ADMINISTRAREA UNITĂŢILOR(perifericelor)

Organizarea sistemului de I/O

Interfaţă API

Administrarea directă a I/O cu testare periodică(polling)

Administrarea operaţiilor de I/O orientată pe întreruperi

Accesul direct la memorie

Utilizarea zonelor tampon(“buffering”)

Organizarea sistemului de I/O.

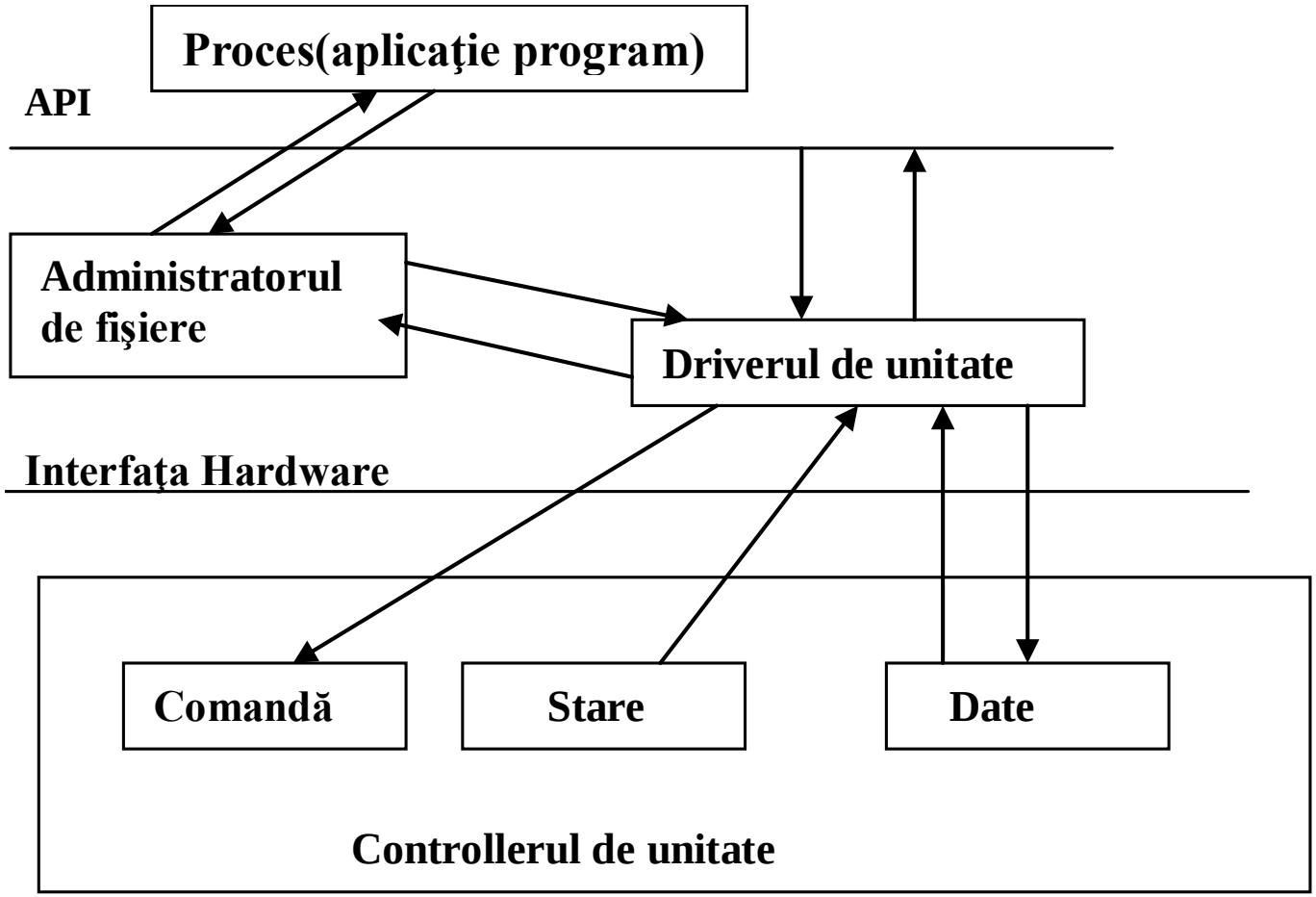
- Un sistem de I/O trebuie să răspundă la cerinţele următoare:
- ►Să poată recunoaşte caractere, indiferent de codul în care acestea sunt reprezentate.
- ►Să realizeze independenţa faţă de dispozitvele periferice a programelor, în sensul că acestea nu trebuie să sufere modificări importante atunci când tipul dispozitivului periferic utilizat este altul decât cel prevăzut iniţial. Acest lucru este posibil prin furnizarea unor operaţii a căror sintaxă şi semantică să fie aceeaşi pentru o clasă cât mai mare de unităţi de I/O, precum şi denumirea uniformă a perifericelor, lucru realizat prin asocierea de fişiere acestor dispozitive.
- ►Realizarea operaţiilor într-un timp cât mai mic, adică găsirea unor metode care să anihileze diferenţele între viteza de operare a CPU şi cea a perifericelor.

- Unităţile de I/O (perifericele) sunt folosite pentru a plasa date în memoria primară şi pentru a stoca cantităţi mari de date pentru o perioadă lungă de timp.
- Ele pot fi:

  - unităţi de stocare (unităţi bloc): discurile;

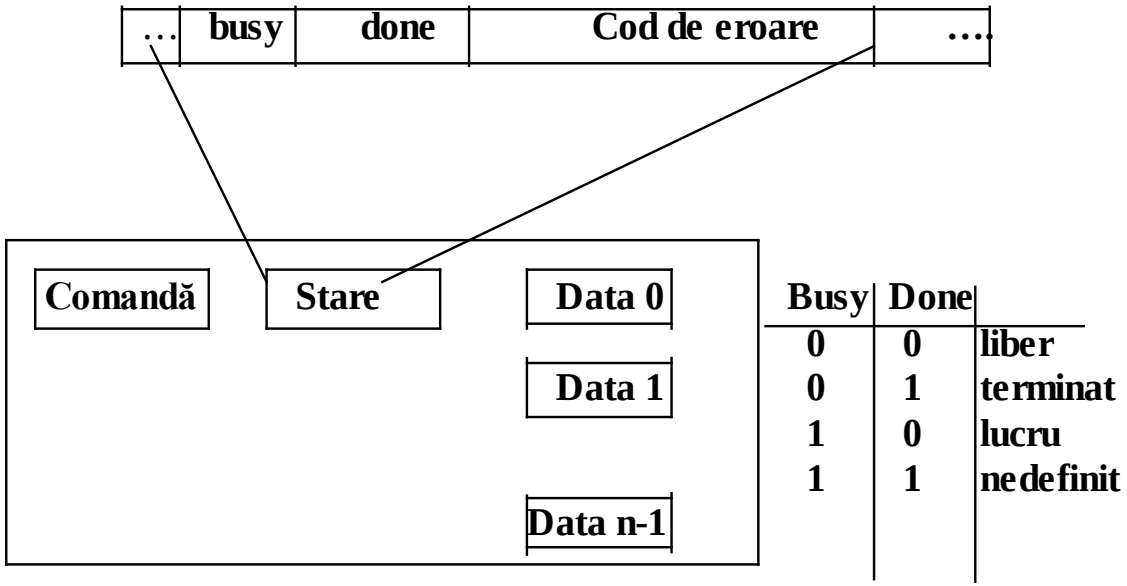
  - unităţi de tip caracter: tastatura, mouse-ul, display-ul terminalului

  - unităţi de comunicaţie:portul serial conectat la un modem sau o interfaţă la reţea.
- Dispozitivele bloc stochează informaţia sub forma unor blocuri de dimensiune fixă, care pot fi accesate individual, prin intermediul unei adrese.
- Dispozitivele de tip caracter lucrează cu şiruri de caractere care nu sunt structurate ca blocuri.
- Există şi dispozitive periferice care nu pot fi încadrate în niciuna dintre categoriile descrise anterior, cum este cazul ceasului în timp real, a cărui sarcină este de a genera întreruperi la intervale bine definite.
- În sistemele de operare moderne, administrarea unităţilor este implementată prin interacţiunea **driverilor de unitate** şi **întreruperilor** (metoda de administrare a I/O orientată pe drivere şi întreruperi) sau prin folosirea **numai** a driverilor prin **evitarea** întreruperilor (metoda administrării directe a I/O cu testare periodică - polling). Figura urm. ilustrează componentele implicate în realizarea operaţiilor de I/O.


Proces(aplicaţie program)
API
Administratorul de fişiere
Driverul de unitate
Interfaţa Hardware
Comandă
Stare
Date
Controllerul de unitate

- Fiecare unitate foloseşte un **controller de unitate** pentru a o conecta la adresele calculatorului şi la magistrala de date.
- Controller-ul oferă un set de componente fizice pe care instrucţiunile CPU le pot manipula pentru a efectua operaţii de I/O.
- Ca şi construcţie, controller-ele diferă, dar fiecare oferă aceeaşi interfaţă de bază.
- SO ascunde aceste detalii de funcţionare ale controller-lor, oferind programatorilor funcţii abstracte pentru accesul la o unitate, scrierea/citirea de informaţii etc.
- În timpul lucrului, orice unitate ar trebui monitorizată de către CPU. Acest lucru este făcut prin controller-ul de unitate.
- Interfaţa între controller şi unitate caută să rezolve problema compatibilităţii între unităţi fabricate de diverşi producători. O astfel de interfaţă este SCSI (Small Computer Serial Interface).

- Interfaţa între magistrală şi controller este importantă pentru cei care se ocupă cu adăugarea unei noi unităţi la arhitectura existentă, pentru ca unitatea adăugată să poată lucra împreună cu celelalte componente.
- **Interfaţa** între componenta software şi controller se realizează prin intermediul **regiştrilor** controller-ului. Această interfaţă defineşte modul cum componenta soft manipulează controller-ul pentru a executa operaţiile de I/O.
- Figura urm. ilustrează interfaţa software cu un controller.
- Unităţile conţin două fanioane în registrul lor de stare: **busy** şi **done**.
- Dacă ambele fanioane sunt setate pe `0`(false), atunci componenta software poate plasa o comandă în registrul de comenzi pentru a activa unitatea.
- După ce componenta software a pus informaţii într-unul sau mai mulţi regiştri de date, pentru o operaţie de ieşire, unitatea este disponibilă pentru a fi utilizată.
- Prezenţa unei noi comenzi de I/O are ca efect poziţionarea fanionului **busy** pe true, începerea efectuării operaţiei şi datele din registrul de date sunt scrise pe unitate.



Busy	Done	
0	0	liber
0	1	terminat
1	0	lucru
1	1	nedefinit

- O operaţie de citire se efectuează dual.
- Procesul poate detecta starea operaţiei verificând registrul de stare. Când operaţia de I/O a fost terminată(cu succes sau eşec), controller-ul şterge fanionul **busy** şi setează fanionul **done**.
- Odată cu terminarea unei operaţii **read**, respective **write** datele sunt copiate de pe unitate în registrul de date, respectiv invers.
- Dacă după o operaţie de scriere ambele fanioane ale unităţii au fost setate pe false, scrierea de noi informaţii în registrul de date va fi sigură.
- Când procesul citeşte date din controller, acesta şterge registrul done, adică controller-ul este din nou gata de utilizare.
- Dacă operaţia se termină eronat, câmpul cod de eroare va fi setat.
- Controller-ele pot conţine o mică zonă de memorie (buffer), pentru a stoca datele după ce au fost citite, înainte de a fi transmise CPU pentru prelucrare. Invers, în memorie pot fi păstrate datele care aşteaptă să fie scrise.
- Controller-ul de unitate furnizează o interfaţă folosită de către mediul microprogramat de la cel mai înalt nivel.
- Componenta SO care manipulează dispozitivele de I/O este formată din **driverele de unitate**.

- **Drivere de unitate.**
- Driverul are două sarcini principale:

►Implementează o interfaţă API (Application Programming Interface) la care procesul are acces.

►Furnizează operaţii dependente de unitate, corespunzătoare funcţiilor implementate de API.

- Fiecare sistem de operare defineşte o arhitectură pentru sistemul său de administrare a unităţilor. Nu există un model universal, dar fiecare din aceste sisteme conţine o API şi o interfaţă între driver şi nucleul SO.
- API furnizează un set de funcţii pe care un programator le poate apela pentru a utiliza o unitate.
- În general, o unitate este utilizată pentru comunicare sau pentru stocare. Astfel, unităţile de intrare produc informaţii care pot fi transmise către diferite medii de comunicaţie sau către unităţi de stocare a datelor.
- Procesele produc informaţii care, fie sunt transmise printr-un mediu de comunicaţie, fie sunt stocate pe un periferic pentru a fi utilizate mai târziu.

- **Administratorul unităţii** trebuie să urmărească starea unităţii, când este liberă, când este folosită şi care proces o foloseşte.
- Administratorul unităţii poate întreţine, în plus faţă de informaţiile păstrate în tabela de stare a unităţii, un descriptor de unitate care specifică alte caracteristici ale unităţii, determinate de proiectantul de software.
- Deci interfeţele driverelor includ funcţiile de deschidere(open) şi închidere(close) pentru a permite administratorului să înceapă/termine utilizarea unităţii de către un proces.
- Comanda **open** alocă perifericul şi iniţializează tabelele şi unitatea pentru utilizare iar **close** şterge informaţiile scrise în tabele, eliberând astfel unitatea.
- Fiecare driver de unitate furnizează funcţii care să permită unui program să citească/scrie de pe/pe o unitate. Funcţiile de **citire/scriere** diferă din punct de vedere sintactic în funcţie de tipul unităţii(caracter sau bloc).

- De **exemplu**, unităţile de tip bloc conţin comenzi de tipul `read()`, `write()` şi `seek()`, pentru a citi, scrie de pe/pe unitatea respectivă şi pentru a accesa direct un bloc de informaţii.
- În cazul unităţilor de tip caracter, cum este cazul tastaturii sau ecranului unui monitor, apelurile de sistem ale acesteia permit comenzi de tipul `get()` sau `put()`, pentru citirea/scrierea caracter cu caracter.
- Semantica fiecărei funcţii depinde de modul de accesare a unităţii (secvenţială sau directă).
- Proiectantul SO trebuie să furnizeze şi alte funcţii API care să permită unui proces, de **exemplu** să aloce/elibereze o unitate, să pună sub/scoată de sub tensiune un disc, să modifice imaginea pe un anumit display etc.

Observaţii.

- Se reduce cantitatea de informaţii pe care procesul (aplicaţia) trebuie să le cunoască pentru a utiliza unitatea respectivă.
- Fiecare driver este specific unei anumite unităţi, fiind capabil să furnizeze controllerului comenzile corespunzătoare, să interpreteze corect conţinutul registrului de stare al controllerului şi să transforme date la/de la regiştrii de date ai controllerului, în confomitate cu cerinţele operaţiei respective.
- Un proces care foloseşte o unitate, îi transmite acesteia comenzi şi schimbă date cu driverul de administrare al unităţii respective.
- Driverele sunt componente ale administratorului de unitate folosite de către un proces pentru a cere efectuarea unei operaţii de I/O.
- Când procesul face o cerere către driverul de unitate, acesta o translatează într-o acţiune dependentă de controller, pentru a realiza interacţiunea cu unitatea respectivă. După ce unitatea îşi începe lucrul, driverul fie testează periodic controllerul pentru a detecta sfârşitul operaţiei, fie plasează informaţii în tabela de stare a unităţii pentru a pregăti lansarea unei întreruperi.

Interfaţa cu nucleul

- Driverul de unitate execută instrucţiuni privilegiate, atunci când interacţionează cu controllerul de unitate. Deci driverul trebuie să fie executat ca parte a SO şi nu ca parte a programului.
- Driverul trebuie să fie capabil să citească/scrie informaţii din/în spaţiile de adrese ale diferitelor procese, deoarece aceeaşi unitate poate fi folosită de către diferite procese.
- În sistemele mai vechi, driverul este încorporat în SO prin modificarea codului sursă al SO şi, apoi recompilarea acestuia. Deci, proprietarul sistemului de calcul trebuie să deţină sursele SO şi să cunoască modul cum trebuie realizată compilarea lui. În condiţiile existenţei sistemelor deschise, acest mod de lucru devine inacceptabil.

- SO moderne simplifică instalarea driverelor prin utilizarea **driverelor de unitate reconfigurabile**. Astfel de sisteme permit adăugarea unui driver de unitate fără a fi necesară recompilarea SO, fiind necesară numai **reconfigurarea** sistemului printr-un set de operaţii oferite chiar de către SO.
- Pentru aceasta, la nivelul fiecărei unităţi, există o tabelă care conţine pointeri către modulele sursă ale funcţiilor interfeţei. Un driver de unitate reconfigurabil trebuie să fie standardizat, adică să ofere aceeaşi interfaţă.
- Deoarece driverul de unitate este adăugat nucleului după ce acesta a fost compilat, nucleul trebuie să furnizeze o interfaţă care să permită driverului de unitate să aloce spaţiu pentru buffere, să manipuleze tabele ale nucleului etc.

- **Interfaţa cu CPU.** Perifericele şi CPU sunt componente distincte care sunt capabile să lucreze independent.
- Administratorul unităţii trebuie să furnizeze metodele prin care să se coordoneze execuţia procesului şi operaţiile de I/O.
- În efectuarea operaţiilor de I/O sunt implicate trei componente: controllerul, driverul şi apeluri de funcţii de I/O conţinute în programul în execuţie. Între acestea trebuie să existe o bună coordonare.
- Deoarece controllerul este o componentă hardware, el ori execută o operatie de I/O, ori este în aşteptare activă pentru a primi în registrul de comenzi codul unei operaţii.
- Driverul unităţii este o componentă software a SO invocată de către procesul în execuţie. Fiecare funcţie a driverului este o procedură apelată de către procesul în execuţie. Toate aceste componente software implicate în lucrul cu controllerul unităţii sunt nişte procese, gestionate de către CPU pe baza algoritmilor de planificare studiaţi.

Administrarea directă a I/O cu testare periodică(polling)

- Administrarea directă a I/O se referă la efectuarea operaţiilor de I/O cu participarea CPU, care este responsabilă de transferul datelor între memoria internă a SC şi regiştrii de date ai controllerului. CPU porneşte unitatea şi testează periodic registrul de stare al acesteia, pentru a determina momentul când operaţia s-a terminat.
- În cazul efectuării unei operaţii de citire, această metodă constă în următorii paşi:
- 1. Procesul în execuţie(corespunzator unei aplicaţii) cere efectuarea unei operaţii **read.**
- 2. Driverul de unitate(specificat în comanda de citire) testează registrul de stare pentru a vedea dacă unitatea este liberă. Dacă unitatea este ocupată, driverul asteaptă până când aceasta se eliberează (aşteptare activă), după care se trece la pasul următor.
- 3. Driverul pune codul comenzii de intrare în registrul comandă al controllerului, pornind astfel unitatea.
- 4. Driverul citeşte în mod repetat registrul de stare, atâta timp cât este în aşteptarea terminării operaţiei.
- 5. Driverul copiază conţinutul regiştrilor de date ai controlerului în spaţiul de memorie al procesului utilizator.
- 6. După terminarea operaţiei, se revine la următoarea instrucţiune din program.

Paşii necesari pentru a executa o operaţie de ieşire(scriere) sunt:

- 1. Procesul în execuţie(aplicaţia) cere execuţia unei operaţii **write.**
- 2. Driverul unităţii specificate interoghează registrul de stare al acesteia, pentru a vedea dacă unitatea este liberă. Dacă unitatea este ocupată, driverul aşteaptă până când aceasta devine liberă, după care se trece la pasul următor.
- 3. Driverul transferă date din spaţiul de memorie al procesului în regiştrii de date ai controllerului.
- 4. Driverul scrie un cod de comandă de ieşire în registrul comandă, pornind astfel unitatea.
- 5. Driverul citeşte în mod repetat registrul de stare, atâta timp cât aşteaptă terminarea operaţiei.
- 6. După terminarea operaţiei, se revine la următoarea instrucţiune din program.

Observaţii.

Fiecare operaţie de I/O necesită o coordonare a componentelor software şi hardware implicate. Prin metoda anterioară, această coordonare este realizată prin încapsularea părţii soft a interacţiunii (driverul), cu controllerul unităţii respective.

Această metodă duce la încărcarea CPU cu sarcini suplimentare, şi anume verificarea periodică a stării controllerului.

Manipularea intreruperilor

- **Execuţia unui program**. CPU extrage şi execută instrucţiunile cod maşină ale procesului încărcat în memoria internă. În acest sens, CPU conţine:
- ►o componentă care extrage o instrucţiune memorată într-o locaţie de memorie;
- ►o componentă care decodifică instrucţiunea;
- ►o componentă care se ocupă de execuţia instrucţiunii, împreună cu alte componente ale SC.
- Regiştrii **contor de program** PC (Program Counter), respectiv **registrul instrucţiune** IR (Instrucţion Register), conţin adresa de memorie, respectiv o copie a instrucţiunii în curs de prelucrare.

- Modul de operare al CPU se desfăşoară după următorul **algoritm**:

```
PC = <Adresa de început a procesului> ;
IR = M[PC];
HaltFlag = CLEAR;
While (HaltFlag nu este setat în timpul execuţiei )
{
      PC = PC+1;
      Execute(IR);
      IR = M[PC];
}
```

- Vectorul `M` identifică locaţiile de memorie internă ale SC, adică `M[i]` reprezintă locaţia de memorie `i`.
- Când un proces urmează să fie executat, **PC** este încărcat cu adresa primei sale instrucţiuni, iar în **IR** este încărcat codul instrucţiunii respective.
- După terminarea execuţiei unei instrucţiuni, **PC** este incrementat, adică se trece la execuţia următoarei instrucţiuni, ş.a.m.d.
- Fanionul **HaltFlag** este testat înainte de a se trece la execuţia instrucţiunii, pentru a se verifica dacă CPU este oprită.
- **Execute** reprezintă procedura care corespunde instrucţiunii extrase.

- Testarea fanioanele busy-done de către CPU prin intermediul driverelor de unitate duce la utilizarea ineficientă a CPU.
- Cea mai bună soluţie pentru desfăşurarea paralelă a operaţiilor de I/O şi a activităţii CPU, este ca atunci când operaţia de I/O se termină, să se semanaleze acest lucru procesorului, lucru care se realizează prin intermediul **întreruperilor**.
- În partea de hardware este încorporat un fanion **InReq** (Interrupt Request) şi unitatea de control este astfel concepută încât să poată verifica fanionul IR de fiecare dată când extrage o nouă instrucţiune.
- Algoritmul de extragere şi execuţie al instrucţiunilor procesului atunci când SC foloseşte întreruperi este:

```
While (HaltFlag nu este setat)
{
IR = M[PC];
PC = PC + 1;
Execute(IR);
If (InReq) // Are loc întrerupere procesului curernt
      {M[0] = PC; // Salvează valoarea curentă a lui
            PC la adresa 0
      PC=M[1]}
}
```

- Partea de hardware conectează toate fanioanele **done** ale unităţilor la fanionul **InReq** prin efectuarea unei operaţii SAU. Oricând este setat un fanion **done** al unui controller, fanionul **InReq** va fi şi el setat.
- CPU va fi atenţionată despre terminarea operaţiei de I/O în intervalul de timp cât execută o instrucţiune, urmând ca după terminarea acesteia să trateze întreruperea respectivă.
- Semnalul de la componenta hardware, la cea software (întreruperea) are ca efect oprirea execuţiei secvenţei de instrucţiuni ale procesului respectiv de către processor (adresate de către PC) şi saltul la o nouă secvenţă de instrucţiuni, a cărei adresă este conţinută la o adresă de memorie(de exemplu `M[1]`).
- Înainte de a se efectua saltul, valoarea lui PC (adresa instrucţiunii care urmează să fie executată) împreună cu starea procesului sunt salvate prin intermediul hardwarelui. Adresa componentei SO care manipulează întreruperile (**IH** - Interrupt Handler) este memorată în locaţia de memorie `M[1]`**.**

- Când IH îşi începe execuţia, regiştrii CPU conţin valori care urmează să fie folosite de către procesul care a fost întrerupt.
- IH trebuie să execute o comutare de context pentru a salva conţinutul tuturor regiştrilor de stare şi generali utilizaţi de proces şi să-şi pună propriile valori în regiştrii CPU pentru a putea realiza terminarea operaţiei de I/O corespunzătoare întreruperii lansate.
- Algoritmul după care lucrează IH este:

```
SaveProcessorState();
For (i=0;i<Numărul de unităţi; i++)
    If (Device[i].done == 1) goto device_handler(i);
        If (InReq && InterruptEnabled)
        {
            DisableInterrupts();
            M[0] = PC;
            PC = M[1]
        }
```

- **IH** testează valorile fanioanelor done ale controllerelor de unitate pentru a determina care unitate a terminat operaţia de I/O(a cauzat întreruperea).
- Dacă două sau mai multe unităţi termină operaţia de I/O în timpul execuţiei aceleiaşi instruc. cod maşină, atunci, se va detecta numai prima dintre ele.
- Odată ce cauza întreruperii a fost determinată, **IH** face un salt la codul întreruperii coresp.unităţii a cărei terminare a operaţiei de I/O trebuie tratată.
- Dacă o întrerupere este declanşată în momentul când **IH** este în timpul execuţiei, adică tratează o întrerupere apărută anterior, aceasta va aştepta până la terminarea întrerup. ant.
- Dacă **IH** ar trata ambele întreruperi în paralel, ar exista posibilitatea ca apariţia unor erori în execuţia uneia dintre întruperi să ducă la eşecul execuţiei ambelor întreruperi.
- Orice instrucţiune care setează fanionul **HaltFlag** al CPU este o instrucţiune privilegiată, deoarece poziţionarea pe TRUE a lui **HaltFlag** are ca efect oprirea execuţiei oricărei instrucţiuni.
- Dacă CPU lucrează în mod utilizator şi un proces doreşte să înceapă execuţia în modul supervizor, atunci modul de lucru este comutat din modul utilizator în cel supervizor si CPU va începe să execute instrucţiunile nucleului sistemului.
- Modul supervizor este setat printr-o instrucţiune cod maşină `trap`.
- Mai întâi CPU comută din modul utilizator în supervizor, după care va sări la o adresă care conţine adresa acelei secvenţe de instrucţiuni(întreruperi) corespunzătoare argumentului instrucţiunii `trap`.

Administrarea operaţiilor de I/O orientată pe întreruperi

- Din punct de vedere funcţional, operaţia de I/O este realizată prin cooperarea mai multor componente hard şi soft, şi anume:
- ►Driverul de unitate, care declanşează operaţia.
- ►Tabela de stare a unităţii.
- ►Administratorul (handler-ul) întreruperilor.
- ►Administratorul unităţii.
- Pentru a realiza o **operaţie de intrare** sunt efectuaţi următorii paşi:
- 1. Procesul cere efectuarea unei operaţii **read.**
- 2. Driverul interoghează registrul de stare pentru a vedea dacă unitatea este liberă. Dacă este ocupată, driverul aşteaptă ca aceasta să devină liberă (aşteptare activă), după care se trece la pasul următor.
- 3. Driverul pune codul operaţiei în registrul comandă al controllerului, pornind astfel unitatea.

- 4. Driverul îşi termină sarcina, salvează informaţia cu privire la operaţia pe care a început-o în intrarea din **tabela de stare a unităţilor,** corespunzătoare unităţii folosite; aceasta conţine adresa de retur din procesul care a lansat operaţia şi alţi parametri speciali ai operaţiei de I/O. Astfel, din acest moment, CPU poate fi folosită de către alte programe, deci administratorul unităţii transmite un mesaj corespunzător componentei de planificare a administratorului de procese.
- 5. Unitatea termină de efectuat operaţia şi atunci lansează o întrerupere către CPU; de rezolvarea acesteia se va ocupa mai departe handler-ul de întreruperi.
- 6. Administratorul(handler-ul) întreruperilor determină care unitate a cauzat întreruperea, după care face un salt la administratorul (handler-ul) acelei unităţi.
- 7. Administratorul unităţii găseşte informaţiile de stare ale unităţii de I/O în tabela de stare a unităţilor; sunt utilizate pentru terminarea operaţiei.
- 8. Administratorul unităţilor copiază conţinutul regiştrilor de date ai controllerului în spaţiul de memorie al procesului.
- 9. Administratorul unităţii transferă controlul procesului apelant.

- **Observaţii.**
- Operaţia de ieşire se desfăşoară similar. Din punctul de vedere al procesului în execuţie, această activitate are aceeaşi semnificaţie ca oricare apel de procedură.
- Când este folosită o instrucţiune **read** într-un program, aceasta trebuie să se termine înainte ca următoarea instrucţiune să fie executată. Dacă CPU a început să execute o instrucţuine în care este implicată o variabilă în care s-a efectuat citirea, după ce driverul de unitate a început să execute operaţia de citire, dar înainte ca ea să se termine, atunci instrucţiunea respectivă va fi executată folosind o valoare veche a variabilei respective, nu cea care este în curs de citire de la unitate. Pentru a preveni aceasta situaţie SO blochează explicit procesul care a iniţiat apelul **read**. SO aşteaptă ca procesul să termine operaţia de I/O înainte de a executa instrucţiunea următoare.
- Deşi procesul însuşi nu este capabil să obţină avantajele desfăşurării în paralel a activităţii CPU şi a operaţiilor de I/O, totuşi SO poate comuta CPU la un alt proces, oricând un proces cere efectuarea unei operatii de I/O. Astfel, perfomanţele generale ale sistemului pot fi îmbunătăţite prin lucrul în paralel al CPU, cu efectuarea unei operaţii de I/O. Când operaţia de I/O s-a terminat, procesul care a iniţiat operaţia va fi replanificat. Mecanismul de planificarea al proceselor va interveni atât în momentul terminării operaţiei de I/O, cât şi atunci când se realizează o serializare a operaţiei de I/O şi a instrucţiunii imediat următoare acesteia.

I/O cu corespondenta in memorie (Memory-Mapped I/O)

- Fiecare controller conţine regiştri de control folosiţi pentru a comunica cu procesorul; prin aceşti regiştri SO poate transmite comenzi sau poate consulta starea echipamentului. De asemenea, multe echipamente au o memorie tampon de date in care SO poate citi sau scrie.
- **Exemplu.** Memoria video este o memorie tampon de tip RAM, în care se scriu date ce se vor transforma în pixeli pe ecran.
- Procesorul poate comunica cu regiştrii de control şi memoria tampon a unui echipament prin trei modalităţi, care vor fi prezentate în continuare.
- (i) Fiecărui registru i se asociază un număr de PORT DE I/O, care este un întreg reprezentat pe 8 sau 16 biţi; folosind instructiuni speciale de I/O ca: “IN REG, PORT” şi “OUT PORT, REG” procesorul poate copia registrul de control PORT în registrul său REG, respectiv poate scrie conţinutul registruluin REG într-un registru de control.
- Spaţiul de adrese pentru memorie şi cel pentru I/O sunt diferite.
- **Exemplu.** Instrucţiunile "IN REG,6" si "MOV REG,6" au semnificaţii diferite: prima citeşte în registrul REG conţinutul portului 6; a doua citeste în REG cuvântul de memorie de la adresa 6; deci cei doi 6 se referă la zone de adrese de memorie diferite şi independente.
- Aceasta metoda a fost folosita de cele mai multe dintre primele calculatoare, inclusiv IBM 360 si succesorii sai.

- (ii) Regiştrii de control sunt puşi în spaţiul de adrese de memorie. Fiecărui registru de control îi este repartizată o adresă unică de memorie, căreia nu-i corespunde o locaţie de memorie fizica sau virtuala. Aceste sisteme se numesc I/O CU CORESPONDENTA IN MEMORIE (MEMORY-MAPPED I/O). Această metodă a fost introdusă de minicalculatorul PDP-11. De regulă, adresele alocate regiştrilor de control se află la sfârşitul spaţiului de adrese.
- (iii) O schema hibridă foloseşte o **memorie tampon** de I/O în spaţiul de adrese de memorie şi porturi de I/O pentru regiştrii de control.
- **Exemplu** Pentium utilizează această arhitectură, cu adresele 640K - 1M rezervate pentru memoriile tampon ale echipamentelor compatibile cu IMB PC şi porturi de I/O de la 0 la 64K.
- Toate aceste scheme functioneaza astfel:

  - Când procesorul vrea să citească un cuvânt, fie din memorie fie dintr-un port de I/O, pune adresa dorită pe liniile de adresă ale magistralei (bus) şi apoi transmite un semnal de citire pe linia de control a magistralei. Pe o altă linie a magistralei pune specificaţia dacă este vorba de spaţiul de adrese de memorie sau spaţiul de I/O. Atunci fie memoria, fie echipamentul de I/O va răpunde cererii.

  - Daca exista doar spatiu de memorie, atunci fiecare modul de memorie si fiecare echipament de I/O compara liniile de adresa cu domeniul de adrese pe care il foloseste; daca adresa se afla in domeniul sau, raspunde cererii, deoarece nici o adresa nu este repartizata atat memoriei cat si echipamentelor de I/O(nu exista ambiguitate).

- **Avantajele sistemului I/O cu corespondenţă în memorie**.
- În cazul (i) sunt necesare instrucţiuni speciale de I/O pentru citirea/ scrierea la porturi (IN/OUT), care nu se regăsesc în limbajele C/C++, astfel încât este necesară folosirea limbajului de asamblare.
- In cazul (ii) regiştrii de control ai echipamentelor se pot asimila cu simple variabile alocate în memorie şi se pot accesa în C/C++ ca orice alta variabilă, deci driverele pentru echipamente pot fi scrise în aceste limbaje evoluate.
- Nu este necesar un mecanism de protecţie suplimentar pentru a împiedica procesele utilizator să efectueze operatii de I/O. E suficient ca SO să nu permită includerea porţiunii de spaţiu de adrese ce conţine regiştrii de control în spaţiul virtual de adrese ale vreunui utilizator.
- Dacă fiecare echipament are regiştrii de control în pagini diferite ale spaţiului de adrese, SO poate permite unui utilizator să controleze anumite echipamente, incluzând paginile respective în tabela lui de pagini.În felul acesta, drivere diferite pot fi plasate în spaţii de adrese diferite, reducându-se dimensiunea nucleului şi făcând totodată ca driverele să nu intefereze între ele.
- Aceleaşi instrucţiuni care pot referenţia memoria pot iniţializa şi regiştrii de control.

Accesul direct la memorie.

- În cazul accesării directe a perifericelor de către CPU cu testare periodică, CPU este utilizată pentru a transfera date între regiştrii de date ai controllerului şi memoria primară. Driverul de unitate copiază date din zona de date a procesului, în controller pentru fiecare operaţie de ieşire şi invers, pentru operaţiile de intrare. În cazul utilizării întreruperilor, administratorul unităţii se ocupă cu sarcina de transfer. Ambele modalităţi presupun implicarea CPU, deci un consum al timpului său.
- Multe dintre calculatoarele moderne folosesc un alt mecanism care cere o implicare a CPU mai redusă în efectuarea acestor operaţii, prin utilizarea unui procesor specializat numit **controller DMA** (**D**irect **M**emory **A**ccess). Mecanismul DMA presupune că operaţia de copiere în memorie să fie efectuată de către controller şi nu de către unitatea centrală.
- Înţelegerea între controllerul DMA şi controllerul de unitate este realizată prin intermediul unei perechi de semnale numite **cerere DMA** şi **confirmare DMA**.
- Când are date de transmis, controllerul de unitate iniţiază o cerere DMA şi aşteaptă până când primeşte o confirmare DMA. După ce s-a efectuat transferul, controllerul DMA semnalează terminarea operaţiei printr-o întrerupere.
- Deşi controllerul DMA lucrează fără a ţine ocupată CPU, totuşi cele două concurează pentru obţinerea accesului la magistrală. Cu toate acestea, DMA creşte performanţele calculatorului, în sensul că operaţiile de I/O se fac printr-o participare mai restrânsă a CPU.

- Procesul de transfer al informaţiilor de pe disc, folosind DMA se desfăşoară astfel:
- 1. Driverului de unitate i se comunică să transmită date în bufferul de la adresa x.
- 2. Driverul de unitate transmite controllerului de disc să transfere c cuvinte de memorie de pe disc în bufferul de la adresa x.
- 3. Controllerul de disc începe transferul DMA.
- 4. Controllerul de disc transmite fiecare cuvânt controllerului DMA.
- 5. Controllerul DMA transferă cuvintele primite în bufferul de la adresa x, incrementând adresa de memorie şi decrementând c, până când c=0.
- 6. Când c=0, controllerul DMA semnalează CPU prin intermediul unei întreruperi, că transferul de date s-a terminat.

- **Observaţii.**
- - Folosind DMA, controllerul de unitate nu trebuie să conţină obligatoriu regiştrii de date, deoarece controllerul DMA poate face transferul direct de la/la unitate.
- - Controllerul trebuie să conţină un registru de adrese, în care se încarcă un pointer către un anumit bloc de memorie, de unde controllerul preia informaţii sau depune informaţii.
- - De asemenea, trebuie să cunoască numărul de octeţi ai blocului pe care trebuie să-l manevreze.
- -Anumite calculatoare folosesc adrese de memorie fizică pentru DMA; altele, mai perfecţionate execută transferul folosind adrese virtuale, adică adrese din memoria secundară, prin mecanismul denumit DVMA(Direct Virtual Memory Acces).

Utilizarea zonelor tampon(“buffering”)

- Utilizarea zonelor tampon la intrare (“input buffering”) este tehnica prin care informaţiile citite de la un dispozitiv de intrare sunt depuse într-o zonă de memorie primară, înainte ca procesul să le solicite pentru a le prelucra.
- Utilizarea zonelor tampon la ieşire (“output buffering”) este metoda prin care datele de ieşire sunt salvate în memorie şi apoi scrise pe o anumită unitate, în acelaşi timp cu continuarea execuţiei procesului.
- Buffering-ul are ca scop lucrul în paralel al CPU şi unităţilor periferice.
- Pentru a înţelege mai bine cum buffering-ul poate creşte performanţele sistemului de calcul, să considerăm câteva caracteristici ale proceselor:
- - Anumite procese sunt orientate către operaţii de I/O, consumând o mare cantitate de timp pentru a efectua aceste operaţii.
- - Alte procese sunt orientate spre calcule.
- - Multe procese conţin faze în care sunt orientate către operaţii de I/O sau faze când sunt orientate către calcule.
- Driverul poate gestiona unul sau mai multe buffere, unde depune caracterele citite, înainte ca acestea să fie prelucrate de către procesul care a iniţiat cererea, respectiv de unde preia caracterele depuse de CPU(procesul utilizator) la scriere.

- Numărul de buffere poate fi extins de la 2 la n.
- Producătorul de date (controllerul în cazul operaţiilor de citire, respectiv CPU pentru operaţiile de scriere) scrie în bufferul `i` în timp ce consumatorul (controllerul în cazul operaţiilor de scriere, respectiv CPU în cazul operaţiilor de citire) citeşte din bufferul `j`.
- În această configuraţie, bufferele de la `j` la `n-1` şi de la `0` la `i-1` sunt pline.
- În timp ce producătorul introduce date în bufferul `i`, consumatorul citeşte date din bufferele `j, j+1,...,n-1` şi `0,1,...,i-1`.
- Reciproc, producătorul poate umple bufferele `i, i+1,...,j-1` în timp ce consumatorul citeşte bufferul `j`.
- În această tehnică de dispunere circulară a bufferelor, producătorul nu poate trece în zona bufferelor administrate de consumator, deoarece el poate scrie peste buffere care nu au fost încă citite.
- Producătorul poate scrie date numai în bufferele până la `j-1` în timp ce datele din bufferul `j` aşteaptă să fie prelucrate.
- În mod similar, consumatorul, nu poate trece în zonele administrate de producător deoarece el ar aştepta să citească informaţii, înainte ca acestea să fi fost plasate în zonele respective de către producător.

- **Observaţii.** Numărul bufferelor de citire, respectiv de scriere trebuie ales în funcţie de tipul proceselor. Pentru un proces orientat către I/O este mai potrivit să se aloce un număr cât mai mare de buffere de citire, pe când pentru un proces orientat spre calcule este necesar ca numărul de buffere de scriere să fie mai mare.
- Orice proces poate fi la un moment dat orientat către I/O, ca mai târziu să devină orientat către calcule şi reciproc. Deci se impune un mecanism de comutare a procesului dintr-o stare în alta şi, corespunzător de modificare a configuraţiei zonelor tampon.