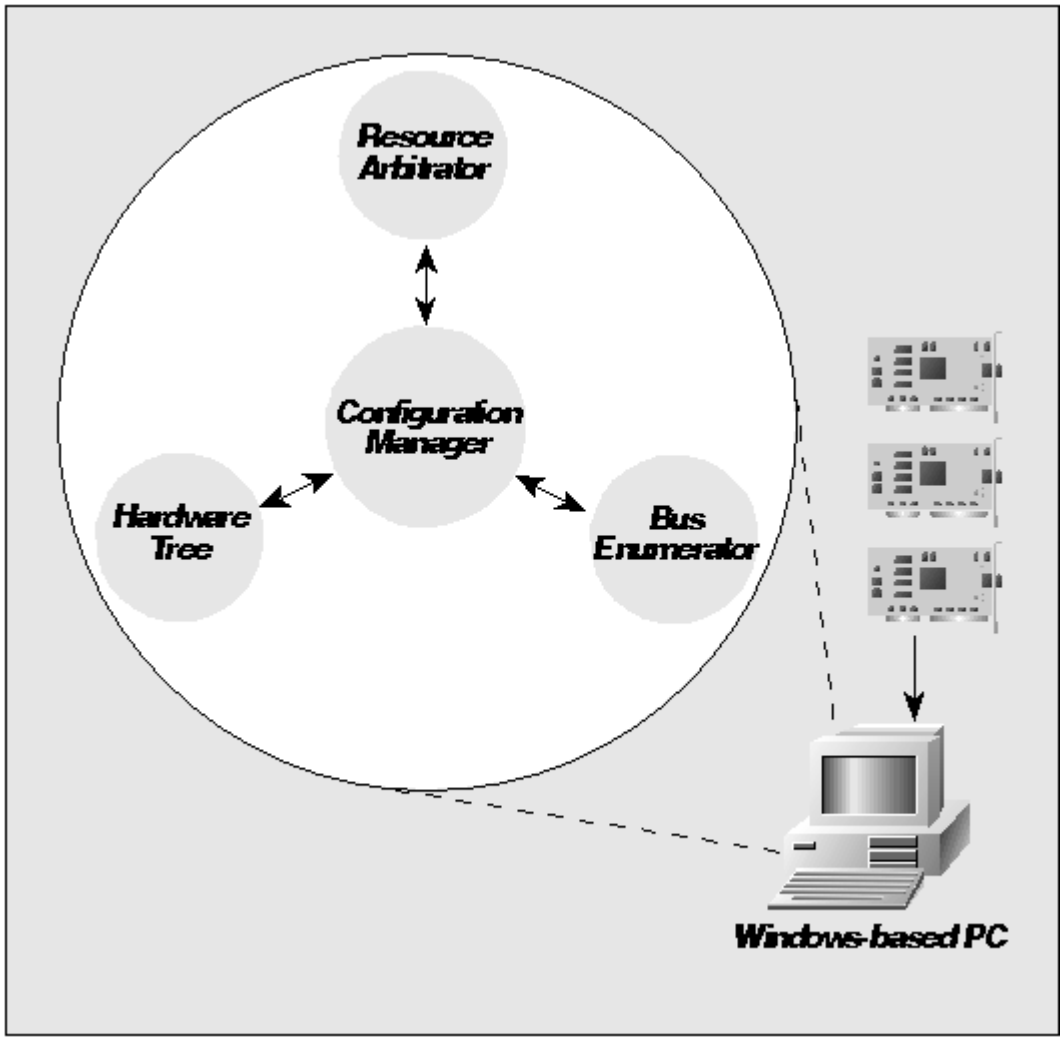
Pe masura ce dispozitivele sunt adaugate sau schimbate , aceste procese se repeta. Sistemul de operare se ocupa in mod deosebit de configurarea acestor procese pentru asigurarea unei bune coordonari. Insa o parte din aceasta configuratie este realizeaza de catre BIOS-ul sistemului la pornire. Pentru ca sistemul sa realizeze operatia de 'boot', BIOS executa initial o configuratie de baza a dispozitivului display, dispozitivului input si dispozitivului de incarcare al programului initial. Apoi, trece informatiile cu privire la fiecare din aceste dispozitive catre sistemul de operare pentru o configurare aditionala a sistemului.

Sistemul de operare continua procesul de configurare identificand fiecare dispozitiv din sistem si cerintele de resurse de care au nevoie. Fiecare dispozitiv non-boot trebuie sa fie inactiv la pornire pentru ca sistemul de operare sa poata identifica orice conflict care ar putea aparea intre resursele de care au nevoie diferitele dispozitive inainte de a le configura. Daca se intampla ca diferite dispozitive sa aibe nevoie de aceleasi resurse, acestea trebuie sa poata oferi informatii catre sistemul de operare despre resursele alternative pe care le pot folosi. O data ce conflictele legate de resurse se rezolva, sistemul de operare programeaza automat fiecare dispozitiv hardware cu configuratia sa de lucru si apoi stocheaza toate informatiile de configurare in baza de date centrala. In final, sistemul de operare incarca driverele catre fiecare dispozitiv .

Daca are loc o schimbare in configuratia sistemului in timpul acestei operatii, sistemul hardware trebuie sa poata anunta sistemul de operare cu privire la acest lucru pentru ca sistemul de operare sa configureze noul dispozitiv.
In plus , aplicatiile trebuie sa poata sa raspunda la schimbarile de configuratie pentru a folosi noul dispozitiv si sa inceteze apelul catre dispozitivul care a fost schimbat.

[1][5] Pentru asigurarea functionarii Plug and Play sitsemul de operare Windows are urmatoarele componente:

- **Managerul de configurare**
- **Arborele hardware (Hardware Tree)**
- **Numaratorii de bus (Bus Enumerators)**
- **Arbitrii resurselor (Resource Arbitrators)**



Managerul de configurare este componenta software principala care se ocupa de toate fazele de configurare ale proceselor. Administreaza ordinea de realizare a operatiilor, accepta si raspunde comunicatiilor dintre BIOS si dispozitivele harware la configurare. De asemenea, raspunde proceselor dinamice din timpul operatiilor de inserare sau eliminare a dispozitivelor. In timpul acestor procese Managerul de configurare comunica informatiile aplicatiilor.

Arborele hardware- realizeaza o inregistrare a configuratiei curente a sistemului. Informatiile arborelui este preluata din baza de date centrala a informatiilor configurate pentru toate dispozitivele numita Registru(Registry).Acesta este stocat local pentru fiecare calculator in parte si contine informatiile necesare despre toate tipurile de dispozitive care pot fi suportate de sistem fie ca acestea sunt deja instalate sau nu. Arborele hardware este creat de Managerul de configurare la fiecare pornire a sistemului.

Numaratorii de bus-sunt responsabili de realizarea arborelui hardware la un sistem Plug and Play. Fiecare numerator are o structura diferita de bus si suporta detaliile de implementare pentru tipul sau.Astfel, un numerator ISA poate identifica dispozitivele de pe un bus ISA, le citeste resursele necesare si le configureaza conform instructiunilor date de Managerul de configurare. La instalare , Windows determina automat ce numerator bus se potriveste respectivului calculator.

Arbitrul de resurse aloca fiecarui dispozitiv resursele care i se potrivesc si rezolva conflictele dintre dispozitivele care cer aceleasi resurse.

[1] **Dispozitive Plug and Play**

Cele mai importante dispozitive Plug and Play sunt: ISA, PCMCIA, PCIA,SCSI, IDE CD-ROM, MicroChannel. Fiecare dispozitiv hardware este identificat diferit fata de alte dispozitive cu acelasi nivel de prioritate in functie de resursele pe care le solicita, driverul pe care il necesita,servicile pe care le ofera, configuratia software.
Principalele resurse folosite sunt : intreruperile, porturile I/O, canalele DMA, sirule de memorie.

In general se folosesc cateva protocoale hardware dinamice denumite:rece (cold), cald(warm) si fierbinte (hot). Cold inseamna ca dispozitivul poate fi doar introdus sau scos(“plugged” sau “unpluged”) in timp ce calculatorul este oprit. Warm se refera la faptul ca sistemul poate ramane pornit dar majoritatea aplicatiilor software(inclusive sistemul de operare si toate aplicatiile) trebuie oprite. Cazul hot este foarte important deoarece se foloseste cand dispozitivele hardware pot fi adaugate sau scoase in timp ce sistemul este pornit , functioneaza si se asteapta ca toate celelalte dispozitive hardware si aplicatii software sa supporte noile schimbari aduse sistemului.
Majoritatea aplicatiilor sunt interesate de protocolul asa-numit “hot.

[4] **Elemente de implementare software la Plug and Play**

Comunicare dintre sistemul de operare Windows si Plug and Play se realizeaza prin Win 32 API . Comunicarea se face prin mesaje, (precum WM_DEVICECHANGE), valabile ca ID prin WM_COMMAND. Mesajul de schimbare device (WM_DEVICECHANGE) foloseste coduri care ofera informatii legate de schimbarile pe care le provoaca in sistem un anume dispozitiv hardware.
In fereastra principala(main) se foloseste procedura switch(..) pentru dispozitivul Plug and Play si parametrii sai.

Codurile DBT_* ofera informatii cu privire la tipul de hardware continut in valorile DBT_DEVTYP_* .Aceste valori sunt:

DBT_DEVTYP_DEVMODE	Numar devmode(specific la Windows 95)
DBT_DEVTYP_VOLUME	Driver logic
DBT_DEVTYP_PORT	Port serial sau paralel
DBT_DEVTYP_NET	Resurse network(UNC)

Dintre toate acestea DBT_DEVTYPE_VOLUME este cel mai util .
Activitatile principale se identifica cu mesajul WM_DEVICECHANG si cu codurile DBT_* atunci cand un hardware nou se adauga.
Principalele mesaje DBT_* folosesc pointeri catre structuri. Aceste structuri se numesc _DEV_BROADCAST_* si contin: dimensiunea dispozitivului, un nume sau un identificator si un flag.
Mesajul cel mai importante Plug and Play este WM_DEVICECHANGE (sau WM_DISPLAYCHANGE).Acest mesaj identifica schimbarile in display mode.

Evolutia plug and play in sistemele Windows

[5] **Windows 95** a fost primul system Microsoft care a folosit Plug and play.Printre imbunatatirle esentiale produse de aceasta tehnologie care s-au pastrat si in viitoarele sisteme de operare Windows se numara:
·La instalarea unui nou hardware in sistem , protocoalele Plug and Play lucreaza cu noul dispozitiv , cu BIOS si cu SO pentru a identifica si aloca resursele necesare, pentru a activa noul dispozitiv si pentru a incarca driverul util.Daca driverul nu se afla in system, SO va cere utilizatorului un disc pentru driver.

·O aplicatie importanta la windows 95 pentru Plug and play este “AutoPlay”. Cand un disc este introdus ,SO il recunoaste si aplica un fisier batch. AutoPlay porneste aplicatia automat.

[5]Spre deosebire de Windows 95, la **Windows 2000** implementarea PnP nu se bazeaza pe APM(Advanced Power Management) BIOS sau pe Pug and Play BIOS. LA Windows 2000 se pastreaza numai datorita compatibilitatii.In acest system de operare PnP se optimizeaza pentru laptopuri, servere, statii de lucru

care include plac de sistem ACPI. Driverul support pentru clasele de dispozitive este Win 32 Driver Model.

Pnetru incorporarea Pnp la Windowds 2000 s-a integrat o noua implementare in codul de baza existent deja in windows.Astfel rezulta urmatoarele modificari:

*Driverele pentru bus-uri sunt separate prin HAR(hardware abstratioon layer).Acestea controleaza bus-ul I/o, incluzand si functionalitatea slot-urilor care sunt dispozitive indepenedente.

*Noile PnP API se folosesc pentru a citi si scrie informatii din registru.Pentru acestea se fac schimbari la nivelul structurii de registru.

Plug and play a continuat sa evolueze ajungandu-se ca Windows Millenium Edition(Me) si Windows XP sa foloseasca Universal Plug and play(UPnP) pentru folosirea dispozitivelor de tip networtk. Functionarea UPnP implica cinci procese importante:

Descoperirea: Un dispozitiv UPnP isi face cunoscuta prezenta in retea catre alte dispozitive si puncte de control folosind protocolul SSDP()Simple Service Discovery Protocol)

Descrierea:Folosind adresa URL in procesul de descoperire un punct de control, primeste informatii XML care descriu caror dispozitive se adresea arhitectura UPnP.

Schimbarile care au loc la diferitele dispozitive din retea sunt anuntate prin mesaje la punctual de control.Aceste mesaje sunt formate in XML si folosesc athitectura numta GENA(General Event Notificaytion).

Prezentarea: Daca un dispozitiv UPnP ofera o prezentare URL se folseste browserul pentru accesarea intefatelor de control,dispozitiv sau service sau orice alt dispozitiv specific implementat de producator.

UPnP foloseste la Windows XP urmatoarele fisiere si service:

•UPnP
Upnpcont.exe
Upnphost.dll – pentru gazduirea dispozitivelor UPnP
Upnp.dll – libraria principala UPnP()DLL
Upnpui.dll –folosit de XP pentru crearea interfetei Windows XP
•SSDP
Ssdpaip.dll - Application Programming Interface (API) DLL pentru SSDP
Ssdpsrv.dll

Bibliografie:

[1]Davis S. Lawyer, Plug-and-Play-HOWTO, 2002, cap 2: What Pnp Should Do: Allocate “Bus-Resources”, cap 3: The Plug-and-Play (PnP) Solution, cap 4: “Setting up a Pnp BIOS”

[2] Andrew S. Tanenbaum , Modern Operating Systems, 2nd Edition, , Editura Prentice Hall, 2002 , Capitolul 5: Input/Output

[3]. Microsoft Windows and the Plug and Play Framework Architecture

[4] Win32 Application Support for Plug and Play

[5]. Plug and Play for Microsoft Windows 95

6.Module incarcabile la Linux

Modulele incarcabile (sau LKM – loadable kernel modules) sunt fisiere care prin incarcarea lor in memorie extind kernelul unui sistem de operare.

Un modul incarcabil poate fi descris ca o biblioteca de functii. Acesta este un fisier ce contine cod pentru extinderea functiilor sistemului de operare.
Marele avantaj oferit de aceste module este acela ca se pot incarca in memorie doar cand este nevoie de functiile continute de acestea si pot fi la fel de usor descarcate, eliberand astfel memoria.

Aceste module sunt in general folosite pentru driverele sau fisierele sistem noi adaugate.[1]

Modulele incarcabile au fost introduse pentru prima data in anul 1995, dar s-au dezvoltat cu adevarat pana in anul 2000. In acest moment toata sistemele de operare folosesc aceste module, doar ca sub diferite denumiri.

Pana la introducerea modulelor incarcabile kernelul Linux continea intreaga biblioteca de drivere. Modulele nu sunt parti(programe) exterioare kernelului, ci odata incarcate ele devin parte integranta a acestuia[2].

Un alt avantaj al modulelor vine prin partea de instalare/ depanare. Din punct de vedere al instalarii, nu mai trebuie recompilat intreg kernelul pentru adaugarea unei biblioteci(nou create), ci trebuie doar incarcat modulul cu biblioteca in memorie. Din punct de vedere al depanarii este evident faptul ca daca biblioteca (defecta) s-ar gasi direct in kernel, ar fi greu de depistat, de reparat si recompilat. Asa odata kernelul incarcat se poate cauta biblioteca defecta si se poate repara separat.

In principal modulele incarcabile linux sunt folosite pentru urmatoarele:
-Driverele pentru sistem. Pentru a functiona un floppy, etc., foloseste drivere. Acestea sunt folosite sub forma de module, nemaifiind nevoie de stocarea tuturor driverelor in kernel. Acestea se incarca odata cu detectarea lor.

Măracineanu Alexandru-Ciprian 441A

1) http://en.wikipedia.org/wiki/Loadable_Kernel_Module
2) http://tldp.org/HOWTO/Module-HOWTO/x73.html

-Sistemul de fisiere. In functie de cum sunt stocate fisierele avem nevoie de un drive care sa interpreteze modul de folosire al acestora. (ext2)

-Pentru apelurile de sistem. Se pot crea module care sa inlocuiasca/foloseasca comenzile prezente in kernel pentru a crea functii noi, de genul alarma la o anumita ora, sau stingere programata, etc.

-Pentru protocoalele de comunicatie.

Modulele incarcabile se pot gasi cam in orice director din al sistemului, dar in general ele se gasesc in : /lib/modules, organizate pe subdirectoare aici.
Din punct de vedere al adaptibilitatii s-ar putea ca unele module sa nu fie compatibile cu toate kernelurile existente. In acest sens acestea contin si versiunea pentru care sunt facute si la incercarea de incarcare al acestuia se verifica versiunea si compatibilitatea acestuia.

In timpul cautarilor am descoperit faptul ca pentru a incarca/descarca/sterge modulele in/din memorie exista o serie de utilare cu aceste functii. Acestea serie de utilare se numeste LKM Utilities. Ea are mai multe functii printe care :
-insmod -> insereaza un modul in kernel
-rmmod -> scoate un modul din kernel
-lsmod -> listeaza modulele incarcate in memorie
-modinfo -> afiseaza versiunea pentru care a fost creat.[13]

Instalarea utilitarelor pentru module se poate face cu urmatoarea comanda: (daca fisierul este un RPM)

bash# rpm -i /mnt/cdrom/Redhat/RPMS/modutils*.rpm

Cum am spus si mai sus comanda de inserare pentru un modul nou este : insmod. In mod obisnuit modulele au extensia .o.
Exemplele de mai jos cu extensia .o au fost folosite pana la versiunea 2.6
Cea mai vizibila schimbare dupa aceasta versiunea este cea a extensie modulelor, care s-a transformat din .o in .ko .
Un exemplu de folosire al acestei comenzi este :

insmod drv_floppy.o

31) http://tldp.org/HOWTO/Module-HOWTO/x146.html

drv_floppy.o este denumirea fisierului ce contine modulul care va fi incarcat in memorie. Daca se foloseste comanda lsmod atunci vor fi afisate toate modulele incarcate in memorie. Un astfel de rezultat obtinut in urma utilizarii comenzii este :

Used by: drv_floppy 3

Unde “drv_floppy”este numele modului (nu al fisierului), iar 3 este dimensiunea pe care acesta o ocupa in memorie (in pagini). O pagina = 4k memorie.

Comanda rmmod se foloseste in sintaxa :

rmmod drv_floppy

Paramentru acestei comenzi “drv_floppy” trebuie neaparat sa specifice numele modului ce va fi scos din memorie, nu numele fisierului.

De foarte multe ori comanda, insmod nu functioneaza deoarece versiunea modului, respectiv kernelului nu se potrivesc.
Pentru a verifica daca fost executata cu succes comanda, chiar daca nu a fost intors nici un mesaj de eroare, atunci puteti sa cautati modulul in directorul :`/proc/modules`

In general modulele incarcabile nu au parametrii standard, fiecare creator al unui astfel de modul alegand ce parametrii sa se incarce din kernel o data cu executia modulului.

Dupa incarcarea cu succes a unui modul in memorie, primul lucru pe care il face noul modul la bootare este acela de a verifica daca exista vreun driver pentru care sa fie activ, adica vreo componenenta a calculatorului care sa poata fi utilizata de acel driver si invers. Daca aceasta nu este prezent atunci incarcarea va esua.

Daca incarcarea s-a facut cu success si driverul a fost gasit, atunci componenta hardware pe care acesta o controleaza poate fi accesata prin intermediul functiilor acestuia. Adica se pot folosi functiile de intrerupere alea hardwarerului, setupul acestuia, etc.

Un driver (modul) creat corect comunica cu kernelul spunandu-i acestuia pentru ce parti hardware este folosit si daca este pregatit. Totodata acesta afiseaza mesajele si in consola si in fisiere log *:* `/var/log/messages.`

Totodata un driver bine facut afiseaza si mesajele de eroare cu privire la motivele pentru care nu a putut fi incarcat in memorie.[3]

Măracineanu Alexandru-Ciprian 441A

3 http://tldp.org/HOWTO/Module-HOWTO/x197.html

Sistemul contine si functii avansate care vin sa usureze si sa verifice mult mai usor incarcare modulelor.

O astfel de comanda este : *modprobe drv_floppy*
Aceasta verifica toate driverele (modulele incarcabile) de care este nevoie pentru a incarca, respectiv rula acest driver. Si daca drv_floppy va avea nevoie de alte drivere pentru a rula va fi rulata comanda insmod si va incarca toate aceste drivere.
Pentru verificarea si scanarea modulelor de care modprobe are nevoie, aceasta va apela automat comanda depmod.

Modulele pot fi facute si in asa fel incat sa fie incarcate la cererea kernelului. Acest lucru se face prin "loader-ul" kernelului pe versiunile mai noi, fie prin intermediul daemonului pe versiunile mai vechi.

Pentru a vedea modulele prezente se foloseste comanda:
cat /proc/modules
Modulele prezente vor fi afisate sub forma:

```
lp              5280   0 (unused)
parport_pc      7552   1
parport         7600   1 [lp parport_pc]
```

In parantezele drepte se gasesc modulele de care care depinde driverul incarcat.Spre exemplu modulul parport este dependent de modulul parport_pc.

Modulele incarcabile pot fi scrise in limbajul de programare C. Site-urile trecute ca sursa contin paginile manualelor de linux. Sau sunt capitole ale unor carti.

Bibliografie :
http://www.faqs.org/docs/kernel/x45.html
http://en.wikipedia.org/wiki/Loadable_Kernel_Module
http://tldp.org/HOWTO/Module-HOWTO/
http://www.linux.sgi.com/LDP/HOWTO/Kernel-HOWTO/loadable_modules.html

7.Concluzie

Colecţia de drivere dispozitiv la Linux *Zamfir Narcis*

Arhitectura Linux se bazează foarte mult pe manipularea fişierelor. De aceea şi dispozitivile I/E vor fi abstractizate tot ca nişte fişiere în spaţiul de lucru(care este denumit sistemul virtual de fişiere). Dispozitivele care pot fi detectate folosind driverele dispozitiv care vin cu Linux-ul se regăsesc ca şi fişiere în directorul *root/dev/*. Dacă se doreşte scrierea într-un astfel de dispozitiv se pot folosi comenzi simple ca *write()* sau *read()* dar aplicaţiile avansate pot folosi dispozitivele şi prin comenzi complexe prin intermediul driverelor corespunzătoare. Linux oferă multe unelte de modificare şi control al driverelor. În versiunile mai noi de Linux driverele sunt păstrate în directorul *root/sys/* şi organizate după caracteristici şi ierarhii. De asemenea se oferă posibilitatea de introduce drivere noi în rularea kernel-ului prin *module încarcabile* precum şi configurarea celor existente.

Spre deosebire de windows se pune accentul pe posibilitatea configurării de drivere prin modificarea lor în medii de dezvoltare ce cer cunoştinţe medii de programare(de obicei C++ sau VirtualC). În fişierele de bază ale Linux-ului se găsesc un set de dll-uri (=librarii software) care conţin un set de drivere dispozitiv care sunt încarcate şi rulate în kernel dacă se detectează un dispozitiv compatibil. Dacă mai multe dispozitive de acelasi fel foloseasc acelaşi tip de driver atunci dll-ul respectiv este încărcat încă o dată în memorie şi va rula ca un proces independent. De obicei procesele de drivere în linux au privilegii limitate şi nu pot manipula întreruperi sau deschide porturi noi de I/E. Sistemul de operare procesează întreruperile şi trimite procesului driver un mesaj de notificare a apariţiei întreruperii. Linux a avansat de-a lungul versiunilor şi se poate spune astăzi ca poate concura cu Windows-ul din punct de vedere al mediului de dezvoltare de drivere dispozitiv. Componentele necesare creării unui driver de Linux au o structură standardizată şi din ce în ce mai multe dispozitive vin cu drivere de Linux.

Colectia de drivere dispozitiv la Windows *Alexandru Bălan*

Driverele sunt componente pe care sistemul de operare le foloseste sa interactioneze cu dispozitivele exterioare. Un driver este alcatuit din subrutine ce sunt apelate pentru rezolvarea unuor cereri de intrare/iesire. Driverele sunt de doua feluri: drivere ale modului utilizator si drivere ale modului nucleu. Driverele virtuale apartin modului utilizator si rolul lor este acela de a substitui dispositive hardwere(mouse, tastatura). Driverele modului kernel(nucleu) sunt: driverele de BUS, de functionare de filtrare.

Interfata driver – nucleu la Linux *Bălănica Victor*

Cea mai mare parte a interfetei dintre diverse drivere si module din nucleu si serviciile nucleului este compusa din functii declarate in headerele fisierele aflate in kernel. Aceasta practica a fost considerate mai eficienta din considerente de viteza a executiei. Totusi aceasta tehnica are un dezavantaj mare in necesitatea de recompilare a driverelor dupa fiecare instalare a unui nou kernel.

Cea de a doua varianta de conectare a diverselor drivere cu nucleul pe care am prezentat-o nu prezinta aceasta necesitate. Linux Streams ofera posibilitatea instalarii doar a unor pachete continand drivere care respecta modelul Streams. LiS este de cele mai multe ori singura secventa de cod care trebuie recompilata. Totusi LiS este o varianta de interfata in dezvoltare scopul fiind dorinta de a se ajunge ca driverele Stream sa fie independente de versiunea kernelului.

Geostinarul Plug and play la Windows *Bogdan Andra*

Plug and play a adus imbunatatiri majore in industria PC. Acesta tehnologie este flexibila ,robusta si se poate implementa la toate nivelurile. Sistemele cu BIOS compatibil cu Plug and Play recunosc cu usurinta elementele de hardware din sistem si impreuna cu sistemul de operare creeaza conditii optime de lucru cu aceaste dispozitive.

Printre tehnologiile Plug and Play se numara:ISA,PCMCIA PCI, SCSI, IDE CD-ROM, MicroChannel.Principalele resurse folosite sunt: porturile I/O, canalele DMA, sirurile de memorie.

Unele clase de aplicatii pot determina cand resurse hardware sunt adaugate sau indepartate din sistem, cand intervin fluctuatii de current prin si transmit mesaje catre utilizator prin: WM_DEVICEBROADCAST, WM_POWERBROADCAST, DBT_* si PBT_*.

Module incarcarbile pentru Linux *Mărăcineanu Ciprian*

In concluzie modulele incarcabile sunt fisiere care prin incarcarea lor in memorie extind kernelul unui sistem de operare.
Un modul incarcabil poate fi descris ca o biblioteca de functii. Acesta este un fisier ce contine cod pentru extinderea functiilor sistemului de operare.

Marele avantaj oferit de aceste module este acela ca se pot incarca in memorie doar cand este nevoie de functiile continute de acestea si pot fi la fel de usor descarcate, eliberand astfel memoria.

Principalele domenii de activatate ale modulelor incarcabile linux:
-Driverele pentru sistem. Pentru a functiona un floppy, etc., foloseste drivere. Acestea sunt folosite sub forma de module, nemaifiind nevoie de stocarea tuturor driverelor in kernel. Acestea se incarca odata cu detectarea lor.
-Sistemul de fisiere. In functie de cum sunt stocate fisierele avem nevoie de un drive care sa interpreteze modul de folosire al acestora. (ext2)
-Pentru apelurile de sistem. Se pot crea module care sa inlocuiasca/foloseasca comenzile prezente in kernel pentru a crea functii noi, de genul alarma la o anumita ora, sau stingere programata, etc.
-Pentru protocoalele de comunicatie.