6.GESTIUNEA DE INTARI SI IESIRI :

LACUREZEANU ADRIAN, 432A: a)Concepte fundamentale: gesiune de intreruperi, device driver.

BIZON CRISTIAN, 432A: b)Apelurile de sistem I/E in Linux si apelurile API de intrare/ iesire pentru Windows.

GESTIUNEA DE INTARI SI IESIRI

- PROIECT -

Profesor indrumator: **BIZON CRISTIAN, 432A**

STEFAN STANCESCU **LACUREZEANU ADRIAN, 432A**

Cuprins:

1. **Gestiunea de intreruperi**
   1.1 Sistemului de operare - gestionarea resurselor sistemului de calcul
   1.2 Gestionarea intreruperilor
   1.3 Surse de intrerupere
   1.4 Controlul intreruperilor
   1.5 Ierarhizarea intreruperilor

2. **Device drivers**
   2.1 Definitie, exemple
   2.2 Clasificare: device driver logice si fizice
   2.3 Mecanisme de functionare

3. **Apelurile de sistem I/E in Linux**
   3.1 Introducere
   3.2 Apelurile de sistem I/E in Linux

4. **Apelurile API de I/E in Windows**
   4.1 Introducere
   4.2 Apelurile API de I/E in Windows

1. Gestiunea de intreruperi

1.1 Sistemul de operare – gestionarea resurselor sistemului de calcul

Folosirea eficace a unui sistem de calcul trebuie sa asigure in orice moment executarea cat mai rapida a unui numar cat mai mare de sarcini. Pentru aceasta este nevoie de garantarea exploatarii la capacitatea maxima a resurselor sistemului.

Sarcinile care trebuiesc efectuate in sistem sunt complexe, diferite cu un caracter variat din punct de vedere al naturii lor, al momentului aparitiei sau al ordinii in care se succed.

Pentru coordonarea optima a activitatii componentelor sistemului de calcul, in aceste conditii de nedeterminare, au fost realizate functii special ale sistemului de operare cu scopul de a manipula modul de desfasurare a proceselor din sistem, functii care realizeaza componenta sistemului de operare de gestionare a resurselor sistemului de calcul.

O activitate esentiala pentru coordonarea executiei proceselor si a alocarii resurselor este actiunea de comunicare intre procese si sistemul de operare, de exemplu pentru a semnala atingerea unor anumite stari in cadrul unui proces (incheierea folosirii unei resurse, aparitia unei erori care retine desfasurarea procesului, etc.); aceasta comunicare este recomandata sa nu se efectueze prin intermediul UCP, pentru a nu se incarca aditional aceasta resursa critica a sistemului de calcul. Din aceasta cauza se foloseste o metoda de comunicare la nivel scazut (la nivel fizic) cu ajutorul sistemului de intreruperi.

Prin sistem de intrerupere se intelege o combinatie de componente hardware si software care asigura comunicarea intre componentele functionale elementare ale unui sistem de calcul prin intermediul intreruperilor. Componenta software a sistemului de intreruperi este inglobata in componenta de comanda si control a sistemului de operare prin functiile care realizeaza gestionarea intreruperilor.

1.2 Gestionarea intreruperilor

Exista o varietate de sarcini a caror aparitie in sistem face necesara abordarea lor imediata de catre UCP; vom numi aceste sarcini evenimente. Pe de alta parte, UCP nu trebuie sa afecteze timpul pentru a inspecta daca a aparut o astfel de solicitare in sistem, stiut fiind ca timpul UCP este o resursa critica a sistemului de calcul. Apare astfel o problema de comunicare de la componentele fizice sau logice ale sistemului de calcul catre UCP, problema rezolvata in sistemele de calcul moderne utilizand tehnica intreruperilor.

Prin intreruperea se intelege suspendarea temporara a rularii procesului care are alocata UCP, in momentul in care apare un eveniment in sistem, in scopul tratarii acestui eveniment de catre UCP.

Pentru ca o intrerupere sa fie functionala este nevoie sa fie indeplinite urmatoarele doua conditii:

- capacitatea UCP de a fi intrerupta, adica sa fie posibil ca UCP sa fie alocata altui proces, inainte de terminarea procesului in executiei la un moment dat;
- posibilitatea de a mentine parametrii procesului suspendat, pentru ca acesta sa poata fi reluat ulterior din punctul in care a fost interupt.
-

1.3 Surse de intrerupere

Sursele de intrerupere identifice sunt acele componente fizice (sau logice) ale sistemului care pot cere intreruperi in cazul aparitiei unor evenimente.

Cererile de intrerupere(CI) reprezinta semnalele transmise pentru UCP de catre sursele de intrerupere care solicita o intrerupere.

Se pot distinge mai multe tipuri de intreruperi in functie de natura surselor de intrerupere si a evenimentelor care pot genera o intrerupere:

a. Intreruperi hardware, generate de surse hardware

de exemplu:

- dispozitivele periferice pot solicita, prin intermediul CI, servicii sau actiuni specifice (unitatea de disc anunta terminarea unei operatii de citire/scriere sau producerea unei erori in timpul unei astfel de operatii; tastatura anunta activarea sau dezactivarea unei taste, etc.)
- dispozitivele hardware de supraveghere a functionarii optimale a sistemului de calcul pot transmite cereri de intrerupere in cazul sesizarii unor probleme de functionare (circuitele detectoare de paritate pot ilustra erori de paritate in timpul unui transfer de date, circuite specializate pot sesiza scaderea tensiunii de alimentare a sistemului, etc.)

b. Intreruperi software, care apar in timpul executiei unui proces; exista doua categorii
   - programaate - acestea apar cu scopul de a solicita anumite servicii; aceste intreruperi nu apar accidental, in timpul rularii programului, ci sunt generate intentionat, prin intermediul unei instructiuni specializate care se insereaza in program si care, atunci cand se executa, lanseaza o cerere de intrerupere.

   - de exceptie - acestea apar in momentul rularii unui program, cand se c executia unor operatii nepermise, de exemplu:
     - impartirea la 0
     - accesul la o zona de memorie protejata, etc.

   Aceste intreruperi sunt determinate de conditii de exceptie in rularea programului si sunt generate cu scopul de a evita cazuri imprevizibile sau eronate a procesului.

   Intreruperi sunt concomitente cu programul, adica daca o intrerupere de exceptie apare in timpul rularii unui program, cu un anumit set de date, atunci aceasta intrerupere va fi afisata de fiecare data cand programul respectiv se ruleaza cu acest set de date.

1.4 Controlul intreruperilor

In cursul rularii unui proces, exista diferite operatii importante, care nu trebuie intrerupte. De aceea, exista posibilitatea invalidarii unei CI.

Controlul intreruperilor face referire la posibilitatea acordata UCP de a agrea sau nu CI. In cod masina avem doua instructiuni complementare:

- EI (Enable Interrupt) - permite confirmarea unei cereri de intrerupere
- DI (Disable Interrupt) - nu permite confirmarea unei CI, pana la o noua instructiune EI. Prin intermediul acestei instructiuni se poate realiza invalidarea soft a unei cereri de intrerupere.

Exista 2 tipuri de intreruperi :

- mascabile – intreruperi invalidate soft
- nemascabile – nu pot fi invalidate soft – trebuie sa genereze obligatoriu intreruperea procesului in curs

1.5 Ierarhizarea intreruperilor

in sistem, pot fi activate, simultan, mai multe CI. Acestea sunt preluate in ordinea in care apar in sistem. De exemplu: se opreste, temporar, rularea unei rutine de intrerupere, pentru a rula o alta rutina de intrerupere, apelata de CI aparuta ulterior in sistem.

Se poate efectua o ierarhizare a intreruperilor, prin acordarea de prioritati de executie fiecarui nivel de intrerupere (in functie de sursa intreruperii fiecare CI se identifica printr-un numar, numit nivel de intrerupere). In cazul ierarhizarii intreruperilor, ordinea de servire a cererii de intrerupere va fi stabilita conform urmatoarelor principii:

- servirea unei cereri de intrerupere poate fi intrerupta numai de o cerere de intrerupere cu nivel de prioritate mai mare;
- intreruperile nemascabile au prioritate maxima;
- dupa incheierea unei rutine de tratare a intreruperii se reia executia intrerupta, care poate fi a unui proces sau a altei rutine de tratare a unei intreruperi cu prioritate mai mica.

2. Device drivers

2.1 Definitie, exemple

“device driver”-ul este programul ce face legatura intre comunicarea unui program de nivel mai inalt cu partile fizice ale unui calculator.

Device drivers sunt folosite pentru lucra cu:

- imprimante
- adaptoare video
- placi retea
- placi sunet
- diverse tipuri de magistrale locale
- magistrale de banda ingusta (mouse, tastatura, USB)
- magistrale pentru medii de stocare (hard disk, cd-rom,dvd-rom) de diverse tipuri (ATA, SATA)
- implementarea suportului pentru diverse sisteme de fisiere
- implementarea suportului pentru camere foto digitale si scanner

Atunci cand un program solocita o formalitate de la driverul respectiv acesta transmite o comanda catre device-ul in cauza. O data ce <<aparatul>> trimite date inapoi catre driver, acesta poate solicita la randul lui proceduri de la programul initiator.

Driver-ul reduce complexitatea programarii actionand ca un translator intre componentele folosite de un sistem si SO sau aplicatiile ce le folosesc.

O clasificare a driverelor I/O poate fi :

- drivere de sistem (standard) – au o anumita structura
- drivere de aplicatie – reprezinta o suma a functiilor pentru gestiunea perifericului

2.2 Clasificare : device drivers logice si fizice

Device Drivers pot fi de 2 tipuri:

- de tip logic – prelucreaza date ptr diferite clase de device-uri (de ex ethernet)

- de tip fizic

Driverele de tip fizic sunt manipulate de driverele de tip logic, acestea fiind manipulate la randul lor de sistemul de operare.

Orice dispozitiv I/O atasat unui computer are nevoie de un cod unic pentru a fi controlat de acesta. Acest cod numit „driver” este in general scris de producatorul dispozitivului respectiv si distribuit impreuna ca acesta pe un suport de tip CD-ROM. Din moment ce fiecare sistem de operare foloseste propriul tip de driver producatorii asigura mai multe astfel de tipuri astfel incat sa acopere cele mai populare sisteme de operare.

Pentru accesul la registri de control ai dispozitivului in cauza, driverele sunt adesea incluse in nucleul SO. In urma acestui lucru performanta este marita, dar in acelasi timp o fiabilitate precara deoarece un bug in orice astfel de driver poate afecta intregul sistem.

2.3 Mecanisme de functionare

Ocupatia unui driver, in caz general, este sa preia sarcini de la programe separate, superioare si sa verifice daca acestea sunt efectuate. Daca driverul se gaseste in stare de repaus la momentul respectiv, atunci el trece la realizarea sarcinii imediat. Daca, este ocupat cu o alta sarcina atunci va rula noile sarcini de indata ce este posibil in ordinea in care au fost emise.

Un prim pas este in realizarea unei cereri I/O este inspectarea ca parametrii de intrare sunt corecti, in caz contrar returneaza un mesaj de eroare. Daca cererea este valida se trece la traducerea ei din termeni abstracti in termeni concreti. Pentru un

driver al unui disc acest lucru inseamna identificarea pozitiei pe disc a blocului respectiv, verificarea ca motorul acestuia lucreaza, determinarea ca bratul este pozitionat pe cilindrul corespunzator si asa mai departe. Pe scurt driverul trebuie sa aleaga ce operatii sunt necesare si in ce ordine trebuie realizate.

Dupa ce driverul determina ce comenzi sa transmita catre controller, realizeaza acest lucru mutandu-le in registrii controllerului. Controlleri pot fi de 2 tipuri : simpli (care pot realiza o singura comanda) sofisticati (pot primii o serie de comenzi legate pe care apoi le realizeaza fara ajutorul SO).

Odata ce comanda sau comenzile au fost transmise se disting doua situatii. In marea majoritate a situatiilor, driverul astepta controllerul sa realizeaze diferite sarcini asa ca se blocheaza pana in momentul in care primeste semnalul de intrerupere, acesta deblocandu-l. In alte conditii insa operatia se efectueaza fara delay astfel incat driverul nu se blocheaza. In oricare din aceste 2 situatii, dupa ce operatiunea este completata trebuie verificata aparitia erorilor. Daca totul este in regula driverul poate avea date de transmis catre programul independent. In cele din urma returneaza informatii legate de starea in care se afla pentru a se putea raporta erorile catre programul care a solicitat procedura.

3. Apelurile de system I/E in Linux

3.1 Introducere. Linux

Linux este un sistem de operare de tip Unix, inceput in antul 1991 de catre Linus Torvalds, un student de origine finlandeze. El este in continuare principalul scriitor al sistemului de operare Linux, dar la codul de baza au contribuit deja mii de voluntari din intreaga lume cu peste 800 000 linii de cod. Trebuie spus ca sistemul este de o calitate foarte buna, rivalizind cu succes cu produse ale marilor firme care costa foarte mult. Diferenta este ca Linux este disponibil in surse oricui il doreste; poate fi obtinut contra cost sau gratuit de pe Internet. Linux evolueaza foarte rapid; noi versiuni ale nucleului apar la fiecare citeva zile.

Fisierul si procesul sunt abstractizari fundamentale pentru sistemului de operare. Un fisier este utilizat pentru a stoca informatiile necesare functionarii sistemului de operare si interactiunii cu utilizatorul.

Fiecarui fisier ii este asociat un nume (pentru a fi identificabil), un set de drepturi de acces si zone care contin informatii utile.

In Linux nu se face deosebirea intre fisierele aflate pe harddisk sau pe CD. Toate aceste fisiere fac parte din ierarhia unica a directorului `root`. Ca o solutie, sistemele de fisiere se vor monta intr-unul din directoarele sistemului de fisiere radacina.

In Windows exista mai multe ierarhii, cate una pentru fiecare partitie. Spre deosebire de Linux, delimitatorul intre numele directoarelor dintr-o cale se noteaza '`\`', iar numele ierarhiei va fi scris de forma `C:\`, `D:\` sau `\\FILESERVER\myFile` (pentru retea). Ca si Linux, Windows foloseste '.' pentru directorul curent si '..' pentru directorul parinte.

3.2 Apelurile de I/O

- **open**

  Pentru deschiderea unui fisier se foloseste functia open, avand antetul:

```
int open(const char *pathname, int flags);                 /* deschidere */
int open(const char *pathname, int flags, mode_t mode);    /* creare */
```

- **creat**

  Pentru crearea unui fisier se foloseste functia creat, avand antetul:

```
int creat(const char *pathname, mode_t mode);
```

- **close**

  Pentru inchiderea unui fisier se foloseste functia close, avand antetul:

```
int close(int fd);
```

- **unlink**

  Pentru stergerea unui fisier de pe harddisk se foloseste functia unlink, avand antetul:

```
int unlink(const char *pathname);
```

- **read**

  Pentru citirea din fisier se foloseste functia read, avand antetul:

```
ssize_t read(int fd, void *buf, size_t count);
```

  Functia read intoarce numarul de octeti efectiv cititi, cel mult `count`. Valoarea minima este de `1` octet, iar cand se ajunge la sfarsitul de fisier se va intoarce `0`.

- **write**

  Pentru scrierea in fisier se foloseste functia write, avand antetul:

```
ssize_t write(int fd, const void *buf, size_t count);
```

Observatie: Pentru read/write exista versiunile pread/pwrite, care permit specificarea unui offset in fisier de la care sa se efectueaze operatia de citire/scriere.

Pozitionarea in fisier (lseek)

- **lseek**

  Functia lseek permite mutarea cursorului unui fisier la o pozitie absoluta sau relative.

```
off_t lseek(int fd, off_t offset, int whence)
```

  Parametrul `whence` reprezinta pozitia relativa de la care se face deplasarea:

- `SEEK_SET` - fata de pozitia de inceput

- `SEEK_CUR` - fata de pozitia curenta
- `SEEK_END` - fata de pozitia de sfarsit

Observatie: lseek permite si pozitionari dupa sfarsitul fisierului. Scrierile care se fac in astfel de zone nu se pierd, ceea ce se obtine fiind un fisier cu *goluri*, o zona care este *sarita* - nu este alocata pe disc.

Pentru aceasta functie exista si o versiune lseek64 la care offset-ul este pe 64 de biti.

4. Apelurile API de I/E in Windows

4.1 Introducere. Windows API

Windows API (*Application Programming Interface*) este o interfata a sistemului de operare Windows, utilizata pentru programarea aplicatiilor. Windows API este cunoscuta, in general, sub numele de *Win32 API*, dar aceasta denumire(*Windows API*) este mai corecta deoarece este utilizata atat pe sistemele de operare Windows pe 32 biti, cat si pe sistemele de operare Windows pe 64 biti.

Unelte, cat si documentatia necesara programatorilor pentru a realiza aplicatii folosind Windows API se pot gasti in Microsoft Windows SDK(*Software Development Kit*)

Cu ajutorul lui Windows API, programatorul are acces direct la multe functii de nivel de baza ale sistemului de operare, putand crea si edita cu usurinta diferite aplicatii.

Windows API contine diverse servicii pentru aplicatii care se bazeaza pe ferestre grafice (de exemplu *Windows*). Prin aceasta interfata de programare utilizatorilor le este permis sa realizeze o interfata grafica pentru aplicatiile create, sa aiba acces la sistemul calculatorului, la memoria acestuia, la dispozitivele de intrare sau de iesire, sa implementeze in aceste aplicatii sunete, poze, filme sau functiuni de retea.
Programarea cu Windows API inseamna primirea, interpretarea, trimiterea de „mesaje”

catre „ferestre”, sau „controale” (obiecte controlabile - ex. ToolBox, EditBox, Button, Text, CheckBox).

Alegem ca apelurile sa faca parte din Windows API, pentru a descrie comportamentul operatiilor de intrare-iesire al sistemului de operare Windows, deoarece Windows API este cel mai apropiat de kernel-ul Windows.

Operatiile de intrare/iesire sunt mai lente decat operatiile de procesare din cauza intarzierilor cauzate de:

- timpul de access la sectoarele hard-disk-urilor
- rata de transfer scazuta dintre hard-disk si memoria RAM
- transferul de date peste retea

4. 2 Apelurile API de I/E

- **deschiderea sau crearea unui fisier**

Cu ajutorul functiei CreateFile se poate crearea sau deschide un fisier (si intoarce in ambele cazuri un handle asociat cu fisierul):

```
HANDLE CreateFile(
   LPCTSTR lpFileName,
   DWORD dwDesiredAccess,
   DWORD dwShareMode,
   LPSECURITY_ATTRIBUTES
lpSecAttributes,
   DWORD dwCreationDisposition,
   DWORD dwFlagsAndAttributes,
   HANDLE hTemplateFile
);
```

- **inchiderea unui fisier**

Prin apelul functiei CloseHandle se eliberaza structurile de fisiere asociate procesului si a handle-ului acelui fisier.

```
BOOL CloseHandle(HANDLE hObject);
```

- **citirea dintr-un fisier**

Citirea dintr-un fisier se face cu ajutorul functiei ReadFile. Aceasta functie copiaza un numar de octeti (de la pozitia curenta a cursorului de fisier) intr-un buffer si intoarce numarul de biti citii intr-o variabila.

```
BOOL ReadFile(
      HANDLE hFile,
      LPVOID lpBuffer,
      DWORD nNumberOfBytesToRead,
      LPDWORD lpNumberOfBytesRead,
      LPOVERLAPPED lpOverlapped
);
```

Functia ReadFile returneaza o valoare diferita de zero in caz de succes, si zero altfel. Daca valoarea returnata este diferita de zero, dar numarul de octeti cititi este zero, atunci inseamna ca s-a ajuns la sfarsitul fisierului.

- **scrierea intr-un fisier**

Functia WriteFile copiaza intr-un fisier un numar specificat de octeti dintr-un buffer si returneaza numarul de octeti pe care ia copiat.

```
BOOL WriteFile(
      HANDLE hFile,
      LPCVOID lpBuffer,
      DWORD nNumberOfBytesToWrite,
      LPDWORD lpNumberOfBytesWritten,
      LPOVERLAPPED lpOverlapped
);
```

Parametrii functiei WriteFile au aceleasi semnificatii cu parametrii functiei ReadFile, adaptate pentru operatii de scriere.

- **pozitionarea intr-un fisier**

Functia care face pozitionarea intr-un fisier se numeste SetFilePointer. Aceasta functie contine un cursor de fisier care indica locul de unde vor incepe noile operatii de scriere, citire, etc.

```
DWORD SetFilePointer(
         HANDLE hFile,
         LONG lDistanceToMove,
         PLONG lpDistanceToMoveHigh,
         DWORD dwMoveMethod );
```

Campurile `lDistanceToMove` si `lpDistanceToMoveHigh` specifica numarul de octeti cu care se muta cursorul; cele doua campuri de 32 de biti formeaza o valoare de 64 de biti. Uzual cel de-al doilea camp este NULL.

Apelul returneaza noul loc in care va fi pozitionat cursorul, daca `lpDistanceToMoveHigh` este NULL.

Varianta extinsa SetFilePointerEx a functiei SetFilePointer retine locul cursorului intr-un singur camp, in loc de doua campuri separate. Functia SetFilePointer face ca operatia de pozitionarea a cursorului sa fie mult mai usoara.

Bibliografie

- http://wikipedia.com
- Operating Systems System Calls and I/O by Henry Newman
- Silberschatz A., Galvin P.B. and Gagne G. (2005). Operating Systems Concepts, 7th edn. John Wiley & Sons
- Tanenbaum A.S. (1992). Modern Operating Systems. Englewood Cliffs NJ: Prentice-Hall.