Device drivere în Linux

Device driverele in UNIX se impart in doua categorii, ierarhie facuta dupa viteza, volumul şi modul de organizare a datelor ce trebuie transferate de la dispozitiv către sistem şi invers:

- de tip caracter: dispozitve lente, care gestioneaza un volum mic de date, iar accesul la date nu necesita operatii de cautare prea frecvente.
  Ex: tastatura, mouse-ul, placa de sunet, joystick-ul.
- de tip bloc: datele sunt organizate pe blocuri, iar volumul fiind mare operatiile de cautare sunt des folosite.
  Ex: hard disk-urile, cdrom-urile, ram discurile, unităţile de bandă magnetică.

Cele două tipuri de device drivere se apeleaza in mod diferit. Dacă pentru dispozitivele de tip caracter apelurile de sistem ajung direct la device drivere, în cazul dispozitivelor de tip bloc device driverele nu lucrează direct cu apelurile de sistem. Diferenta e data de interpunerea subsistemului de gestiune a fisierelor. Rolul acestuia este de a pregăti device driverului resursele necesare (buffere), de a menţine în buffer cache datele recent citite şi de a reordona operaţiile de citire şi scriere din motive de performanţă.

Identificarea in UNIX a dispozitivelor se face cu ajutorul identificatorului. Acestia se aloca atat static cat si dinamic, si este alcatuit din doua parti: **major** si **minor**. Astfel se identifica mai intai(prin major) tipul driverului prezent iar, prin minor, fiecare tip de dispozitiv deservit de acesta.

Device drivere de tip caracter

Device driverele de tip caracter primesc nealterate apelurile de sistem efectuate de utilizatori asupra fişierelor speciale. Deci, pentru a implementa un device driver, vor trebui implementate apelurile de sistem de lucru cu fişiere: open, close, read, write, lseek, mmap, etc.

Structurile inode şi file

Un *inode* identifica in mod unic unic un fisier intr-un sistem de fisiere. Acesta reprezinta un fisier avand ca atribute dimensiunea, drepturile si timpii asociati. Structura *file* se apropie mai mult de punctul de vedere al utiliztorului in ceea ce priveste un fisier, lucru regasit si din atribute: inode-ul, numele, posibilitati de deschidere.

Ca drivere cele doua structuri au intotdeauna modalitati standard de folosire: inode-ul se foloseste pentru a identifica majorul si minorul, iar filepul se foloseste pentru a identifica flag-urile cu care a fost deschis fisierul dar si pentru a memora si accesa mai tarziu date private.

Atunci când se creează un dispozitiv de tip caracter se recomandă crearea unei structuri care să conţină informaţii despre dispozitivul dat, informaţii utilizate în cadrul modulului, si anume „struct cdev” .

Accesul la spaţiul de adresă al procesului

Un dispozitiv este interfaţa de comunicaţie între o aplicaţie şi hardware, astfel vor trebui accesate date din user-space. Accesarea insa nu se poate face direct, folosindu-se pentru aceasta functii speciale.

Operaţii implementate de device drivere de tip caracter

Open si Close

Functiile „open” si „close” realizeaza initializarea unui dispozitiv respectiv eliberarea resurselor specifice acestuia. O problemă care apare la implementarea funcţiei „`open`” este controlul accesului. Uneori este necesar ca un dispozitiv să fie deschis o singură dată la un moment dat; mai exact, nu se permite al doilea „`open`” înainte de „`close`”.

Read si Write

Functiile „read” si „write” sunt folosite pentru transferul datelor intre dispozitiv si user-space. „Read” citeşte datele de la dispozitiv şi le transferă în user-space, în timp ce „write” citeşte datele din user-space şi le scrie pe dispozitiv. Se primeste ca parametru un buffer reprezentand un pointer in user-space, motiv pentru care este necesară folosirea funcţiilor „`copy_to_user`” sau „`copy_from_user`”.Valoarea întoarsă este numărul de octeţi scrişi sau citiţi. Dacă valoarea întoarsă este mai mică decât parametrul size (numărul de octeţi ceruţi), înseamnă ca s-a realizat un transfer parţial. In cazul in care rezultatul nu este cel dorit aplicatia apeleaza functia pana cand se transferă numărul de date cerut.

Ioctl
Ofera posibilitatea de a realiza anumite operatii de control asupra dispozitivului fizic. Valoarea trimisă ca parametru reprezintă valoarea transmisă din user-space. Inainte de a implementa functia se aleg numerele ce corespund comenzilor.

Sincronizare - cozi de aşteptare
Cozile de aşteptare sunt extrem de utile în probleme de sincronizare. În Linux, o coadă de aşteptare este o listă în care sunt trecute procesele care aşteaptă un anumit eveniment.

Device drivere în Windows

Windows preia modelul I/O folosit de VMS. Acest model foloseste pachete numite I/O Request Pachets (IRP). Driverele primesc pachetele cu ajutorul rutinelor de dispach. Astfel cererea se poate rezolva imediat, poate fi pusa in asteptare sau driverul o poate trimite urmatorului device driver din stiva.

Accesul la spaţiul de adresă al procesului

Relativ la transferul datelor intre user-space si kernel-space se folosesc 2 abordari: BUFFERED I/O (managerul aloca un buffer in kernel-space) si DIRECT I/O.

Structuri de date importante

Fiecarui driver ii corespunde un obiect driver (DRIVER_OBJECT). Unui driver ii pot corespunde mai multe dispozitive descrise printr-un DEVICE_OBJECT.
Obiectul DRIVER_OBJECT contine cateva campuri importante: DriverUnload (pentru functia de unload a modulului), DeviceObject(lista dispozitivelor asociate driverului)

Obiectul DEVICE_OBJECT are urmatoarele campuri importante : DeviceExtension(bloc de memorie alocata de I/O manager la inregistrare), Flags (specifica strategia de transfer a datelor : buffered sau direct).

I/O Request Packet (IRP)

Un IRP este o structura alocata din memorie si e format din header si informatii pentru fiecare driver.

Campuri din header accesibile driverelor:

1.IoStatus : contine campuri ce trebuie completate de driver la terminarea operatiei: STATUS_SUCCES sau codul de eroare si numarul de octeti transferati in caz de succes.

2.Associated Irp.SystemBuffer: pointer catre buffer (daca se foloseste buffer)

3.MdlAddress: pointer folosit pentru maparea bufferului din user-space in kernel-space.

4.UserBuffer: pointer catre bufferul din user-space

Dupa header sunt plasate informatii pentru driverele din stiva. Functia `IoGetCurrentIrpStackLocation e folosita pentru a afla informatiile adresate driverului curent si returneaza o structura IO_STACK_LOCATION, cu urmatoarele campuri:`

1. `MajorFunction – identifica tipul operatiei`
2. `Parameters - uniune cu parametri pentru cereri: Read (ByteOffset-offsetul de la care se realizeaza citirea, Length-numarul de octeti cititi), Write(ByteOffset-offsetul de la care se realizeaza scrierea, Length-numarul de octeti scrisi), DeviceIoControl (IoControlCode-codul de control pentru ioctl, InputBufferLength-dimensiunea bufferului de intrare, OutputBufferLength-dimensiunea bufferului de iesire)`
3. `DeviceObject – identifica device-ul`
4. `FileObject- identifica fisierul pe care se face operatia`

Înregistrarea şi deînregistrarea dispozitivelor

In rutina DriverEntry driverul trebuie sa detecteze dispozitivele fizice din sistem si sa le anunte acestuia cu ajutorul functiei IoCreateDevice:

```
NTSTATUS IoCreateDevice(PDRIVER_OBJECT DriverObject,

                        ULONG DeviceExtensionSize,
```

```
PUNICODE_STRING DeviceName,
DEVICE_TYPE DeviceType,
LONG DeviceCharacteristics,
BOOLEAN Exclusive,
    PDEVICE_OBJECT *DeviceObject);
```

Funcţii de dispatch

Pentru ca sistemul sa stie ce functii sa apeleze in caz de o cerere din user space, driverul trebuie sa inregistreze una sau mai multe functii de dispacth.

Open si Close

La deschiderea si inchiderea dispozitivului se vor apela functii asociate cererilor de tip IRP_MJ_CREATE, IRP_MJ_CLOSE.
Aceste rutine sunt folosite de device driver pentru a initializa hardwarepul, a aloca buffere si a initia alte actiuni administrative.

Read si Write

Sunt tratate de cererile de tip IRP_MJ_READ si IRP_MJ_WRITE, device driverele pregatind in aceste rutine hardware-ul.

Ioctl

Apelul de genul DeviceIoControl reprezinta echivalentul Ioctl din Unix, functia care se foloseste de acest apel fiind asociata cu IRP_MJ_CONTROL.
Codul operatiei este dat de un numar pe 32 de biti, in care se codifica informatii despre apelul Ioctl.

Cleanup

Reprezinta o functie ascociata cu IRP_MJ_CLEANUP si se apeleaza atunci cand un proces renunta la o cerere.

Sincronizare – evenimente

Pentru realizarea sincronizarii intre thread-uri exista obiecte de sincronizare (KEVENT, KSEMAPHORE, KMUTEX, KTIMER, KTHREAD). In orice moment, aceste momente se pot afla in una din starile „signaled” sau

„not-signaled”, iar prin apelarea unei functii se poate ca un thread sa poata astepta ca un astfel de obiect sa ajunga in starea „signaled”.

KEVENT-ul reprezinta un eveniment Kernel si este de doua tipuri:
-eveniment de notificare – evenimentul trecut in starea „signaled” ramane in aceasta pana cand este resetat in mod explicit. In plus toate thread-urile care asteapta la acest eveniment sunt eliberate.
-eveniment de sincronizare – evenimentul trece automat in starea „not-signaled” cand un thread este eliberat.