GESTIUNE DE INTRARI/IESIRI

Cuprins cu contribuţia fiecăruia, cu bibl şi ref proprii

Cuprins:

1.Introducere

1.1 Ce este un sistem de operare?
1.2 Incadrarea gestiunii de intrare/iesire in atributiile sistemului de operare
1.3 Tipuri de dispozitive de intrare/iesire
1.4 Principii hardware si software

Ancuta Razvan 442A

Bibliografie :

1. www.wikipedia.com
2. MODERN OPERATING SYSTEMS
   SECOND EDITION
   by Andrew S. Tanenbaum
3. Understanding the Linux Kernel, 3rd Edition , By Daniel P. Bovet, Marco Cesati

2.Gestiunea de întreruperi

Boboc Octavian 442A

3.Device drivere

3.1 Definitie, exemple
3.2 Clasificare; device drivere logice si fizice
3.3 Mecanisme de functionare
3.4 Exemple

Coman Mihai 442A

Bibliografie:

1) Andrew S. Tanenbaum, Sisteme de operare moderne

4.Gestiune din sistemul de operare

Lixandru Nicolae Petrut 442A

Bibliografie:

1) Andrew S. Tanenbaum, Sisteme de operare moderne

5.Gestiune de catre utilizatori

5.1 Prezentare generala
5.2 Principiile dispozitivelor harware de Intrare/iesire
5.3 Caracterizarea dispozitivelor
5.4 Intreruperi
5.6 Accesarea Directa a Memoriei (DMA)

IONESCU D S Catalin 442A

Bibliografie:

John Levine - "Linkers and Loaders" (http://www.iecc.com/linker)
Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004; www.wikiepedia.com.

6.Apeluri de sistem i/o in Linux

6.1 Ce este linux-ul?
6.2 Apeluri de sistem (system calls)
6.3 Apeluri de sistem de intrare/iesire (in linux)
6.3.1 Apeluri pentru i/o sincrone(aplicatie linux;exemple)
6.3.2 Apeluri pentru i/o asincrone(aplicatie linux;exemple)

Ancuta Razvan 442A

Bibliografie :

1.www.wikipedia.com
2.Understanding the Linux Kernel, 3rd Edition , By Daniel P. Bovet, Marco Cesati
3.CS360 Lecture notes -- Introduction to System Calls (I/O System Calls) by Jim Plank
4.Operating Systems System Calls and I/O by Henry Newman

7.Apeluri API de i/o pentru Windows

1. API/NT
2. Win32/Linux
3. Tipuri de apeluri API
4. Apeluri API pentru Intrari / Iesiri in Windows – prezentare
5. Cateva categorii de apeluri Win32 API

Gavrila Cosmin 442A

Bibliografie :

1. S. Tanenbaum - **Sisteme de operare moderne**
2. www.wikipedia.com
3. Operating Systems System Calls and I/O by Henry Newman

1. INTRODUCERE

1.1 Ce este un sistem de operare?
1.2 Incadrarea gestiunii de intrare/iesire in atributiile sistemului de operare
1.3 Tipuri de dispozitive de intrare/iesire
1.4 Principii hardware si software

1.1 Ce este un sistem de operare?

Un sistem de operare este un soft care imparte si administreaza resursele unui sistem de calcul si ofera programatorilor o interfata mai usor de folosit pentru a folosi puterea de procesare a sistemului hardware intr-un mod organizat.

Sistemul de operare proceseaza datele intrare ,aloca resurse si organizeaza task-urile (un task este ceva ce trebuie sa faca sistemul).

Principalele functii ale sistemului de operare sunt:

- alocarea memoriei si resurselor de procesare
- sa organizeze task-urile primite in functie de prioritate
- gestionarea de intrari si iesiri a sistemului
- organizarea informatiei in sisteme de fisiere (1)

1.2 Incadrarea gestiunii de intrare/iesire in atributiile sistemului de operare

Toate computerele au componente hardware pentru a citi date de intrare si a afisa sau pastra rezultatele procesarii.Aceste dispozite de intrare sau iesire pot fi tastaturi, sisteme de achizitii de date, monitoare, imprimante si multe altele. Toate acestea au nevoie de un sistem pentru a le utilize, astfel ca sistemele de operare au un subsistem de intrare/iesire specializat in acest sens. (2)

Aici tre despre spaţiile de adrese mem şi I/O

Se stie din arhitectura microprocesoarelor ca spatiul de adrese presupune o serie de adrese, fiecare corespunzand unui registru fizic sau virtual, un **i/o device** (dispozitiv de intrare/iesire), un sector de disk sau alta entitate fizica sau logica. O adresa pointeaza spre locul in care se gaseste informatia cautata.

Gestiunea de intrare/iesire dintr-un sistem de calcul se realizeaza prin doua metode complementare:

a) **Memory-mapped I/O** (**MMIO**) (harta de adrese de intrare/iesire)

Cu aceasta metoda microprocesorul foloseste aceeasi magistrala si aceleasi registre de adresare pentru memorie si **i/o device-**uri. Pentru aceasta o portiune din harta de adrese este rezervata (temporar sau permanent) pentru intrare/iesire.

Un plus al acestei gestiuni este ca microporocesorul necesita mai putina logica interna, mai ieftin si mai repede de implementat adica are influente **RISC** (reduced instruction set computing).Orientarea spre arhitecturi de 64 de biti face ca numarul de adrese sa nu mai fie o problema, deci se poate rezerva un numar suficient de adrese pentru intrare/iesire fara ca sa incurce adresarea memoriei.

b) **Port-mapped I/O** (**PMIO)(**harta de porturi de intrare/iesire**)**

Microprocesorul foloseste un set de registre speciale pentru acesarea **i/o device-**urilor(in general la microprocesoare INTEL). **Device-**urile au o harta de adrese special rezervata pentru ele si in anumite cazuri o intreaga magistrala dedicata sau instructiuni dedicate (“in”, “out” pentru INTEL).

Avantajul acestui mecanism este la microprocesoare cu spatiu de adresare limitat ceea ce duce la o gestionare mai buna a memoriei principale, dar se poate spune ca aduce un plus asupra usurintei cu care poate fi scris/citit un cod in limbaj de asamblare datorita instructiunilor dedicate.([1])

Conceptele fundamentale pe care se bazeaza acest subsistem sunt

- gestiune de intreruperi (**interrupt handlers**)
- driveri de dispozitive (**device drivers**)
- gestiune de catre sistemul de operare (**kernel level mode**)
- gestiune de catre utilizator(**user level mode**)
- principiile de intrare/iesire hardware (**input/output hardware**)

([2])

1.3 Tipuri de dispozitive de intrare/iesire

Dispozitivele (devices) de intrare/iesire pot fi categorizate in mare in doua parti: **block devices** si **character devices**.

Block devices sunt cele care memoreaza informatia in blocuri de dimensiuni fixe (de obicei de la 512 la 32,768 bytes) fiecare cu adresa proprie. Proprietatea fundamentala a acestor este ca informatia poate fi citita doar din blocurile care intereseaza , desigur pentru un cost de timp (exemplu hard-diskurile).

Un character device trimite sau primeste un sir de caractere (**stream -uri**) fara nici o organizare a informatiei pe adrese, de aceea nu se poate accesa o anumita portiune.

Alte device-uri nu se pot integra in una din aceste categorii, cum ar fi ceasurile care nu fac altceva decat sa dea intreruperi sau afisoarele cu memorie.

Sistemule de operare trebuie sa ia in considerare si rata de transfer a informatiei a dispozitivelor de intrare/iesire : ([2])

Device	Rata de transfer
Tastatura	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Canal telefonic	8 KB/sec
Dual ISDN	16 KB/sec
Imprimanta Laser	100 KB/sec
Scaner	400 KB/sec
Ethernet	1.25 MB/sec
USB (Universal Serial Bus)	1.5 MB/sec
Camera digitala	4 MB/sec
IDE disk	5 MB/sec
40x CD-ROM	6 MB/sec
Fast Ethernet	12.5 MB/sec
ISA bus	16.7 MB/sec
EIDE (ATA-2) disk	16.7 MB/sec
FireWire (IEEE 1394)	50 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
SCSI Ultra 2 disk	80 MB/sec
Gigabit Ethernet	125 MB/sec
Ultrium tape	320 MB/sec
PCI bus	528 MB/sec
Sun Gigaplane XB backplane	20 GB/sec

([2])

De unde tabelul?

1.4 Principii hardware si software

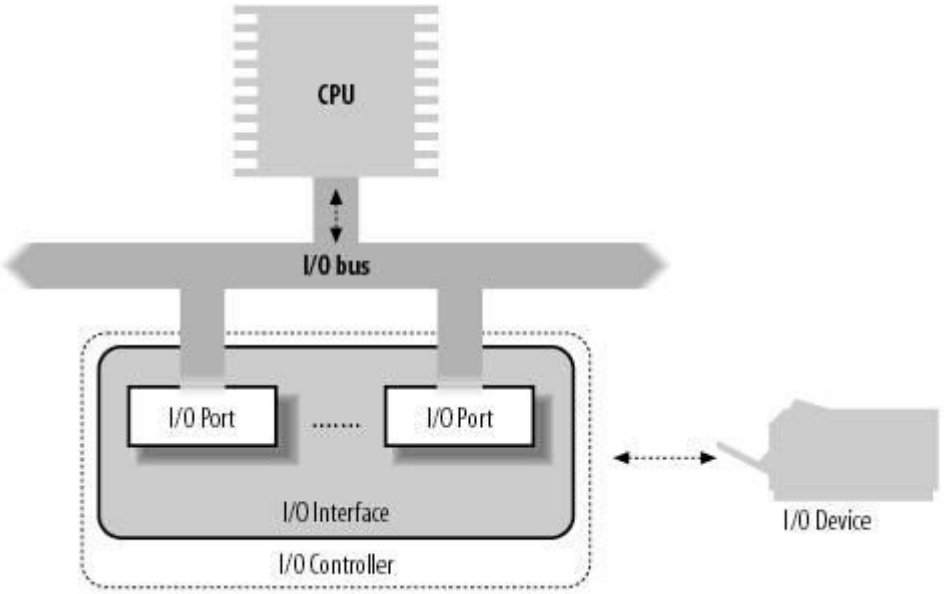
Orice sistem de calcul are magistrale prin care procesorul, memoria si dispozitivele de intrare/iesire comunica intre ele.Aceste magistrale sunt niste canale primare de comuncatii din interiorul sistemului si se numesc **bus-uri**. De exemplu intr-un computer sunt mai multe tipuri de bus-uri PCI (Peripheral Component Interconnect), ISA, EISA, MCA, SCSI, USB si altele. Mai multe bus-uri de diferite tipuri sunt legate impreuna de un dispozitiv hardware numit **bridge**.

De o importanta mai mare sunt:

- **front side bus** – intre procesor si memoria principala(RAM)
- **back side bus –** intre procesor si memoria cache
- **i/o bus –** intre procesor si un dispozitiv de intrare/iesire

ar trebui si o definiţie a lor system bus unde este?_ si I/O bus care este?

Figura de mai jos presupune arhitecura de intrare/iesire a unui calculator:



De unde poza? Nu o puteai face pe rom, sa includa despre ce ai vorbit in text?

I/O controller (adapter) din figura reprezinta partea electronica a **device**-ului de exemplu la computerele personale presupune placuta cu circuite electronice ce este introdusa in sloturile de expansiune.

I/O interface se refera la interfata sub care este conectat **device**-ul, de exemplu PCI.

Fiecare dispozitiv conectat la i/o bus are un set de adrese numite **porturi de i/o** dupa cum s a mentionat in **1.2**.Numarul de adrese pe care le poate atribui depinde de arhitectura sistemului si sunt pe 2^k biti, iar doua sau

mai multe adrese se pot trata ca una singura(exemplu doua adrese consecutive de 8 biti se pot considera o singura adresa de 16 biti).

La sfarsit pentru a introduce o apropiere spre usurinta programarii fara a sacrifica performanetele se introduce un set de registre intre procesor si interfata **device**-ului ca in figura de mai jos: ([3])

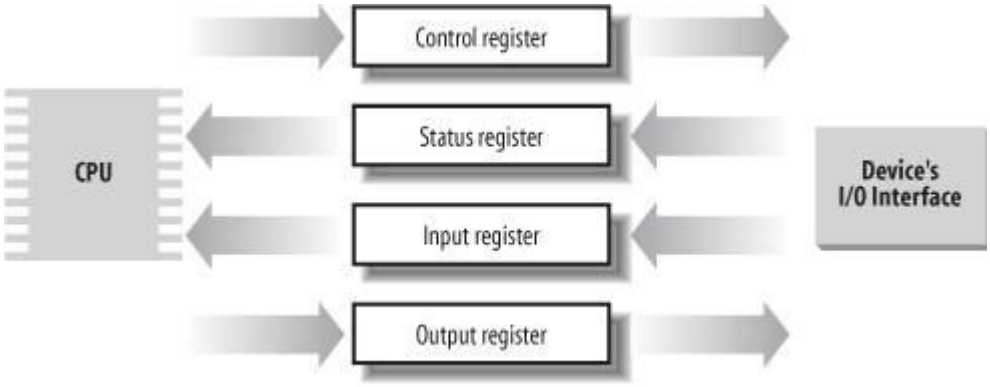


Figura de unde este și ce rost are in GIO, explică, și cu ref la text([3])

Microprocesorul scrie in registrul **Control** comenzile care trebuie trmise device-ului respectiv si citeste o valoare care presupune starea interna a device-ului din registrul **Status**. Procesorul scrie pe device punand in registrul **Output** si citeste de pe device citind de pe registrul **Input.** ([3])

DMA (Direct memory access) este o tehnologie a calculatoarelor moderne care presupune avantajul de a permite unor subsisteme hardware(device-uri) de a citi sau scrie din/in memoria independent de microprocesor. Fara DMA, utilizand **Programmed input/output (PIO)** procesorul este ocupat pe intreaga perioada in care se efectueaza operatia de intrare/iesire si nu poate efectua alte operatii. Cu DMA, procesorul initiaza transferul, efectueaza alte operatii si cand primeste o intrerupere de la DMA cand s-a terminat operatia de intrare/iesire.

Ancuta Razvan 442A

Bibliografie

1. www.wikipedia.com
2. MODERN OPERATING SYSTEMS
   SECOND EDITION
   by Andrew S. Tanenbaum

3. Understanding the Linux Kernel, 3rd Edition , By Daniel P. Bovet, Marco Cesati

Cam putin

2.Gestiunea de întreruperi

O întrerupere este un semnal care informează un program cu privire la faptul că a apărut un eveniment. Când un program primeşte un semnal de întreruperi, ia o anumită acţiune (care poate fi şi ignorarea semnalului). Semnalele de întreruperi pot determina programul să îşi suspende rularea pentru a trata întreruperea. (*bibliografie*: **www.webopedia.org**)

Întreruperea este în general definită ca un eveniment care modifică secvenţa de instrucţiuni executată de un procesor Care este rolul lor pentru dialogul cu disp I/O şi de ce sunt necesare..

Rolul întreruperilor este acela de găsire a unei modalităţi de diverta procesorul de la execuţia normală. Când primeşte un semnal de întrerupere, acesta trebuie să lase ce are de făcut şi să treacă la o altă activitate; acest lucru este făcut de procesor prin salvarea valorii curente a numărătorului de program (PC) (în speţă, conţinutul registrelor eip şi cs) în stivă şi prin plasarea unei adrese legate de tipul întreruperii în numărătorul de program. (*bibliografie*: **O'Reilly- Understanding the Linux Kernel**)

Aceste întreruperi pot fi *sincrone* sau *asincrone*.

Cele **sincrone** (numite şi „excepţii") sunt produse de unitatea de control a CPU în timp ce se execută instrucţiuni, şi se numesc sincrone datorită faptului că unitatea de control le produce numai după terminarea execuţiei unei instrucţiuni. terminarea este o excepţie?.Terminarea în sine nu este o excepţie.

Întreruperile **asincrone** (numite şi ‚întreruperi') sunt generate de alte mecanisme hardware care sunt aceste alte? la momente arbitrare. Acestea sunt, de exemplu, apăsarea unui buton de la tastatură, sau imprimanta).

În timp ce, spre exemplu, cele asincrone sunt cauzate de mecanismele de I/O, cele sincrone sunt cauzate fie de condiţii anormale, fie de erori de programare care trebuiesc gestionate de kernel. (*bibliografie*: **O'Reilly- Understanding the Linux Kernel**)

Tratarea întreruperilor este unul dintre cele mai sensibile task-uri executate de kernel, din moment ce acesta trebuie să satisfacă următoarele constrângeri:

Întreruperile pot surveni în orice moment, când kernel-ul ar dori să termine alt lucru pe care încerca să îl facă. Scopul kernel-ului este, prin urmare, să gestioneze întreruperea într-un timp cât mai scurt şi să întârzie cât mai multă procesare cu putinţă. De asemenea, tot din acest motiv, acesta trebuie să poată gestiona o întrerupere în timp ce o alta se produce.

Deşi kernel-ul acceptă un nou semnal de întreruperi în timp ce gestionează un altul, unele regiuni critice există în interiorul codului kernel-ului în care întreruperile trebuiesc dezactivate. Aceste regiuni trebuiesc limitate cât mai mult, pentru că kernel-ul, şi în particular gestionarii de întreruperi, trebuie să lucreze în majoritatea timpului cu întreruperile activate.

Întâi defineşte ce înseamnă masca IT.

Masca de întreruperi reprezintă o entitate prin care putem cataloga o întrerupere ca fiind de nerecunoscut pentru procesor.

Întreruperile asincrone pot fi de 2 feluri, anume *mascabile* sau *nemascabile*.

Întreruperi mascabile:

- ➔ pot fi generate datorită cererilor de întreruperi (IRQ)
- ➔ sunt de 2 tipuri: - mascate - sunt ignorate de unitatea de control atâta timp cât ramân mascate
  - nemascate

Întreruperi nemascabile:

- ➔ sunt întotdeauna recunoscute de procesor
- ➔ numai anumite evenimente critice (ca de exemplu căderi de sistem) le pot genera.

Întreruperile sincrone (excepţii) pot fi la rândul lor:

-> excepţii *detectate de CPU* - sunt generate atunci când procesorul detectează condiţii anormale în timpul execuţiei unei instrucţiuni;

-> *erori corectabile cu restartare*- în general pot fi corectate; odată corectate, programul poate fi restartat.

-> *capcane* – se execută instrucţiunea capcanei defineşte capcana; *definiţie: capcane = semnale de intrerupere generate de programe; se mai numesc şi întreruperi software sau excepţii* atunci când kernel-ul dă controlul programului, este lăsat să îşi continue execuţia fără pierderi de continuitate.

-> *erori severe cu ieşire* - de exemplu căderi de sistem; unitatea de control nu poate stoca în acest caz locaţia precisă a instrucţiunii căreia i se datorează excepţia.

-> excepţiile *programate* – au loc la cererea utilizatorului; acestea sunt declanşate de instrucţiunile int şi int3, sau de into şi bound, atunci când condiţia verificată nu e adevărată; sunt gestionate de unitatea de control ca şi capcane şi sunt adesea numite întreruperi software; acestea se folosesc pentru implementarea apelurilor de sistem şi pentru a înştiinţa debugger-ul despre un anumit eveniment. (*bibliografie*: **O'Reilly- Understanding the Linux Kernel**)

Mai mult, rolul lor în debug

Sistemele de operare oferă proceselor care rulează în modul User un set de interfeţe pentru a interacţiona cu dispozitivele hardware cum ar fi procesorul, discurile, imprimanta, etc. Aceste interfeţe au câteva avantaje. În primul rând, programarea devine mai uşoară, userii nemaifiind nevoiţi să studieze caracteristicile programării la nivelul dispozitivelor hardware. În al doilea rând, sistemul este mult mai bine securizat, din moment ce kernel-ul poate verifica corectitudinea cererii la nivelul interfeţei înainte de a încerca să o satisfacă. În al treilea rând, aceste interfeţe fac programele mai portabile din moment ce ele pot fi compilate şi executate corect în cazul oricărui kernel care oferă acelaşi set de interfeţe.

Majoritatea excepţiilor datorate CPU sunt interpretate de ***Linux*** şi La Windows cum este, ai prezentat? drept *condiţii de eroare*. Când una dintre ele are loc, kernel-ul trimite un semnal procesului care a cauzat excepţia pentru a-l înştiinţa cu privire la condiţia de anormalitate. Dacă, de exemplu, un proces realizează o divizare cu 0, intervine o excepţie „de eroare de divizare", iar gestionarul de excepţii corespunzător trimite un semnal SIGFPE procesului actual, care apoi ia măsurile necesare pentru a-şi reveni sau – în cazul în care nu avem gestionar de excepţii pentru semnal – pentru a ieşi. (*bibliografie*: **O'Reilly- Understanding the Linux Kernel**)

Gestionarul de excepţii are o structură ce ţine de 3 paşi:
-salvarea conţinutului majorităţii registrelor în stiva Kernel Mode (această parte e codată în limbaj de asamblare)
-gestionarea excepţiei printr-o funcţie C
-ieşirea din gestionar cu ajutorul funcţiei ret_from_exception

La Windows -> există 2 tipuri de gestiune de întreruperi:
->gestiune de întreruperi structurată (SEH)
->gestiune de întreruperi vectorizată (VEH)

Gestiunea de întreruperi structurată foloseşte aşa-numitele „noduri de excepţie bazate pe stivă". La arhitectura x86, Microsoft foloseşte o valoare a unui pointer la FS:[0] pentru a pointa la frame-ul gestionarului current de excepţii. Informaţia despre frame include o adresă pentru apelare atunci când intervine o excepţie.

Gestiunea de întreruperi vectorizată a fost introdusă odată cu *Windows XP*, şi reprezintă o extensie a gestiunii structurate de excepţii. O aplicaţie poate înregistra o funcţie pentru a supraveghea şi gestiona toate excepţiile. Gestionarii vectorizaţi nu se bazează pe frame-uri, aşadar, se poate adăuga un gestionar care va fi apelat, indiferent unde ne aflăm într-un frame de apel. Gestionarii vectorizaţi sunt apelaţi în ordinea în care au fost adăugaţi.

Pentru a adăuga un gestionar vectorizat, se foloseşte *Windows API* – un set de interfeţe de programare a aplicaţiilor (***a****pplication* ***p****rogramming* ***i****nterfaces*) prezent în sistemele de operare Windows. Toate programele Windows, cu excepţia programelor de consolă trebuie să interacţioneze cu Windows API, indiferent ce limbaj s-ar folosi. Un acces mai limitat într-un sistem Windows, cel mai adesea solicitat de către *device-drivere*, este oferit de către WDF (*Windows Driver Foundation*) în versiunile curente de Windows.
Un kit de dezvoltare software (*SDK - **s**oftware **d**evelopment **k**it*) este valabil în Windows, prin care se oferă documentaţie şi unelte pentru a oferi posibilitatea utilizatorilor să creeze software folosindu-se de Windows API şi de tehnologiile Windows corespunzătoare. *(bibliografie:* **www.winasm.net – x86 Assembly Community***)*

În cazul gestionării întreruperilor, aceasta *depinde de tipul întreruperii (bibliografie:* **Understanding the Linux Kernel, 3rd Edition***)*:
-întreruperile de I/O
-întreruperi de timer – un timer determină apariţia întreruperii; acest tip de întrerupere spune kernel-ului că un interval de timp anume s-a terminat
- sunt gestionate în majoritatea cazurilor ca întreruperi I/O
* activităţilor computerizate le sunt asociate măsurări de timp, adeseori ascunse utilizatorului. Ca exemplu ar fi cazul în care ecranul este închis automat după ce utilizatorul încetează să mai utilizeze computerul; acest lucru se datorează unui timer care permite kernel-ului să ţină evidenţa timpului scurs de când utilizatorul a apăsat o tastă sau a mişcat mouse-ul.
* există două tipuri principale de măsurări de timp care pot fi efectuate de kernel:
- păstrează timpul şi data curente astfel încât acestea pot fi returnate programelor user cu ajutorul *time(), ftime() şi gettimeofday()*;
- menţine mecanismele timerelor care pot notifica kernel-ul sau un program user de faptul că un anumit interval de timp s-a scurs.

-întreruperi interprocesor (IPI) – un CPU a determinat apariţia unei întreruperi la un alt CPU în cazul unui sistem multiprocesor.
* În sisteme multiprocesor, Linux utilizează 3 tipuri de întreruperi interprocesor:
`CALL_FUNCTION_VECTOR` (*vector* `0xfb`) – trimisă tuturor CPU în afară de expeditor, forţând aceste procesoare să ruleze o funcţie;
`RESCHEDULE_VECTOR` (*vector* `0xfc`) – când un CPU recepţionează acest tip de întrerupere, gestionarul corespunzător numit

reschedule_interrupt( ) se limitează la a confirma întreruperea. Replanificarea se face automat la întoarcerea de la întrerupere;

INVALIDATE_TLB_VECTOR (*vector* 0xfd) – trimisă tuturor CPU în afară de expeditor, forţându-le să invalideze buffer-ele de translaţie.Gestionarul corespunzător este *invalidate_interrupt( )*.

Gestiunea întreruperilor I/O (bibliografie: **Understanding the Linux Kernel, 3rd Edition***)*:

În general, un gestionar de întreruperi I/O trebuie să fie destul de flexibil pentru a servi mai multe mecanisme în acelaşi timp – de exemplu, în cadrul arhitecturii magistralei PCI, mai multe dispozitive ar putea împărţi aceeaşi linie IRQ. Aceasta înseamnă că vectorul de întreruperi, singur, nu spune toată povestea. În următorul tabel, acelaşi vector 43 e asignat portului USB şi plăcii de sunet. Cu toate acestea, unele dispozitive hardware găsite în arhitecturile de PC mai vechi (cum ar fi ISA) nu operează sigur dacă linia lor IRQ este împărţită cu alte dispozitive.

IRQ	INT	Hardware device
0	32	Timer
1	33	Keyboard
2	34	PIC cascading
3	35	Second serial port
4	36	First serial port
6	38	Floppy disk
8	40	System clock
10	42	Network interface
11	43	USB port, sound card
12	44	PS/2 mouse
13	45	Mathematical coprocessor
14	46	EIDE disk controller's first chain
15	47	EIDE disk controller's second chain

un exemplu de corespondenţă a IRQ cu dispozitivele I/O

(bibliografie tabel: **Understanding the Linux Kernel, 3rd Edition***)*

Flexibilitatea gestionarului de întreruperi poate fi atinsă prin 2 metode distincte, astfel:

Ar mai trebui dezvoltat tratarea IT, eventual un exemplu, partea finală de extins

IRQ sharing - gestionarul de întreruperi execută mai multe rutine de servire a întreruperilor (ISR). Fiecare dintre acestea este o funcţie asociată unui singur dispozitiv care împarte linia IRQ. Deoarece nu este posibil să se ştie dinainte cărui dispozitiv corespunde IRQ, fiecare rutină de servire a întreruperilor este executată pentru a verifica dacă dispozitivul corespunzător acesteia necesită atenţie; în caz de adevăr, ISR execută toate operaţiile necesare a fi luate atunci când un dispozitiv generează o întrerupere.

IRQ alocare dinamică – o linie IRQ e asociată cu un driver în ultimul moment posibil; de exemplu, linia IRQ a floppy-disk-ului este alocată numai atunci când un user accesează floppy-disk-ul. În felul acesta acelaşi vector IRQ este folosit de mai multe dispozitive hard chiar dacă ele nu pot impărţi linia IRQ; bineînţeles, dispozitivele hard nu pot fi folosite în acelaşi timp.

Nu toate măsurile care trebuiesc luate când are loc o întrerupere au aceeaşi urgenţă. ***Linux*** împarte aceste măsuri în *3 clase*:

- măsuri critice – exemple - confirmarea unei întreruperi către PIC, reprogramarea PIC sau a device-controlerului, sau updatarea structurilor de date accesate atât de către dispozitiv, cât şi de procesor.

  - pot fi executate rapid
  - trebuiesc luate cât mai repede cu putinţă
  - sunt luate de către gestionarul de întreruperi imediat, cu întreruperile mascabile dezactivate.

- măsuri non-critice – exemplu - updatarea structurilor de date care sunt accesate numai de procesor (de ex, citirea codului după ce s-a apăsat un buton la tastatură)

  - pot fi luate rapid, aşadar sunt executate de gestionarul de întreruperi imediat, cu întreruperile activate.

-măsuri non-critice întârziabile – exemplu - copierea conţinutului unui buffer în spaţiul de adrese al unui proces

  -pot fi întârziate într-un interval de timp lung fără a afecta operaţiunile kernel-ului;

procesul interesat va păstra pur şi simplu date.

În Windows Vista există un driver care specifică nivelul urgenţei întreruperii relativ la alte dispozitive din sistem. Acest nivel de urgenţă este ales conform „politicii" de prioritate a întreruperilor. De exemplu, dispozitivele care nu sunt compatibile Plug & Play necesită o prioritate mare. Aceste tipuri de dispozitive au numai de câştigat dintr-o politică de prioritate de întreruperi mare.
Pentru a specifica politica de prioritate de întreruperi pentru un dispozitiv, un driver trebuie să seteze următoarea valoare de registru în datele corespunzătoare respectivului dispozitiv (*bibliografie:* **Interrupt Arhitecture Enhancements in Microsoft Windows Vista- www.microsoft.com***):*

\Interrupt Management
\Affinity Policy
DevicePriority: REG_DWORD: *InterruptPriorityPolicyValue*

unde *InterruptPriorityPolicyValue* poate lua următoarele valori, indicând politica de întreruperi cerută:

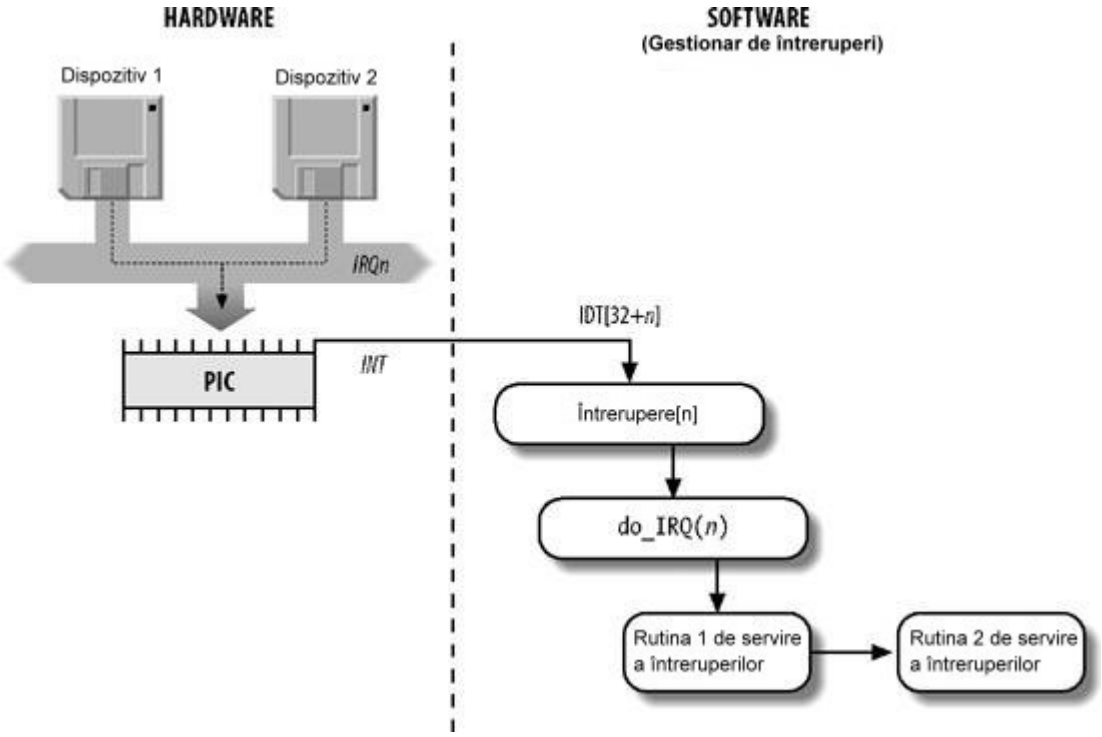
Constantă simbolică	Valoare	Definiţie
IrqArbPriorityUndefined	0x00	Indică faptul că un dispozitiv nu a încheiat o politică de prioritate. În Windows Vista, acest lucru este echivalent cu specificarea **IrqArbPriorityNormal**
IrqArbPriorityLow	0x01	Indică faptul că un dispozitiv poate tolera latenţe mai mari de întreruperi decât majoritatea; necesită un IRQL al dispozitivului mai mic decât al majorităţii dispozitivelor.
IrqArbPriorityNormal	0x02	Indică faptul că un dispozitiv poate tolera latenţe de întreruperi tipice. Aceasta este în mod *default* şi se potriveşte majorităţii

		dispozitivelor Windows.
IrqArbPriorityHigh	0x03	Indică faptul că dispozitivul necesită cea mai mică latenţă de întreruperi posibilă; necesită de asemenea un IRQL al dispozitivului mai mare decât al majorităţii device-urilor.

(bibliografie tabel: **Interrupt Arhitecture Enhancements in Microsoft Windows Vista***)*

Aici ar fi de precizat faptul că IrqlArbPriorityNormal, valoarea *default*, corespunde mai degrabă comportamentului sistemului în versiunile anterioare de Windows.

Oricare ar fi tipul de circuit care a cauzat întreruperea, toate gestionarele de întreruperi I/O iau următoarele măsuri de bază:
-salvează valoarea IRQ şi conţinutul registrelor din stiva Kernel Mode
-trimit o confirmare către PIC care serveşte linia IRQ
-execută rutinele de întreruperi (ISR) asociate cu toate dispozitivele care împart IRQ.
-termină prin saltul la adresa `ret_from_intr( )`



(Bibliografie figură: **Understanding the Linux Kernel, 3rd Edition + modificată***)*

Figura reprezintă printr-o metodă schematică circuitele hardware şi funcţiile software folosite pentru gestionarea unei întreruperi.

Boboc Octavian 442A

3. Device Drivers

Cuprinsul tau?

Te rog sa sistematizezi, să pui subtitluri şi să retransmiţi.

3.1 Definitie, exemple
3.2Clasificare; device drivere logice si fizice
3.3 Mecanisme de functionare
3.4 Exemple

3.1 Definitie, exemple

"device driver"-ul este programul ce intermediaza comunicarea unui program de nivel mai inalt cu parti fizice componente ale unui calculator

Driverele sunt folosite pentru a interfata 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, floppy disk) de diverse tipuri (ATA, SATA, SCSI)
- implementarea suportului pentru diverse sisteme de fisiere
- implementarea suportului pentru camere foto digitale si scannere

Un driver comunica in general cu un device prin sistemul intern de comunicatii al masinii la care este conectat acesta. Atunci cand un program solocita o procedura de la driverul respectiv acesta emite 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.

Un driver simplifica programarea actionand ca un translator intre componentele utilizate de un sistem si sistemul de operare sau aplicatiile ce le folosesc.

Orice versiune a unui dispozitiv, cum ar fi o imprimanta necesita comenzi specifice, insa majoritatea aplicatiilor acceseaza un astfel de dispozitiv (ex: trimiterea unui fisier catre imprimanta) folosind comenzi generice de nivel inalt ca "PRINTLN". Driverul este cel care primeste aceste expresii generice si le converteste in comenzi low-level pentru a fi acceptate de dispozitivul respectiv.

Driverele de I/O pot fi : -de sistem (standard)
-de aplicatie

Driverele de sistem (standard) respecta o anumita structura pe cind cele de aplicatii sunt pur si simplu o suma a functiilor pentru gestiunea perifericului. De asemenea se poate inlocui driverul standard de tastatura cu unul propriu etc.

Aici sunt dd pt MS-DOS?

MS-DOS permite inlocuirea sau adaugarea unor drivere urmatoarelor dispozitive standard de I/O :

CON: Console input –Keyboard
Console output -Monitor
AUX: auxiliary I/O (COM1, comunicatii)
PRN : standard printer output (LPT1)
NUL : dispozitiv fictiv

3.2Clasificare; device drivere logice si fizice

Device Drivers pot fi stratificate ca fiind de tip fizic sau logic. Cele de tip logic proceseaza date pentru o clasa de device-uri cum ar fi porturile ethernet in timp ce cele de tip fizic cu o componenta specifica. De exemplu: un port serial de comunicatii trebuie sa administreze protocoale standard de tip XON/XOFF comune oricarui device ce utilizeaza un astfel de port. Acest lucru este asigurat de stratul logic. Stratul logic trebuie insa sa comunice cu cipul portului serial. Acestea difera insa intre ele iar stratul fizic se adreseaza fiecarei variatii de chip in mod specific.AM spus la curs ca dd sunt cele de care te referi aici ca fizice, care devin logice. Cele logice sunt incluse manevrate direct de SO. Deci cele logice le manevrează pe cele fizice. Nu sunt pe acelaşi nivel ierarhic. Cele logice îndeplinesc rolul dispoozitivelor din

SO, cele fizice fac virtualizuarea fizice în logice, numai schimbă reprezentarea.

As dori ca aspectele particulare de la fiecare SO sa fie discutate după ce ai terminat problemele de principiu, aşa că ref la LINUX nu îşi are rostul aici, poate mai jos.

Pune titluiri, sistematizează şi retransmite.

Driverele Linux sunt integrate in nucleul sistemului de operare (kernel) si astfel adaptate largimii de banda necesare. De asemeni driverele pot constitui si parte separata de kernel ca module ce se pot incarca doar atunci cand este necesar, economisind memoria nucleului. Astfel de drivere sunt fisierele .sys in Windows si modulele .ko in Linux.

Fiecare dispozitiv de intrare/iesire atasat unui computer necesita un cod specific 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 a putea accesa registrii de control ai dispozitivului in cauza driverele sunt in mod normal inglobate in nucleul sistemului de operare. Acest lucru confera o performanta sporita dar in acelasi timp o fiabilitate precara deoarece un bug in orice astfel de driver poate afecta intregul sistem. MINIX 3 porneste de la acest model pentru a mari fiabilitatea. Astfel in MINIX3 fiecare driver este un proces separat pentru modul utilizator (user).

Majoritatea sistemelor de operare definesc o interfata standard pe care sa o suporte toate driverele de tip „block" si o alta interfata pentru driverele „character".Aceste interfete sunt formate dintr-o serie de proceduri pe care restul sistemului de operare sa le poata apela pentru ca driverul sa efectueze sarcinile pe care le primeste.

3.3 Mecanisme de functionare

In termeni generali se poate spune ca ocupatia ?? unui driver este sa accepte sarcini de la programe independente superioare si sa verifice ca acestea sunt efectuate. O astfel de cerere standard pentru driverul unui disc

este sa citeasca blocul n. Daca driverul se afla in stare de repaus la momentul respectiv atunci el trece la indeplinirea sarcinii imediat. Daca, insa, este ocupat cu o alta sarcina atunci va executa noile sarcini de indata ce este posibil in ordinea in care au fost emise.

Primul pas este in indeplinirea unei cereri I/O este verificarea ca parametrii de intrare sunt corecti, in caz contrar returnand 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 reprezinta 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 decida ce operatii sunt necesare si in ce ordine trebuiesc efectuate.

Odata ce driverul a determinat ce comenzi sa dea catre controller efectueaza acest lucru trecandu-le in registrii controllerului. Controlleri mai simpli pot efectua o singura comanda odata in timp ce altii mai sofisticati pot accepta o serie de comenzi legate pe care apoi le indeplinesc fara ajutorul sistemului de operare.

Dupa ce comanda sau comenzile au fost emise se disting doua situatii. In majoritatea cazurilor driverul trebuie sa astepte pana ce controllerul indeplineste anumite sarcini asa ca se blocheaza pana primeste semnalul de intrerupere care sa il deblocheze. In alte conditii insa operatia se efectueaza fara delay astfel incat driverul nu trebuie sa se blocheze. In oricare din cele doua 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.

Lucrul cu cereri de citire/scriere este activitatea principala a unui driver dar pot exista si alte functii. De exemplu driverul poate fi solicitat sa intializeze un device la pornirea sistemului sau la prima utilizare a acestuia. De asemeni poate fi nevoie ca acesta sa administreze probleme de alimentare, sau sa inregistreze loguri.

Driverele standard sint scrise pentru a pune la dispozitie servicii prin API.

Se pot scrie drivere care sa inglobeze mecanisme ca :

-software de comunicatii folosind intreruperile
-interfata ceas/calendar
-spooler de printare in background
-generator de numere aleatoare
-ram-disk
-handlere pentru protocoale de comunicatii

3.4 Exemple

Driverele standard incarcate de DOS implicit sint cele pentru ecran, tastatura,
imprimanta, disc si ceas.

Gestiunea unui periferic cuprinde scrierea unui driver continind :
-antet
-colectie de subrutine pentru functii specifice
-formarea unui sir al operatiilor ce urmeaza sa le efectueze driverul cu dispozitivul de I/O.

Elementele care trebuie puse in evidenta in scrierea unui driver sint:
-antetul driverului
-cererea pachet facuta de DOS
-pointerul local pentru cerere
-rutina strategie
-rutina intrerupere

Driverele standard se incarca in memoria calculatorului (se instaleaza la initializare - boot-time) prin CONFIG.SYS cu directiva DEVICE= . Driverele instalate in boot-time formeaza la rindul lor o coada sau sir sau lista inlantuita, antetul fiecarui driver continind un pointer DWORD (o adresa) la urmatorul. Ultimul driver din sir are un martor de sfirsit de sir cu toti bitii din antet pe -1.

Antetul driverului contine deci :
-pointer la urmatorul driver
-un atribut de driver
-o adresa a unei subrutine din interiorul driverului care se numeste cod de strategie
-o adresa a unei subrutine din interiorul driverului care se numeste cod de intrerupere
-numele driverului

Cele doua adrese (cod de strategie si intrerupere) sint prevazute pentru viitoare versiuni multitasking ale MS-DOS-ului, deoarece este evident ca operatiile de I/O cu dispozitvele periferice trebuie sa fie asincrone fata de cereri. Astfel subrutina cod de strategie va fi executata dind nastere la un sir al cererilor pentru operatiile de I/O cu dispozitivul periferic respectiv astfel ca driverul nu asteapta pina se efectueaza operatia respectiva dind controlul sistemului de operare. Este sarcina subrutinei cod de intrerupere, lansata imediat dupa de DOS, de a rezolva cererile din acest sir una cite una cind dispozitivul de I/O este pregatit. Cind o cerere este rezolvata subrutina cod de intrerupere semnalizeaza aceasta printr-un fanion de "Done". MS-DOS scaneaza periodic cererile lansate pentru dispozitivul respectiv si cind sesizeaza una din ele cu fanionul "Done" rezolva toata functia care solicita cererea respectiva. Deoarece aceste cereri apar aleatoriu este clar ca ele nu se pot transmite ca niste informatii in citeva registre sau variabile de memorie si trebuie transmise sub forma unei structuri standard de cerere numita cerere-pachet care este introdusa in sirul operatiilor ce urmeaza sa le efectueze driverul cu dispozitivul de I/O. Cererea pachet are doua parti :

-un antet al cererii care este acelasi pentru toate cererile (13 octeti)
-o zona specifica cererilor.

Cererea pachet este transmisa driverului prin ES:BX

La terminarea normala a unui driver, bitul "Done" este setat si se iese printr-un FAR RET. Bitul "Busy" este setat cind nu se poate satisface o cerere deoarece disp de I/O este ocupat cu satisfacerea unei alte cereri. Bitul mai foloseste pentru a returna informatii diverse ca : "coada tast nevida" sau "media este removable". Bitul "eroare prezenta" este setat cind un driver primeste o cerere pe care nu o poate rezolva sau se genereaza o eroare cind se executa o
cerere. Daca bitul este setat, bitii 0-7 (octetul de la offsetul 3 din cerere) se completeaza cu un cod de eroare.

Un driver standard este apelat in momentul in care sistemul de operare are introdus cel putin o cerere-pachet intr-un sir de operatii atasat unui dispozitiv de I/O. Este evident ca gestiunea sirurilor de cereri-pachet introduse si care sint "Done" o tine sistemul de operare. MS-DOS nu implementeaza mecanismul sirurilor. Cind apare o cerere se asteapta pina cind ea se rezolva. Codul de strategie retine adresa cererii-pachet la o anumita adresa intr-un sir si codul de intrerupere care se executa imediat dupa, rezolva cererea. Cind se

termina codul de intrerupere de executat MS-DOS stie ca cererea este "Done". Antetul driverului ocupa primii 18 octeti din corpul sau. Pentru aceasta trebuie ca fisierul executabil .COM sau .EXE care se obtine din sursa scrisa pentru driverul respectiv sa aiba originea zero (sau ORG 0 sau deloc, in nici un caz ORG 1OOh pentru a avea PSP-program segment prefix). Exista doua tipuri de drivere standard pentru dispozitive periferice :

-drivere caracter
-drivere bloc

Acestea difera intre ele prin informatiile ce le contin. Driverele caracter sint cele care transfera octet cu octet informatia cu un dispozitiv de I/O. Driverele bloc transfera o informatie egala cu un bloc de octeti, de exemplu discurile (FD,HDD) deoarece se poate transfera la nivelul unui sector (512 octeti). MS-DOS-ul intotdeauna proceseaza mai intii driverele definite in CONFIG.SYS inaintea celor deja incluse in el. Aceasta ne permite sa inlocuim de exemplu driverul MS-DOS-ului de tastatura sau imprimanta cu unul propriu; daca driverul nostru va avea numele CON sau PRN in zona de nume.

Deci :

1. MS-DOS pregateste cererea-pachet
2. MS-DOS lanseaza codul de strategie
3. MS-DOS lanseaza codul de intrerupere

Aceasta se preteaza in viitor foarte usor la un mecanism multitasking. Pointerul la urmatorul driver este format din patru octeti (offset + segment). Cind driverele se incarca in memoria calculatorului de pe HDD la boot-time adresele driverelor unde se gasesc in memorie formeaza un sir (o lista). Astfel fiecare antet de driver va contine in zona de pointer adresa driverului urmator daca exista sau -1(FFFF FFFF) daca este ultimul incarcat. Daca este necesara o operatie cu un periferic definit prin nume simbolic, MS-DOS cauta in acest sir pina cind numele simbolic se potriveste cu un nume din zona de nume al unui driver. Daca nu se gaseste apare mesajul "acces denied". Daca un singur fisier contine mai multe drivere, atunci zona de pointer a fiecaruia in faza de programare (scrierea sursei program) trebuie sa conţină adresa urmatorului driver din fisierul sursa respectiv, ultimul avind aceasta zona egala cu -1.

Bib şi ref in text

Coman Mihai 442A

4. Gestiunea I/O din SO(pentru perifericele logice)

Te rog sistematizează, pune subtitluri şi retransmite

Prezentare generala

Un concept important in design SO este cunoscut sub numele de "independenta dispozitivelor".Ceea ce vrea sa spuna acest concept este ca este este posibil sa se scrie un program care sa poata sa acceseze orice dispozitiv I/O fara sa fiu nevoit sa-i specific dispozitivul in avans.Un astfel de exemplu este acela al programului de citire al unui fisier ca intrare indiferent daca se gaseste pe floppy disk,CD,hard disk fara sa fiu nevoit sa modific programul pt fiecare dispozitiv in parte.De asemenea este posibil sa scriu urmatoarea comanda:

sort < input > output

si aceasta sa functioneze indiferent daca la intrare este vorba de floppy disk, IDD,SCSI,disk iar la iesire avem orice mediu de stocare or monitor.Aceasta problema intra in responsabilitatea SO,care trebuie sa rezolve problema cauzata de faptul ca aceste dispozitive sunt diferite si necesita comenzii diferite sa scrie si sa citeasca.

Foarte apropiat de acest concept este si scopul uniformizarii numelor.Numele unui fisier trebuie sa fie intreg si sa nu depinda in nici un fel de dispozitiv.In UNIX toate discurile pot fi incluse intr-un sistem de ierarhizare in moduri arbitrare astfel incat utilizatorul sa nu aiba nici o grija referitoare daca numele corespunde unui anumit dispozitiv.De exemplu floppy disk poate fi fixat in fata directoarelor /usr/ast/backup astfel daca copiez un fisier la /usr/ast/backup/Monday sa copiez fisierul pe floppy disk.In felul acesta toate fisierele sunt indexate in acelasi fel: dupa o cale précis definita.

O alta problema problema importanta pt S/O este buna organizare a erorilor.In general erorile trebuie sa fie cat mai bine organizate si apropiate de hardware.Daca controlerul descopera o eroare de citire,ar trebui sa incerce sa corecteze singur eroarea daca poate.Daca nu poate atunci driverul dispozitivului ar trebui sa incerce s-o repare,probabil prin incercarea de a citi din nou adresa de date.

Multe erori sunt trecatoare,un astfel de exemplu este eroarea de citire cauzata de praful de pe capul de citire,care va disparea daca operatia de citire se va repeta de cateva ori.Numai daca etajele inferioare nu pot sa solutioneze

problema,atunci trebuie sa se apeleze la cele superioare.De cele mai multe ori,recuperarea erorilor poate fi facuta transparent la un nivel scazut fara ca etajele superioare sa stie de eroarea respectiva.

O alta problema este blocarea versus intrerupere.Majoritatea dispozitivelor I/O sunt cu intrerupere - procesorul porneste transferul si inceteaza sa faca altceva pana cand intreruperea soseste.Programele utilizatorului sunt mai usor descris daca operatiile I/O sunt blocate - operatia de citire e suspendata pana cand datele sunt valabile in buffer.

Alta problema o reprezinta memorarea intermediara a datelor.De obicei datele care provin de la un dispozitiv nu pot fi stocate direct in locatia lor finala.De exemplu,cand se primeste un pachet de pe retea ,SO nu stie unde sa-l puna pana nu-l stocheaza undeva si il analizeaza.
Memorarea intermediara a datelor implica o serie considerabila de copieri , fapt ce are un impact major asupra performantelor I/O.

Un ultim concept prezentat este acela al dispezitivelor dedicate vs cele generalizate.Unele dispozitive ,cum ar fi discurile,pot fi utilizate de mai multi utilizatori in acelasi timp.Alte dispozitive cum ar fi discurile cu banda magnetica sunt dedicate unui singur utilizator.Introducerea dispozitivelor dedicate a adus si la aparitia unor probleme ,gen deadlocks.

Inca o data SO trebuie sa se descurce atat cu cele dedicate ,cat si cele generale fara sa apara erori.

Exista trei cai fundamentale diferite prin care I/O lucreaza:

-I/O programate;
-I/O intrerupte;
-I/O care foloseste DMA;

I/O programate

Cea mai simpla modalitate I/O este sa aiba un processor care sa faca toata treaba pt el;aceasta metoda se numeste I/O programate.
Cel mai simplu mod de a pune in evidentza este sa folosim un exemplu.Consideram ca utilizatorul doreste prin intermediul unui program sa printeze 8 caractere dintr-un sir "ABCDEFGH" cu ajutorul imprimantei.Prima data se pune sirul in memoria intermediara din spatial utilizatorului.Dupa aceea procesul utilizarorului face un apel catre imprimanta pt ca aceasta sa inceapa scrierea.Daca imprimanta este deja folosita de catre un alt proces,atunci aceasta incercare esueaza si se emite o eroarea ori se asteapta pana cand imprimanta este disponibila,aceasta decizie depinde de sistemul de operare si de parametri procesului .

De indata ce se elibereaza imprimanta,procesul utilizatorului face un apel catre sistemul de operare prin care ii cere sa inceapa printarea sirului de caractere cu ajutorul imprimantei.

Sistemul de operare atunci de obicei copiaza din memoria intermediara sirul intr-o zona,sa zicem p,in nucleu principal de unde se poate accesa cel mai usor.Atunci se verifica daca imprimanta este disponibila,daca nu se asteapta pana aceasta este.De indata ce se intampla asta,sistemul de operare copiaza primul caracter in registrul de date al imprimantei,in acest caz folosind memorie indexata I/O.Aceasta actiune activeaza imprimanta.Se poate ca ,caracterul sa nu apara deoarece unele imprimante asteapta sa se copieze in memoria intermediara mai multe caractere chiar si o pagina pana sa inceapa printarea.

De indata ce se copiaza primul caracter la imprimanta,sistemul de operare verifica daca imprimanta este pregatita sa primeasca alt caracter.In general imprimanta are un registru in care se specifica starea ei.Scrierea de date in registru face ca starea imprimantei sa fie ocupata.Cum controlerul imprimantei a scris caracterul current,atunci starea imprimantei se schimba prin trimiterea uni bit in registrul sau ori prin scrierea unei valori in el.

In acest punct sistemul de operare asteapta iar ca imprimanta sa fie disponibila.Acest procedeu continua pana cand se printeaza intreg sirul de caractere.
Aceste actiuni prezentate mai sus sunt sintetizate in urmatoarele linii de cod:

```
copy_from_user(buffer, p, count);          /* p is the kernel buffer */
for (i = 0; i < count; i++) {              /* loop on every character */
   while (*printer_status_reg != READY) ;  /* loop until ready */
   *printer_data_register = p[i];          /* output one character */
}
return_to_user();
```

I/O programate sunt simple dar au dezavantajul de a tine CPU ocupat pana cand se termina toate operatiile de I/O.Daca timpul alocat pt printarea unui caracter este scurt atunci asteptarea cat este ocupat este buna.De obicei asteptarea este mare de aceea este nevoie de o alta metoda I/O.

I/O intrerupte

Acum vom analiza cazul in care imprimanta nu copiaza caracterele in nici o memorie intermediara ci le printeaza asa cum vin.Daca imprimanta poate sa printeze sa zicem 100caractere/secunda atunci fiecarui character ii trebuie 10ms sa fie printat.Asta inseamna ca dupa fiecare scriere a unui caracter in registrul imprimantei, SPU intra in asteptare pt 10ms asteptand urmatorul caracter sa fie scos.Acest este un timp mai mult decat suficiet ca procesorul sa castige timp si pt alte procese care altfel s-ar fi risipit.

Felul prin care reusim sa facem procesorul sa execute si alte operatii pana imprimanta este pregatita pt a fi folosita este acela de a utiliza intreruperile.Cand sistemul apeleaza imprimanta sirul de caractere este deja facut,memoria intermediara este copiata in nucleul central,cum am aratat mai devreme si primul caracter este copiat la imprimanta de indata ce este disponibila aceasta.In acest punct procesorul executa alte procese.Procesul care se ocupa de printare este blocat pana cand nu ne printeaza intreg sirul.

Acest lucru este facut de urmatoarele instructiuni:

```
Copy_from_user(buffer, p, count);
enable_interrupts();
while(*printer_status_reg != READY) ;
*printer_data_register = p[0];
scheduler();
```

Cand imprimanta a printat un caracter si este pregatita sa-l accepte pe urmatorul este generata o intrerupere.Acesta intrerupere opreste aceast proces si ii salveaza stare lui.Atunci se ruleaza procedura de intrerupere la imprimanta care este prezentata intr-o varianta in liniile urmatoare :

```
if(count == 0) {
```

```
    unblock_user();
} else {
    *printer_data_register = p[i];
    count = count – 1;
    i = i + 1;
}
acknowledge_interrupt();
return_from_interrupt();
```

Folosirea DMA

Un aspect deranjant la intreruperea I/O e ca intreruperea intervine la fiecare caracter.Intreruperea ia timp ,fapt ce face ca acest lucru sa ocupe mult timp alocat CPU.O solutionare e sa folosirea DMA.Ideea la DMA consta in trimiterea de caracterele catre imprimanta fara ca procesorul sa fie deranjat.In esentza DMA este I/O programate,cu diferenta ca toata treaba este facuta de controlerul DMA nu de processor.Un exemplu de cod de prinatare a unui sir folosind DMA este urmatorul:

```
copy_from_user(buffer, p, count);
set_up_DMA_controller();
scheduler();
```

Marele castig al folosirii DMA este acela ca se reduce numarul intreruperilor de la unul la un caracter la unul pe memoria intermediara de printare.Daca avem multe caractere si intreruperile incite,acest lucru este o imbunatatire mare.Pe de alta parte ,controlerul DMA este mult mai lent decat procesorul.

Daca controlerul DMA nu este folosit la capacitatea lui maxima si processor nu are altceva de facut in timp intreruperii DMA,atunci I/O programate si I/O intrerupte pot fi mai bune.

Bibliografie: 1) Andrew S. Tanenbaum, **Sisteme de operare moderne**
LIXANDRU V Nicolae Petrut 442A

5.DISPOZITIVE HARWARE DE Intrare/Iesire

1.Prezentare generala
2. Principiile dispozitivelor harware de Intrare/iesire
3.Caracterizarea dispozitivelor
4. Intreruperi
5. Accesarea Directa a Memoriei (DMA)

1.Prezentare generala

Controlul dispozitivelor conectate la un calculator este un obiectiv major al oricarui sistem de operare. Subsistemul I/O al nucleului unui sistem de operare implementeaza o mare varietate de metode pentru a controla diferitele dispozitive atasate calculatoarelor din ziua de azi. Acest subsistem I/O ajuta la izolarea de restul nucleului, practic scutind Unitatea Centrala de Procesare (CPU) de complexitatea controlarii in detaliu a dispozitivelor de I/O .
Elementele harware de I/O de baza, ca porturile, magistralele si componentele electronice sau adaptoare (numite si device controllers in engleza) pot accepta o varietate mare de dispozitive de I/O. Nucleul sistemului de operare foloseste un concept de incapsulare (device drivers) pentru a implementa detaliile fiecarui dispozitiv in parte.
Dispozitivele de Intrare/Iesire ofera o interfata intre ele si subsistemul I/O, interfata uniforma definita pe “device driver's “. Prin aceasta interfata se face legatura practic cu nucleul subsistemului care face cererile catre aceasta interfata si de la care primeste datele de intrare.
Functiile ce trebuiesc indeplinite in controlul dispozitivelor sunt functii de comanda, functii de identificare a intreruperilor si functii de rezolvare a erorilor ce pot aparea pe parcurs.
Partea de cod pe care o reprezinta toate aceste functii de I/O reprezinta una importanta din totalul codului sistemului de operare, de aceea pentru a intelege intradevar functionarea unui sistem de operare trebuie sa intelegem si aceasta parte importanta a dispozitivelor I/O.

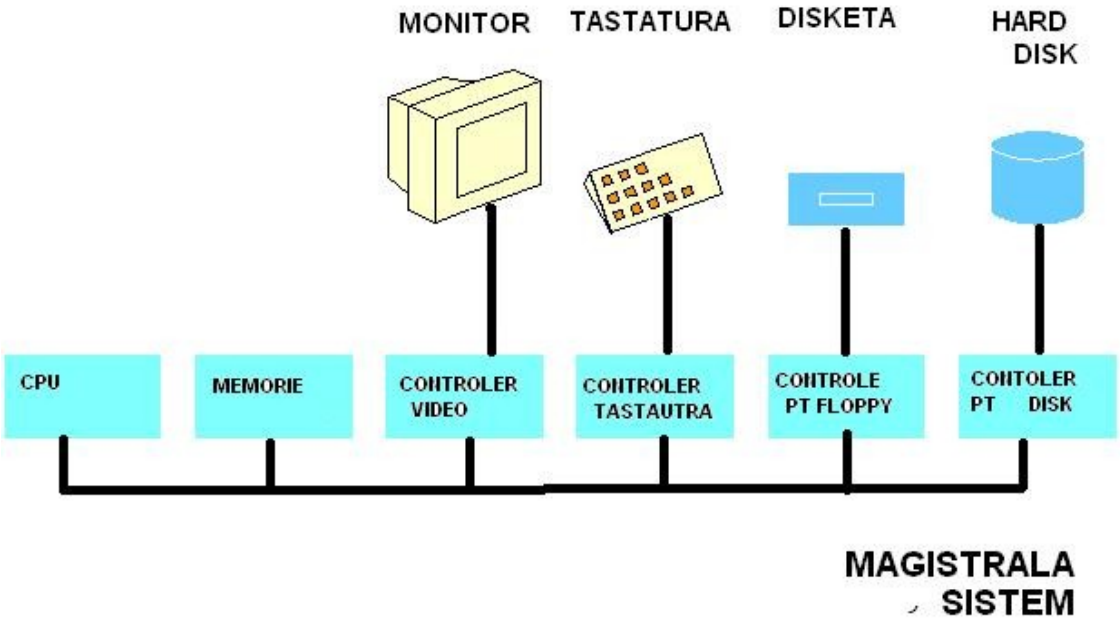
2.Principiile dispozitivelor harware de Intrare/iesire

Dispozitivele hardware pot fi vazute din diferite puncte de vedere. Electric, componentrele hardware sunt percepute ca cipuri, fire, surse de putere,

motoare; din punct de vedere al comandarii, prgramatorii privesc interfata software cu functiile pe care le intoarce la iesire, parametrii de intrare si erori ce apar.

3.Caracterizarea dispozitivelor

Dispozitivele I/O pot fi clasificate in doua categorii :
block devices - stocheaza informatia in blocuri de dimensiune fixa, fiecare cu adresa lui, facand astfel posibila citirea sau scrierea fiecarui bloc independent; hard disk-urile fac parte din aceasta categorie.
character devices - acestea livreaza sau accepta un sir de caractere fara a tine cont de vreo structura de bloc, nu este adresabila si nu are nici o operatie de cautare. Din aceasta categorie fac parte imprimantele, mouse-ii, etc.
Un calculator cu un sistem obisnuit(folosit pentru scopuri generale) consta dintr-un numar de controlere care sunt conectate print-o magistrala comuna, fiecare controler fiind responsabil de un anumit tip de dispozitiv.



Schema preluata din Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004;

Unitatile de I/O sunt in mod normal compuse dintr-o parte mecanica si una electronica. Este foarte des posibil sa separan cele doua parti pentru a realiza un design modular mai general.
Componenta electronica este numita adaptor(device controller) si reprezinta o placa de circuit ce poate fi introdusa intr-un slot de exapnsiune.

Componenta mecanica este de fapt dispozitivul in sine. Din connector sunt iesiri spre dipozitive, dipozitive ce pot fi un monitor, un mouse sau orice alt dispozitiv perifiric.
Sistemul de operare aproape intotdeauna lucreaza cu controller ul si nu cu device ul.
Rolul convertorului este de a transforma sirul de biti primiti de la device ca si comanda intr-un block de biti, de a realiza corectarea erorilor, si apoi de a trimite mai departe in blocurile de biti sa fie copiate in memorie.
Calculatoarele cele mai moderne sunt centrate in jurul magistralei PCI (P Peripheral Component Interconnect) si folosesc mai multe magistrale.
Controlerele de generatie mai noua contin si registre de memorie ce sunt folosite pentru a comunica cu CPU. Facand scrierea in aceste registre sistemul de operare poate comanda dispozitivul sa livreze date, sa accepte date, sa se porneasca sau sa se stinga, sau orice alta actiune.
Mai mult pe langa registrele de control, multe dispozitive au si un buffer de date pe care sistemul de operare poate sa l citeasca sau sa l scrie.

4.Intreruperi

Registrele controlololelor au unul sau mai multi biti de status ce pot fi testati pentru a determina daca o operatie de iesire este completa sau daca date noi sunt disponibile de la un dispoazitiv de intrare. O unitate CPU poate executa un salt testand un bit de status de fiecare data pana cand un dispozitiv este pregatit sa accepte sau sa puna la dispozitie date noi.

Intreruperi – datorate I/O
Procesorul fae o comanda I/O si continua sa o execute; moldulul I/O intrerupe procesorul cand a terminat procesul de I/O; procesorul initiat poate fii suspendat asteptand intreruperea.

5.Accesarea Directa a Memoriei (DMA)

Un modul DMA ontroleaza schimbul de date intre modulele de I/O si memoria principala;procesorul face o cerere de transfer a unui bloc de date de la modulul DMA si apoi este intrupt numai dupa intregul bloc a fost transferat

Bibliografie

John Levine - "Linkers and Loaders" (http://www.iecc.com/linker)
Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004;
www.wikiepedia.com.

La fel bib si ref in text, doar nu esti autorul tuturor info prezentate.

IONESCU D S Catalin 442A

6. APELURI DE SISTEM INTRARE/IESIRE IN LINUX

Ca la ceilalti, sumar propriu si sistematizare mai bună, plan de prezentare, bibliografie cu ref in text

6.1 Ce este linux-ul?
6.2 Apeluri de sistem (system calls)
6.3 Apeluri de sistem de intrare/iesire (in linux)
6.3.1 Apeluri pentru i/o sincrone(aplicatie linux;exemple)
6.3.2 Apeluri pentru i/o asincrone(aplicatie linux;exemple)

6.1 Ce este linux-ul?

Linux – ul este un sistem de operare care respecta structura si comportarile Unix – ului (unix-like) initiat si coordonat de Linus Torvalds cu ajutorul comunitatii open-source.([1])

Pai el numai coordona, de facut l-au facu şi îl fac alţii. Ca şi cum NC a făcut casa poporului.

Se va considere studiu de caz pentru acest sistem de operare.

6.2 Apeluri de sistem (system calls)

Sistemele de operare ofera proceselor care ruleaza in spatiul utilizator (user mode) un set interfete prin care sa comunice cu procesorul, hard – disk ul, imprimanta si altele. Acest nivel suplimentar are un set de avantaje : face treaba programatorului mai usoara ferindu-l de limbaje primitive (asamblare) si caracteristici hardware, sporeste securitatea sistemului (adica de exemplu se resping cererile care duc la erori sau violari de adrese) si asigura o portabilitate a programelor (de exemplu daca se schimba o parte hardware atunci programul poate rula pe noul sistem pentru ca sistemul de operare are acelasi interfata cu programul).([2])

Linux-ul implementeaza astfel de interfete intre procesele utilizatorilor si hardware sub forma unor apeluri de sistem si in mare parte este compatibil **POSIX**. Posix-ul ("Portable Operating System Interface")reprezinta un standard pentru a defini **API**(application programming interface) compatibile cu diverse versiuni de unix.([1])

pai mai sus ai spus ca este compatibil POSIX ca std API. Explica ce intelegi prin compatibilitate AS / API, ca mai apoi sa se înţeleagă deosebirile

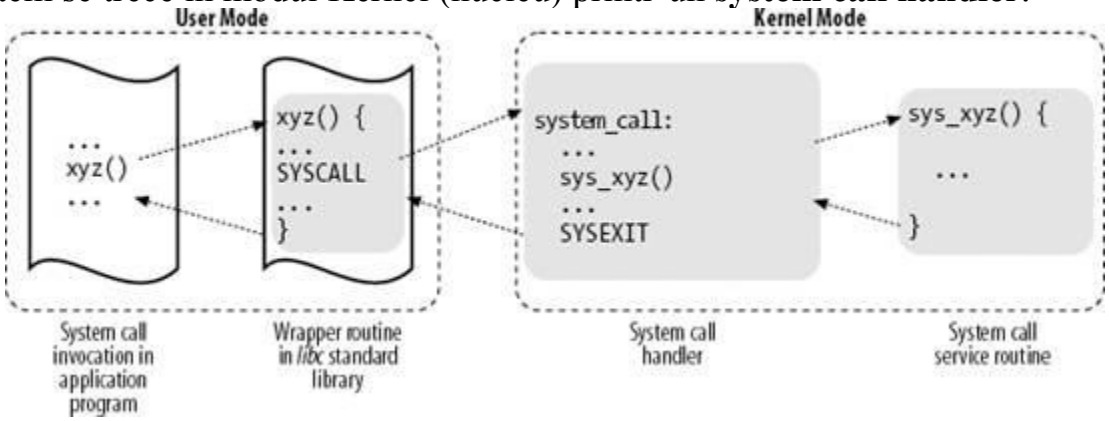
Standardul POSIX se refera la API si nu la apeluri de sistem. Un sistem poate fi certificat POSIX daca ofera un set de api pentru programe aplicatii, neinteresand modul in care sunt implementate.

Programatorul nu face deosebiri intre apel de sistem sau API, pentru ca el urmareste doar numele functiei, parametrii si rezultatul functiei dorite. Pentru cel care construieste kernele sunt doua notiuni diferite pentru ca apelurile de sistem apartin kernel-ului pe cand API-urile sunt in librarii din spatiul utilizatorului.

Un apel se deosebeste de un API pentru ca este un mod explicit de a cere kernel–ului ceva printr-o intrerupere software pe cand cel de al doilea este o functie care specifica modul de a obtine un serviciu anume.

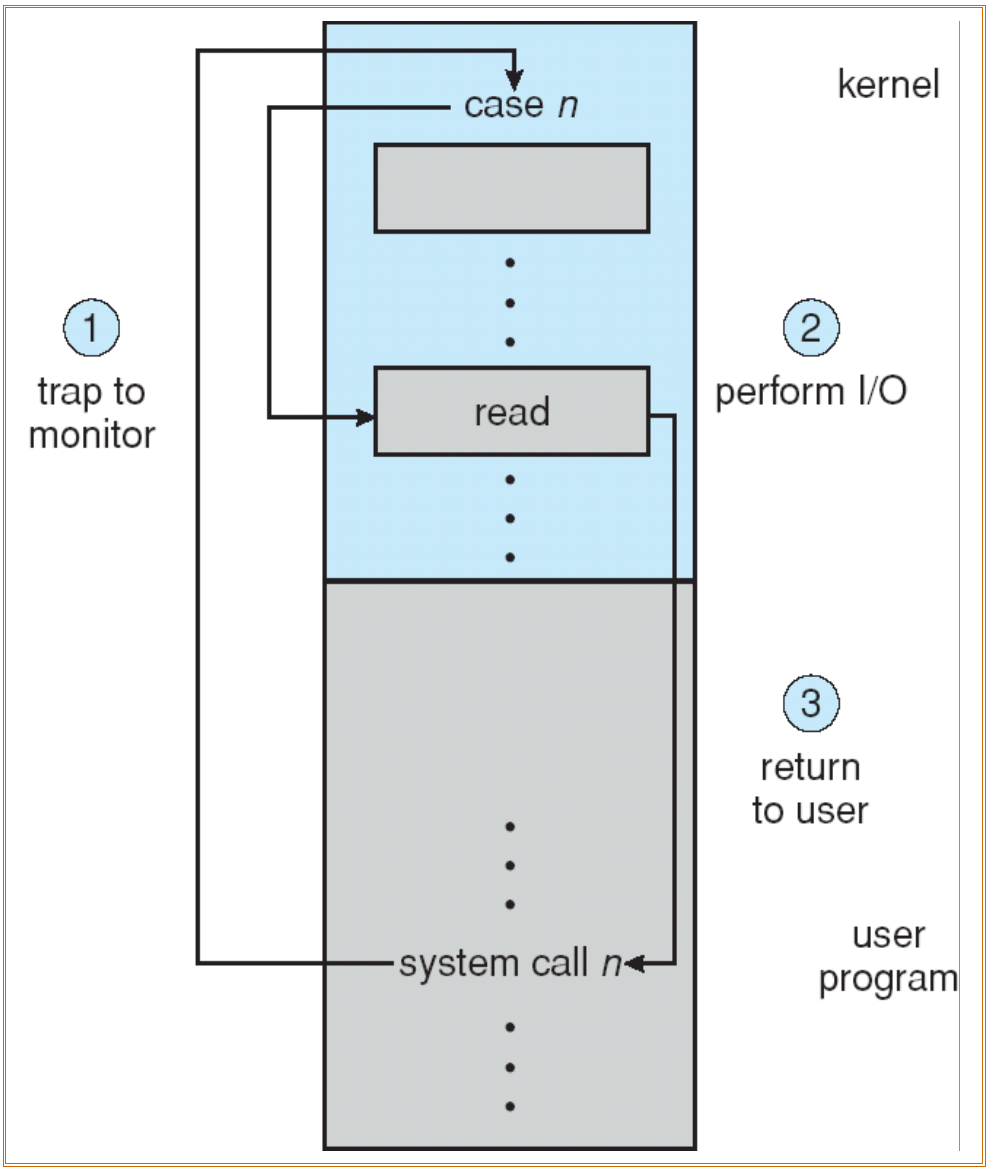
Sistemele unix include cateva librarii de functii API pentru programatori. Cateva dintre API-urile din *libc* , libraria C standard se refera la **rutine wrapper** (rutine care au ca singur scop un apel de sistem). De obicei fiecare apel de sistem are o rutina API echivalenta(**wrapper routine**).

Cand un program din spatiul utilizatorului face o cerere de tip apel de sistem se trece in modul Kernel (nucleu) printr-un **system call handler**.



Din figura se observa distinctia dintre spatiul kernel **(kernel mode)** si spatiul utilizator(**user mode**);in **user mode** un program care doreste sa faca un apel de sistem mai intai executa o fuctie API care contine rutina respectiva ; apoi se trece in **kernel mode** si se executa apelul de sistem in cauza.

Toate apelurile de sistem returneaza un tip de data integer care daca este pozitiv sau 0 spune ca s-a executat cu succes, iar daca este negativ reprezinta un cod de eroare.Parametrii necesari apelului sunt transmisi via registri specializati.([2])



de unde? Sau figura e făcută de tine? (figura este dintr-un curs in format ppt **Operating systems concepts – Silberschatz,Galvin and Gagne**)

6.3 Apeluri de sistem de intrare/iesire (in linux)

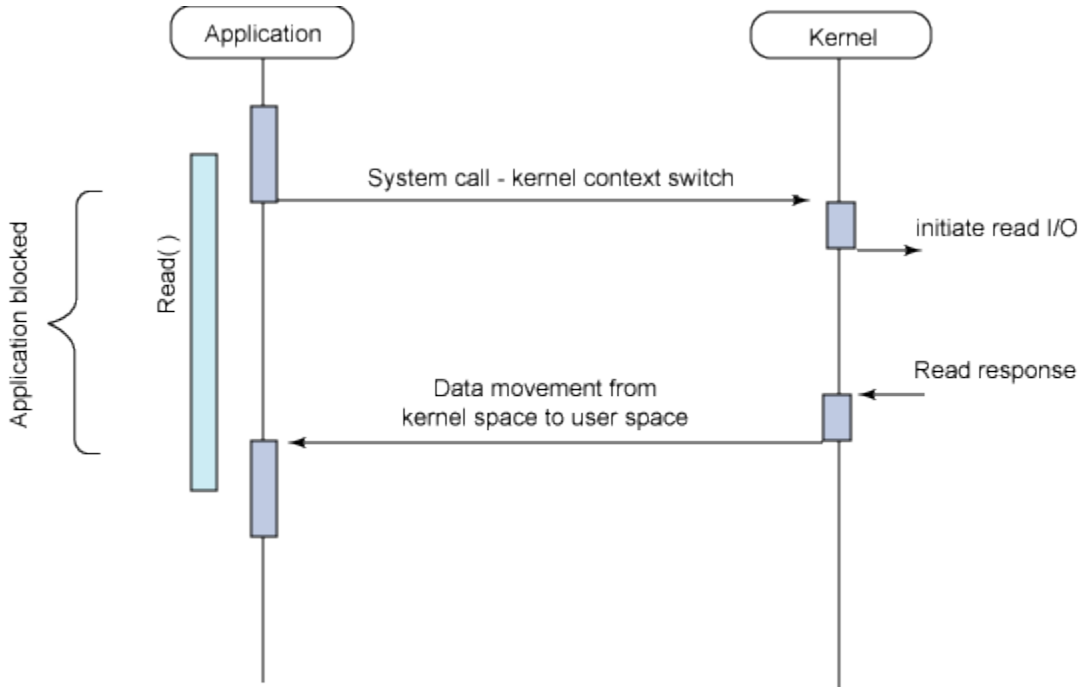
Sunt 2 tipuri de intrari/iesiri:

a)sincrone – fiecare cerere de i/o este astepta sa isi termine treaba cu descriptorul de fisier pentru ca urmatoarea cerere sa fie acceptata si programul asteapta terminarea cererii pentru a trece la urmatoarea instructiune

b)asincrone – cerera este gestionata de sistemul de operare si programul poate sa ii execute urmatoare instructiune

6.3.2 Apeluri pentru i/o sincrone(aplicatie linux;exemple)

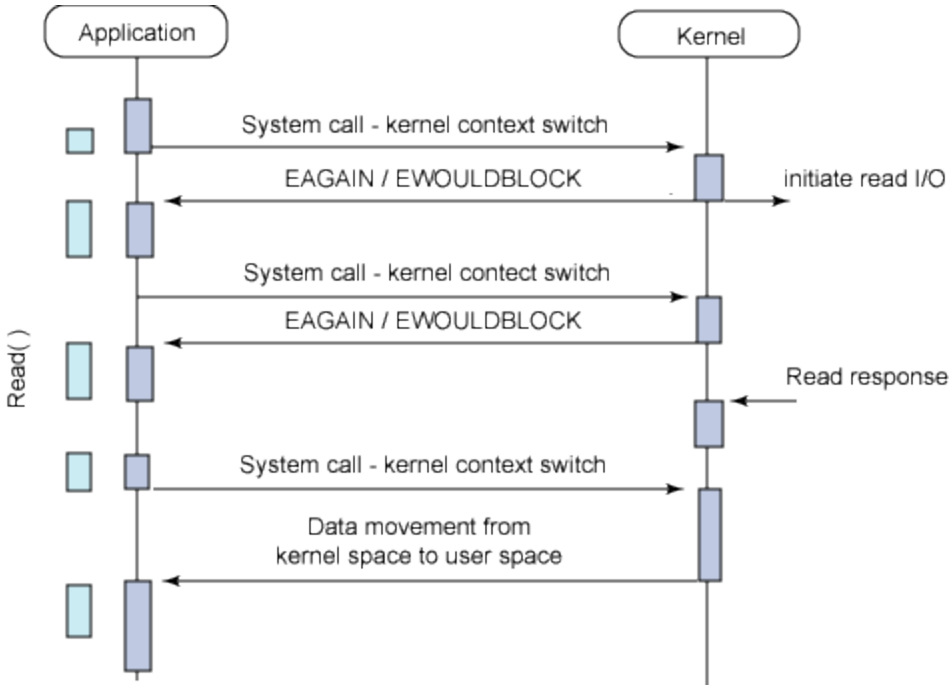
In schema urmatoare ase prezinta un apel de i/o sincron efectuat de catre o aplicatie kernel –ului:



([4])

? fig cu nume engl de unde

Se poate insa efectua o i/o sincrona neblocata astfel: nu se mai termina operatia de intrare/iesire si se returneaza un cod de eroare.



([4])

Exista 5 apeluri principale de sistem pentru intrare/iesire pe care le ofera linux (acestea sunt pentru i/o sincrone):

1. int open(char *path, int flags [, int mode]);
2. int close(int fd);
3. int read(int fd, char *buf, int size);
4. int write(int fd, char *buf, int size);
5. off_t lseek(int fd, off_t offset, int whence);

Se observa faptul ca arata la fel ca niste apelari de proceduri, pentru ca in acest fel se pot programa cu ele.De fapt apelurile sunt diferite , ele dupa cum am aratat mai sus fac apeluri la sistemul de operare.

1.**Open** face o cerere sistemului de operare pentru a citi un fisier.Argumentul ‘path’ specifica calea(absoluta sau cea curenta) si numele fisierului, iar ‘flags’ si ‘mode’ specifica modul de deschidere. Daca sistemul de operare e de acord returneaza descriptorul fisierului vizat (un intreg nenegativ).Daca returneaza -1 atunci accesul este interzis.

Exemple:

```
#include <fcntl.h>
main()
{
  int fd;
```

```
  fd = open("in1", O_RDONLY);
  printf("%d\n", fd);
}

#include <fcntl.h>
main()
{
  int fd;

  fd = open("out1", O_WRONLY);
  if (fd < 0) {
    perror("out1");
    exit(1);
  }
}


#include <fcntl.h>
main()
{
  int fd;

  fd = open("out2", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0) {
    perror("o3");
    exit(1);
  }
}
```

Consola:

```
UNIX> o2
o2: No such file or directory      //pentru ca nu este cerut sa
ase //creeze un fisier nou
UNIX> o3
UNIX> ls -l out*
-rw-r--r--  1 plank           0 Sep 11 08:50 out2
```

2.**Close** spune sistemului de operare ca s-a terminat lucrul cu acel fisier pentru al folosi la altceva.

Exemplu:

3.**Read** impune citirea unui numar de bytes din fisierul deschis cu descriptorul de fisier ‘fd’ si copierea acestora in locatia ‘buf’ si returneaza cati bytes s-au citit concret

Exemplu:

```
#include <fcntl.h>
main()
{
```

```
  char *c;
  int fd, sz;

  c = (char *) calloc(100, sizeof(char));

  fd = open("in1", O_RDONLY);
  if (fd < 0) { perror("r1"); exit(1); }

  sz = read(fd, c, 10);
  printf("apelarea read(%d, c, 10).  a returnat %d byti  cititi.\n",
         fd, sz);
  c[sz] = '\0';
  printf("acestia sunt: %s\n", c);

  sz = read(fd, c, 99);
  printf("apelarea read(%d, c, 99).  A returnat %d byti  cititi.\n",
         fd, sz);
  c[sz] = '\0';
  printf("acestia sunt: %s\n", c);

  close(fd);
}
```

Consola:

```
UNIX> cat in1
Jim Plank
Claxton 221
UNIX> r1
apelarea read(3, c, 10).  A returnat 10 byti cititi.
acestia sunt: Jim Plank

apelarea read(3, c, 99).  A returnat 10 byti cititi.
acestia sunt: Claxton 221
```

4.**WRITE** se comporta exact ca ‘read’ numai ca scrie un set de bytes in loc sa ii citeasca.

Exemplu:

```
#include <fcntl.h>
main()
{
  int fd, sz;

  fd = open("out3", O_WRONLY | O_CREAT | O_TRUNC, 0644);
  if (fd < 0) { perror("er1"); exit(1); }

  sz = write(fd, "cs360\n", strlen("cs360\n"));

  printf("apelarea write(%d, \"cs360\\n\", %d).  a returnat %d\n",
         fd, strlen("cs360\n"), sz);
```

```
  close(fd);
}
```

Consola:

```
UNIX> w1
apelarea write(3, "cs360\n", 6).  a returnat 6
UNIX> cat out3
cs360
UNIX>
```

5.**LSEEK** muta pointerul pe o anumita pozitie in fisier.Toate fisierele au asociate un pointer ‘file pointer’ care reprezinta pozitia curenta.De exemplu cand este deschis un fisier file pointer-ul este pozitionat la inceputul fisierului iar cand se citeste/scrie in fisier pointerul isi modifica pozitia.’Whence’ specifica modul de reper, adica de la inceputul fisierlui, de la sfarsit sau de la pozitia curenta.([4])

Exemplu:

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

main()
{
  char *c;
  int fd, sz, i;

  c = (char *) calloc(100, sizeof(char));
  fd = open("in1", O_RDONLY);
  if (fd < 0) { perror("r1"); exit(1); }

  sz = read(fd, c, 10);
  printf("am deschis in1 si am apelat read(%d, c, 10).\n", fd);
  printf("a returnat ca %d bytes  au fost cititi.\n", sz);
  c[sz] = '\0';
  printf("acestia sunt: %s\n", c);

  i = lseek(fd, 0, SEEK_CUR);
  printf("lseek(%d, 0, SEEK_CUR) returneaza  offset-ul curent  a
fisierului %d\n\n", fd, i);

  printf("acum cautam inceputul fisierului si apelam read(%d, c, 10)\n",
         fd);
  lseek(fd, 0, SEEK_SET);
  sz = read(fd, c, 10);
  c[sz] = '\0';
  printf("functia read returneaza: %s\n", c);

  printf("acum apelam lseek(%d, -6, SEEK_END) care returneaza %d\n",
  fd, lseek(fd, -6, SEEK_END));
```

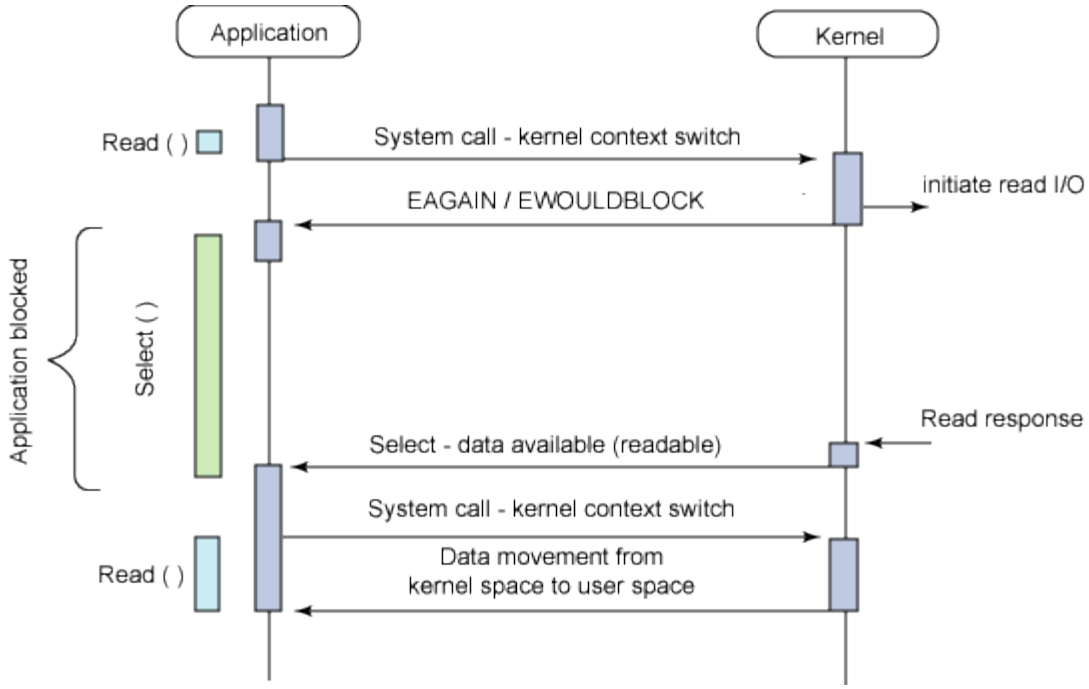
```
  printf("daca apelam read(%d, c, 10), obtinem: ", fd);

  sz = read(fd, c, 10);
  c[sz] = '\0';
  printf("%s\n", c);

  printf("iar daca se apelaeaza lseek(%d, -1, SEEK_SET).  Returneaza
-1.\n", fd);
  printf("perror() ne spunde de ce s-a intamplat eroarea: ");
  fflush(stdout);

  i = lseek(fd, -1, SEEK_SET);
  perror("er2");
}
```

([3])

6.3.1 Apeluri pentru i/o asincrone(aplicatie linux;exemple)

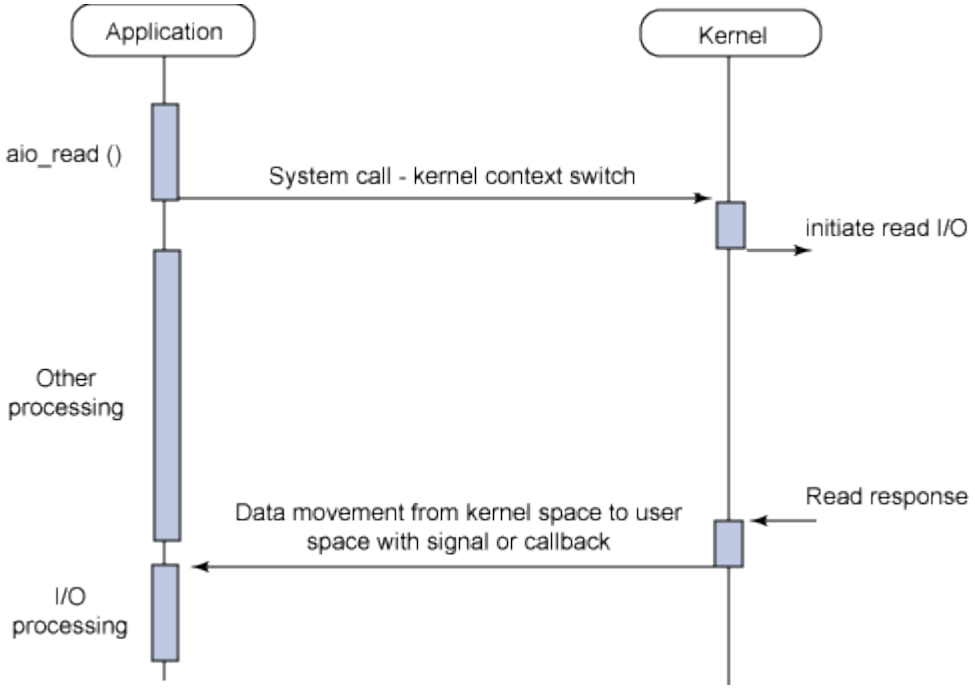
I/O asincrone sunt relativ noi in linux. Ideea esentiala este sa sporeasca productivitatea sistemului, adica avand un numar de dispozitive de i/o kernel-ul sa nu fie nevoit sa blocheze accesul un unui proces la dispozitiv sau sa il faca sa astepte.

Orice dispozitiv I/O asincron poate fi insa si blocat daca este configurat cu ajutorul apelului de sistem ‘select’ care este folosit care este folosit pentru a vedea daca este activitate pentru descriptorul dispozitivului de i/o.



([4])

In schema urmatoare este modelul asincron propriu-zis, neblocat, care returneaza imediat daca operatia se poate efectua cu succes si lasa programul sa isi execute urmatoarele instructiuni :



([4])

I/O asincrone au o API mai simpla pentru a fi mai transparente pentru programatori si in mare parte fac acelasi lucru ca si cele de mai sus pentru I/O sincrone :

1. aio_read
2. aio_error - ofera statusul cererii
3. aio_return - statusul operatiei complete de i/o
4. aio_write
5. aio_suspend - suspenda cererile pana cand una sau mai multe cereri sunt rezolvate (sau interzise)
6. aio_cancel - interzice cererea
7. lio_listio - initiaza lista de cereri

Aceste functii folosesc structura

```
struct aiocb {

  int aio_fildes;               // descriptor de fisier
```

```
  int aio_lio_opcode;           // valid doar pentru  lio_listio //
(r/w/nop)
  volatile void *aio_buf;       // buffer(tampon) de date
  size_t aio_nbytes;            // numarul de biti din buffer
  struct sigevent aio_sigevent; // structura pentru notificare

  /* campuri interne */
  ...

};
```

Exemplu:

```
#include <aio.h>

...

  int fd, ret;
  struct aiocb my_aiocb;

  fd = open( "file.txt", O_RDONLY );
  if (fd < 0) perror("open");

  /* se face pune 0 in structura aiocb (recomandat) */
  bzero( (char *)&my_aiocb, sizeof(struct aiocb) );

  /* alocarea de memorie buffer-ului  pentru cerinta aiocb  */
  my_aiocb.aio_buf = malloc(BUFSIZE+1);
  if (!my_aiocb.aio_buf) perror("malloc");

  /* initializarea campurilor necesare in structura aiocb */
  my_aiocb.aio_fildes = fd;
  my_aiocb.aio_nbytes = BUFSIZE;
  my_aiocb.aio_offset = 0;

  ret = aio_read( &my_aiocb );
  if (ret < 0) perror("aio_read");

  while ( aio_error( &my_aiocb ) == EINPROGRESS ) ;

  if ((ret = aio_return( &my_iocb )) > 0) {
    /* in ret se afla  bytii cititi */
  } else {
    /* citirea a esuat, consultam errno */
  }
```

([4])

Ancuta Razvan 442A

Bibliografie

1.www.wikipedia.com

2.Understanding the Linux Kernel, 3rd Edition , By Daniel P. Bovet, Marco Cesati

3.CS360 Lecture notes -- Introduction to System Calls (I/O System Calls) by Jim Plank
4.Operating Systems System Calls and I/O by Henry Newman

7. Apeluri API pentru I/O sub Windows

Iarăşi, sistematoizare cu subtitluri

Aici subcapAPI/NT

SUBCAPITOLE :
1. API/NT
2. Win32/Linux
3. Tipuri de apeluri API
4. Apeluri API pentru Intrari / Iesiri in Windows – prezentare
5. Cateva categorii de apeluri Win32 API

Subcapitol 1. API/NT

Aplicatiile Windows folosesc functii Win32 si comunica cu subsistemul Win32 ca sa faca apelari de sistem. Subsistemul Win32 accepta apelari ale functiilor Win32 si foloseste modulul librariilor interfetei sistemului (de fapt niste fisiere DLL) pentru a gestiona iesirile adevaratelor apelari ale sistemului NT.

Interfata NT-ului este principala conexiune a programatorului la sistem. Din pacate, Microsoft n-a facut publica lista completa de apelari de sistem ale NT-ului si le mai si schimba la fiecare imbunatatire a versiunii. In asemenea conditii, scrierea de programe care fac apelari directe e aproape imposibila.

Astfel ceea ce Microsoft a facut este sa creeze un set de apelari numit Win32 API (Application programming interface) care sunt public cunoscute. Acestea sunt fie proceduri de librarii care fac apelari de sistem ca sa finalizeze comenzile, fie, in unele cazuri isi fac comenzile direct in spatiul de lucru al procedurilor librariilor sau in subsistemul Win32. Apelarile Win32 API nu se schimba la fiecare versiune noua pentru a imbunatati stabilitatea. Totusi exista apelari NT API care pot schimba in noua versiune. Desi apelari NT API nu sunt toate apelarile de sistem ale NT-ului, sunt mai usor de folosit decat adevaratele apelari de sistem ale NT-ului pentru ca sunt mai bine documentate si mai stabile in timp.

Subcapitol 2. Win32/Linux

Filozofia Win32 este total diferita de cea de la UNIX. La UNIX apelarile de sistem sunt public cunoscute si formeaza o interfata minimala : renuntand la una din ele se reduce functionalitatea sistemului de operare.

Filozofia Win32 este de crea o interfata complexa, care ofera de cele mai multe ori 3 sau 4 cai de a face acelasi lucru, si care include multe functii care e clar ca nu ar trebui sa fie apelari de sistem. Un astfel de exemplu este apelarea API care copiaza un intreg fisier.

Multe apelari Win32 API creaza obiecte kernel de un anumit tip, incluzand fisiere, procese, pipes, threads. Fiecare apel care creaza un obiect kernel returneaza un rezultat numit handle. Acesta poate fi folosit pentru a modifica obiectul si este specific procesului care a creat obiectul . Ele nu pot fi transmise direct unui alt proces si folosite acolo(la fel cum fisierele descriptive UNIX nu pot fi transmise unui alt proces si folosite acolo).

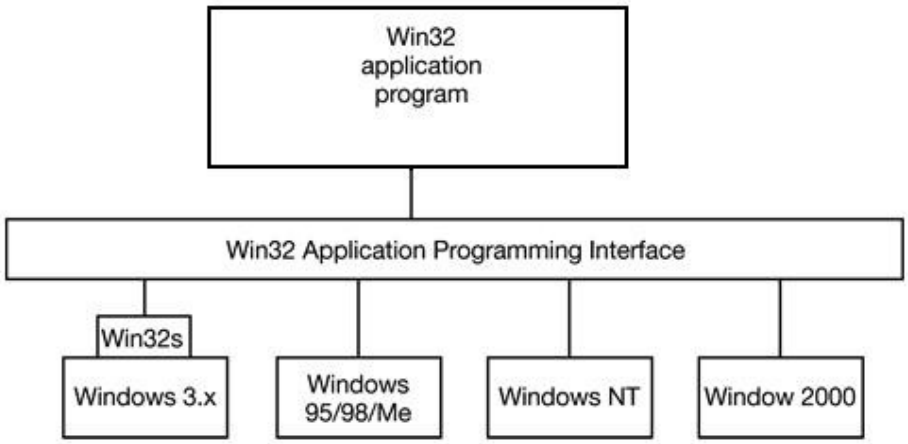
Totusi in anumite circumstante, este posibil sa dublezi un handle si sa-l treci unui alt proces intr-un mod protejat, permitandu-le acces la obiecte care apartin altui proces. Fiecare obiect are un descriptor de securitate asociat lui, care ii spune cine poate si cine nu poate face anumite operatii asupra obiectului.

Subcapitol 3. Tipuri de apeluri API

Despre NT se spune ca este orientat pe obiecte pentru ca singura cale de a manipula obiecte kernel esta prin apeluri de metode (functii API) asupra handle-urilor. Pe de alta parte ii lipsesc cele mai elementare proprietati ale sistemelor orientate pe obiecte. (mostenirea si polimorfismul).
Win32 API este disponibil si pe Windows 95/98 cu anumite exceptii. De exemplu, Windows 95/98 nu are securitate, asa ca apelurile API care au legatura cu securitatea returneaza mesaje de eroare. De asemenea , fisierele NT folosesc caractere Unicode, care nu sunt disponibile sub Windows 95/98. De asemenea mai sunt cateva diferente in parametrii unor functii de apel API. De exemplu, sub NT toate coordonatele date in functiile grafice sunt numere pe 32 de biti; sub Windows 95/98 sunt folosite numere pe 16 biti. Existenta Win32 API pe diferite sisteme de operare face mai usoara trecerea programelor intre ele dar si arata mai clar ca intr-un fel sunt decuplate de la apelurile de sistem actuale.

Programele binare pentru Intel x86 care apeleaza la interfata Win32 API o sa ruleze nemodificat pe toate versiunile de Windows de la Windows 95 pana la Vista. Asa cum se poate vedea in diagrama de mai jos este necesara o noua librarie atasata lui Windows 3.x pentru a crea un set de apeluri pe 32 de biti API pe un sistem de operare pe 16 biti, in timp ce pentru celelalte Windows-uri nu e necesara nici o adaptare. Trebuie precizat ca Windows 2000 aduce noi functionalitati Win32-ului, avand apeluri API neincluse in

versiuni mai vechi ale Win32 si care nu vor functiona pe versiuni mai vechi de Windows.



Win32 API permite programelor sa ruleze pe aproape toate versiunile de Windows

Apelurile Win32 API acopera o mare parte din zonele sistemelor de operare cu care ar putea interactiona si o parte si cu care nu ar fi trebuit sa interactioneze.Exista apeluri pentru crearea si managementul proceselor si threads-urilor. De asemenea mai sunt si apeluri legate de comunicatia intre procese.

Desi o mare parte din sistemul de managementn al memoriei este invizibil programatorului , o componeta esentiala este vizibila : si anume posibilitatea procesului de a localiza un fisier intr-o anumita regiune a memoriei sale virtuale. Astfel i se permite procesului sa citeasca si sa scrie parti din fisier ca si cum mar fi cuvinte de memorie.

O zona importanta pentru multe programe sunt intrarile si iesirile fisierelor. In viziunea Win32, fisierul este o secventa liniara de biti. Are peste 60 de apeluri pentru crearea si distrugerea fisierelor si directoarelor, deschiderea si inchiderea fisierelor, scrierea si citirea lor, cererea si setarea atributelor si multe altele.

O alta zona pentru care Win32 permite apeluri este zona de securitate. Fiecare proces are un ID care ii spune cine il foloseste si fiecare obiect poate avea o lista de control al accesului care spune precis ce utilizatori pot avea acces, dar si ce operatii sunt permise asupra lor. Astfel se pun bazele

securitatii, anumiti utilizatori putand sa aiba acces sau nu asupra fiecarui obiect.

Apelurile pentru procese, sincronizare, managementul memoriei, intrarile si iesirile fisierelor si pentru securitatea sistemului sunt normale. Si alte sisteme de operare le au, desi nu la fel de multe ca Win32. Totusi ceea ce distinge Win32 de concurenta sunt miile de apeluri pentru interfata grafica. Sunt apeluri pentru crearea, distrugerea, managementul si utilizarea ferestrelor, meniurilor, barelor de stare, icoanelor si a multor iteme care apar pe ecran. De asemenea sunt apeluri pentru desenarea figurilor geometrice, “umplerea” lor, modificarea culorilor, fontului, pentru lucrul cu tastatura, mouse-ul si alte dispozitive de intrare, precum si sisteme audio, imprimanta si alte dispozitive de iesire.

Windows trebuie sa aiba informatii despre hardware, software si utilizatori. In Windows 3.x informatia era stocata in sute de fisiere .ini localizate in tot hard-ul. De la Windows 95 informatia necesara pentru bootare si configurare de sistem si pentru atribuirea ei unui utilizator este stocata intr-o baza de date centrala numita registri.

Registrii sunt disponibili in totalitate programatorului Win32. Exista apeluri care sa creeze si sa stearga key-uri (directoare), sa caute valori in interiorul lor si multe altele. Unele din cele mai folositoare sunt :

Functie Win 32 API	Descriere
RegCreateKeyEx	Creaza o noua registry key
RegDeleteKey	Sterge un key de registru
RegOpenKeyEx	Deschide un key ca sa ia un handle catre el
RegEnumKeyEx	Enumereaza subcheile (subkeys) subordonate key of the handle
RegQueryValueEx	Verifica datele pentru o valoare din key

Subcapitol 4. Apeluri API pentru Intrari / Iesiri in Windows

Scopul I/O in Windows este de a asigura un spatiu de lucru (framework) pentru a manevra eficient un numar mare de dispozitive de intrare / iesire. Dintre dispozitivele de intrare amintim : tastaturi, mouse, touch pads, joystick-uri, scanere, camere foto, camere video,cititoare de cod de bare, microfoane. Dintre dispozitivele de iesire amintim monitoare, imprimante, plotere, DVD-Writere si placi de sunet. Dintre dispozitivele de stocare de date amintim : floppy disk-uri, hard-uri IDE si SCSI, CD-ROM, DVD-uri, casete video. Nu e nici o indoiala ca multe dispozitive I/O o sa fie inventate in anii

urmatori, asa ca Windows-ul a fost creat cu un spatiu de lucru general la care sa poate sa fie atasate usor noile dispozitive.

Apeluri API pentru I/O sub Windows

Windows-ul are peste 100 de apeluri API pentru a acoperi o mare parte de dispozitive I/O, incluzand mausi, placi de sunet, telefoane, casete video. Probabil cel mai important este sistemul grafic, pentru care sunt mii de apeluri Win32 API.

Exista apeluri Win32 pentru a putea crea, a sterge si a modifica ferestrele. Windows-ul are o mare varietate de stiluri si optiuni care pot fi specificate, incluzand titluri, margini de fereastra, culori, marime, si bara de scroll. Ferestrele pot fi fixate sau mutate si marimea lor poate fi modificata dupa voie. Multe ferestre contin meniuri asa ca exista apeluri Win32 pentru a crea si sterge meniuri si bare de meniuri. Meniuri dinamice pot sa apara sau pot fi scoase.

Casute de dialog pot sa apara pentru a informa utilizatorul de producerea unui anumit eveniment sau pentru a pune o intrebare. Pot sa contina butoane, slidere, sau campuri de text in care sa poata sa fie introduse caractere. De asemenea sunete pot fi asociate cu casutele de dialog, de exemplu pentru mesajele de alarma.

Sunt sute de functii disponibile pentru desenare si pictare, de la modificarea unui pixel pana la lucruri complexe pe regiuni mai mari. Unele apeluri sunt pentru desenarea liniilor si pentru crearea figurilor geometrice inchise. Se remarca controlul asupra culorii texturii, latimea si multe alte atribute ce pot fi modificate.

Grupuri API	Descriere
Managementul ferestrelor	Crearea, distrugerea si managementul ferestrelor
Meniuri	Crearea, distrugerea si extinderea meniurilor si barelor de meniu
Casute de dialog	Aparitia unei casute de dialog si culegerea informatii
Pictare si desenare	Afisare de puncte, linii si figuri geometrice
Text	Afisare text folosind un anumit font, marime, culoare
Icoane si bitmap-uri	Asezarea bitmap-urilor si incoanelor pe ecran

Paleta de culori	Managementul setului de culori disponibile
Clipboard-ul	Trecerea informatiei de la o aplicatie la alta
Intrari	Primirea informatiei de la tastatura si maus

Subcapitol 5. Cateva categorii de apeluri Win32 API

Un alt grup de apeluri se refera la afisarea textului. De fapt, apelul de afisare al textului, TextOut este simplu. Ceea ce este mai complex este managementul culorilor, marime, latime caracter. Partea buna este ca conversia catre bitmap se face de obicei automat.
Bitmap-urile sunt blocuri mici rectangulare care pot fi plasate pe ecran folosind apelul Win32 BitBlt. Sunt folosite pentru icoane si text. Mai sunt multiple apeluri care creaza, distrug si asigura managementul obiectelor de tip icoana.

Multe display-uri folosesc doar 256 sau 65 536 din cele 2^{24} de culori posibile pentru a reprezenta fiecare pixel cu doar 1 sau 2 biti. In aceste cazuri este necesara o paleta de culori pentru a determina daca sunt disponibile 256 sau 65 536 de culori. Apelurile din acest grup creaza, distrug si asigura managementul paletelor de culori, selecteaza culoarea cea mai apropiata culorii date si incearca sa potriveasca culorile de pe ecran cu cele de la imprimantele color.

Multe aplicatii din Windows permit utilizatorului sa selecteze anumite date (text, o parte dintr-un desen, un set de celule dintr-o foaie), sa le puna in clipboard, si sa permita sa fie copiat intr-o alta aplicatie. Clipboard-ul e in general folosit pentru o astfel de transmitere de date. Multe format-uri clipboard sunt definite incluzand text, bitmap, obiecte si metafiles. Ultimele sunt seturi de apeluri Win32 care atunci cand sunt executate deseneaza ceva, permitand desenelor sa fie copiate si sterse dintr-o locatie si transmise in alta locatie.

Pentru dispozitive de intrare trebuie precizat ca nu exista apeluri Win32 pentru aplicatii GUI pentru citirea intrarii de la tastatura . Programul principal e format dintr-o bucla uriasa care preia mesajele de intrare. Cand utilizatorul tasteaza ceva interesant, un mesaj este transmis programului aratandu-i ceea ce a scris utilizatorul. Totusi exista apeluri legate de maus precum citirea coordonatelor (x,y) care arata pozitia si starea butoanelor mausului. Unele apeluri de intrare sunt de fapt apeluri de iesire, precum selectarea cu ajutorul cursorului mausului a unei icoane si plimbarea ei pe ecran (de fapt e o iesire

catre display). Pentru aplicatii nonGUI este posibil sa se citeasca de la tastatura.

Functii API	UNIX	Descriere
CreateFile	open	creaza un fisier sau deschide unul existent
DeleteFile	unlink	distruge un fisier existent
CloseHandle	close	inchide un fisier
ReadFile	read	citeste continutul unui fisier
WriteFile	write	scrie date intr-un fisier
SetFilePointer	lseek	seteaza pointerul de fisier la o anumita pozitie
GetFileAtributes	stat	returneaza proprietatile fisierului
LockFile	fcntl	blocheaza o parte din fisier pt a avea excludere mutuala
UnlockFile	fcntl	deblocheaza o regiune blocata anterior

Principalele functii Win32 API pentru I/O la fisiere (coloana a doua arata echivalentul sub UNIX)

Gavrilă Cosmin

Bibliografie

1. S. Tanenbaum - **Sisteme de operare moderne**
2. www.wikipedia.com
3. Operating Systems System Calls and I/O by Henry Newman