11 Intrări / ieşiri la nivel fizic

Acest nivel din structura unui sistem de operare este unul extrem de dependent de hardware. In prezenta secţiune vom evidenţia o serie de concepte şi tehnici pe care le vom "debarasa", pe cât posibil, de detaliile dependenţelor specifice hard.

11.1 Tehnica zonelor tampon temporare (buffering, caching)

Această tehnică este extrem de folosită în procesele de schimb de informaţii dintre memoria internă a sistemelor şi mediul exterior: periferice locale, conexiuni prin reţea etc. De asemenea, această tehnică este folosită şi pentru schimbul între diverse nivele ale memoriei, schimb între memoria cache şi memoria internă, între cea internă şi cea secundară etc.

Fiind întâlnită în multe situaţii, literatura o denumeşte diferit, în funcţie de context: *metoda zonelor tampon multiple, cache-ingul paginilor web, pool de pagini recent utilizate, buffering, stoc tampon* etc. De fapt este vorba de o aceeaşi metodă, "îmbrăcată" în contexte diferite, aşa cum se va vedea în secţiunile următoare.

11.1.1 Mecanismul zonelor tampon temporare

Să luăm mai întâi un exemplu din lumea reală. Să presupunem că o firmă îmbuteliază apă minerală. O serie de clienţi doresc să se aprovizioneze cu apă de la firmă şi să aibă pe cât posibil mereu apă proaspătă. O soluţie ideală ar fi conectarea prin conducte direct la firmă şi pe baza contoarelor fixate pe conducte să se facă plata apei. Evident, această soluţie nu este fezabilă! O soluţie uzualăde livrare este folosirea bidoanelor de capacitate rezonabilă dar relativ uşor de manevrat (să zicem bidoane de 10 litri). Fiecare client îşi cumpără câteva bidoane de la firmă. Pe măsură ce apa din bidoane se consumă, bidoanele goale se returnează la firmă şi se primesc înapoi pline.

Revenind în domeniul sistemelor de operare, această tehnică funcţionează după principiul problemei producător / consumator pe care am prezentat-o în 9.1.2.3. Pentru această tehnică, particularizările sunt următoarele:

- Producătorii şi consumatorii depind de contextul în care este aplicată metoda. Pentru exemplul de mai sus producătorul(ii) este(sunt) firma(ele) care îmbuteliază apa, iar consumatorii sunt clienţii.
- Un obiect (articol) care se produce / consumă este o zonă de memorie, un spaţiu disc, (un bidon în exemplul de mai sus) etc. Un astfel de obiect o să îl numim, generic, *zonă tampon.*
- Bufferul din problema producător / consumator este aici ansamblul a ***n*** zone tampon (numărul de bidoane de apă pe care le rulează un client) etc. O să numim acest buffer *pool de zone tampon*.
- Trecerea zonelor tampon de la producător la consumator se face după disciplina FIFO (first in – first out).

In plus faţă de problema producător / consumator, un tampon care urmează a fi depus în pool de către un producător o să îl numim *tampon (buffer) curent de intrare în pool,* iar tamponul extras de către un consumator îl vom numi *tampon (buffer) curent de ieşire din pool*. In

secţiunile următoare vom prezenta mai multe aplicaţii ale acestei tehnici în sistemele de operare.

11.1.2 Aplicaţii ale zonelor tampon temporare

Aplicaţiile care urmează sunt de fapt tehnici folosite mai ales la operaţii de intrare / ieşire şi la gestiunea memoriei.

11.1.2.1 Acces bufferizat la un fişier

Unitatea de schimb între un periferic magnetic şi memoria internă este *blocul*. Pentru suporturile de tip bandă magnetică dimensiunea unui bloc este stabilită de către cel care scrie informaţia pe bandă. Pentru suporturile de tip disc, unitatea de schimb este *sectorul*, pe care îl vom numi în continuare tot *bloc*.

Astfel, în modul cel mai simplu, un fişier îl putem privi ca şi o succesiune de blocuri consecutive, accesibile unul câte unul. Este deci nevoie ca în memorie să existe rezervată *o zonă tampon* în care să încapă orice bloc al fişierului.

Pentru optimizarea accesului la fişier, în memoria internă se rezervă spaţiu nu pentru o zonă tampon, ci un pool de 2 sau mai multe zone tampon. In fig. 11.1 este ilustrată situaţia la un moment dat a poolului de ***n*** zone tampon.

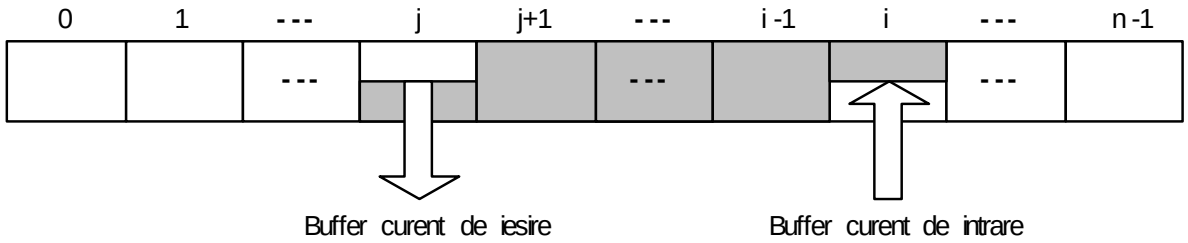


Figura 11.1 Un pool de zone tampon

La momentul curent, în pool sunt zone tampon neconsumate pe poziţiile j+1 până la i-1. Producătorul umple tamponul de pe poziţia i, iar consumatorul a extras ultimul tampon de pe poziţia j. Tratarea pool-ului se face circular: după ce s-a depus un tampon pe poziţia n-1, următorul în care se va depune va fi cel de pe poziţia 0. Pentru extragere se respectă, de asemenea, ordinea circulară. Evident, în conformitate cu principiul problemei producător / consumator, extragerea nu poate depăşi introducerea.

Producătorul şi consumatorul depind de modul de tratare a fişierului, dacă se scrie în el sau dacă se citeşte din el.

Dacă se citeşte din fişier, atunci producătorul este perifericul extern, iar consumatorul este procesul care consultă aceste buffere. La momentul deschiderii fişierului sunt umplute cu informaţii de pe suport toate bufferele din pool. Apoi, pe măsură ce se goleşte câte unul, acesta se umple cu blocul care urmează.

Dacă se scrie în fişier, atunci producătorul este procesul care creează informaţiile care se vor depune în fişier, iar consumatorul este perifericul extern unde se depun blocurile. Pe măsură ce un buffer din pool se umple, acesta este scris pe suportul extern. La terminarea scrierilor de către proces către fişier, toate zonele tampon încă nescrise se vor scrie pe suport.

11.1.2.2 Intreţinerea unui cache disc

Majoritatea sistemelor de operare întreţin după acelaşi principiu pool-uri (cache) disc. In cadrul nivelului fizic de intrare / ieşire, pentru a se optimiza accesul, se întreţine un pool cu copii ale celor mai recente sectoare disc accesate. Vom numi acest pool *cache disc.*

In momentul în care sistemul solicită accesul la un sector disc, se caută mai întâi o copie a lui în cache-ul disc. Dacă găseşte copia, atunci sistemul este servit cu copia şi accesul se consideră încheiat (fără ca de fapt să se fi accesat discul). Dacă se doreşte modificarea conţinutului unui sector, aceasta se face tot în cache şi se marchează sectorul ca fiind modificat.

Cache-ul are implementată o politică de evacuare din cache atunci când nu mai este spaţiu, politică similară cu cele de înlocuire a paginilor în memoria virtuală (NRU, LRU, FIFO etc).

In acest caz producătorul şi consumatorul pot fi, alternativ fie procesul fie perifericul, depinde de operaţia efectuată.

Trebuie menţionat faptul că accesul bufferizat şi cache-ul disc pot fi ambele operaţionale. In acest caz, pentru accesul bufferizat partenerul suport extern disc este înlocuit cu cache-ul disc.

11.1.2.3 Intreţinerea unui cache Web

Navigatoarele Web întreţin, după acelaşi principiu, pool-uri cu paginile web cele mai recent folosite. Mecanismul este similar cu cel al cache-ului de discuri, numai că aici o zonă tampon este de fapt un fişier cu pagina web depozitată în cache.

Producătorii sunt aici furnizorii de pagini web din Internet, iar consumatorul este browserul.

11.1.2.4 Conectarea pipe între două comenzi

Aşa cum am arătat în 1.1.1 şi 6.1, fiecare comandă a unui sistem de operare are un fişier standard de intrare şi unul standard de ieşire. Conexiunea pipe înseamnă conectarea unei ieşiri standard ca intrare standard a comenzii următoare. O astfel de conectare se specifică:

```
comanda1 | comanda2
```

Pentru acest gen de conexiune se creează, de regulă, un pool de zone tampon ca cel din fig. 11.1. Producătorul în acest caz este comanda1 (care furnizează ieşirea standard), iar consumatorul este comanda2 (care cere ieşirea standard).

In mod curent `comanda1` îşi scrie liniile în bufferul curent de intrare, iar `comanda2` îşi ia liniile de intrare standard din bufferul curent de ieşire (vezi fig . 5.34 şi 2.2.2.2).

11.1.2.5 Memoria cache

Memoria cache, în relaţia ei cu memoria internă (vezi 10.1.2) se comportă ca şi un pool de copii ale unor locaţii din memoria internă.

In această situaţie, atât producătorul, cât şi consumatorul este memoria internă.

11.2 Canalul de intrare / ieşire

11.2.1.1 Canalul şi interacţiunea lui cu procesorul central

In sistemele seriale simple, **procesorul central** aşteaptă în general mult. Acest fapt se petrece din cauza diferenţelor mari de viteză dintre echipamentele electronice ale procesorului şi echipamentele, preponderent mecanice, ale dispozitivelor de I/O. Spre exemplu, un cititor rapid de cartele citeşte 300 de cartele/minut. Un compilator traduce un program de 300 de instrucţiuni în 6 secunde. De aici reiese că procesorul este inactiv aproximativ 90% din timpul total necesar compilării. Se impune (de fapt se impunea) creşterea eficienţei în exploatare a procesorului.

Dezvoltarea arhitecturii a condus la apariţia, odată cu generaţia a II-a de calculatoare, a *canalului de intrare - ieşire*. Acesta este un procesor specializat pe operaţii de I/O care poate funcţiona în paralel cu procesorul central. Pentru a fi lansat în execuţie, canalul primeşte de la procesor o comandă de efectuare a unei operaţii de I/O. După lansare, cele două procesoare îşi continuă activitatea în paralel. La sfârşitul operaţiei de I/O, canalul anunţă acest fapt procesorului central prin intermediul unei *întreruperi*.

Să detaliem relaţia dintre procesorul central şi un dispozitiv periferic:

- Fiecare periferic conţine o zonă tampon proprie, capabilă să păstreze o înregistrare (o linie de imprimantă, imaginea unei cartele, un sector disc etc.).
- Procesorul, printr-o rutină de I/O, acţionează dual, în funcţie de operaţia efectuată. In cazul scrierii, pune din memorie informaţii în această zonă tampon, iar în cazul citirii preia informaţiile din ea şi le depune în memorie.
- Dispozitivul periferic acţionează şi el dual, în funcţie de operaţia care îi este comandată. In cazul scrierii, ia informaţiile din zona tampon proprie şi le depune pe suport. In cazul citirii, ia informaţiile de pe suport şi le depune în zona tampon proprie.
- Viteza cu care lucrează procesorul este mult mai mare decât viteza dispozitivului periferic.

Fig. 11.2 [4] ilustrează cele trei variante fundamentale de interacţiune între procesor (CPU) şi un dispozitiv periferic, la două operaţii succesive de scriere. Cu linie îngroşată este figurat timpul de acţiune al rutinei de I/O, iar cu linie punctată timpul de aşteptare al procesorului.

Cazul a). *aşteptare reciprocă*. In fiecare moment, funcţionează numai un singur procesor: sau procesorul central, sau canalul de I/O. Evident că această variantă de schimb este ineficientă (după cum reiese şi din exemplul dat mai sus, cu citirea şi compilarea unui program de 300 de linii).

Cazul b). *polling efectuat de către* procesorul central. acesta foloseşte o instrucţiune maşină de test asupra perifericului, pe care o execută periodic. Dacă aceasta răspunde că perifericul este liber, atunci procesorul lansează o nouă comandă de I/O. După lansare, procesorul şi perifericul lucrează în paralel. Tehnica sondării din când în când a stării perifericelor poartă numele de *polling*. Frecvenţa cu care se face testarea este principala caracteristică a pollingului. Dacă frecvenţa este prea mică, atunci este posibil ca perifericul să fie utilizat ineficient. In schimb, o frecvenţă prea mare duce la creşterea nejustificată a timpului destinat doar testării, încărcând timpul de regie al **SO**. Tehnica polling este folosită de regulă în cazul **SC** interactive multiutilizator, pentru legătura cu terminalele de teletransmisie.

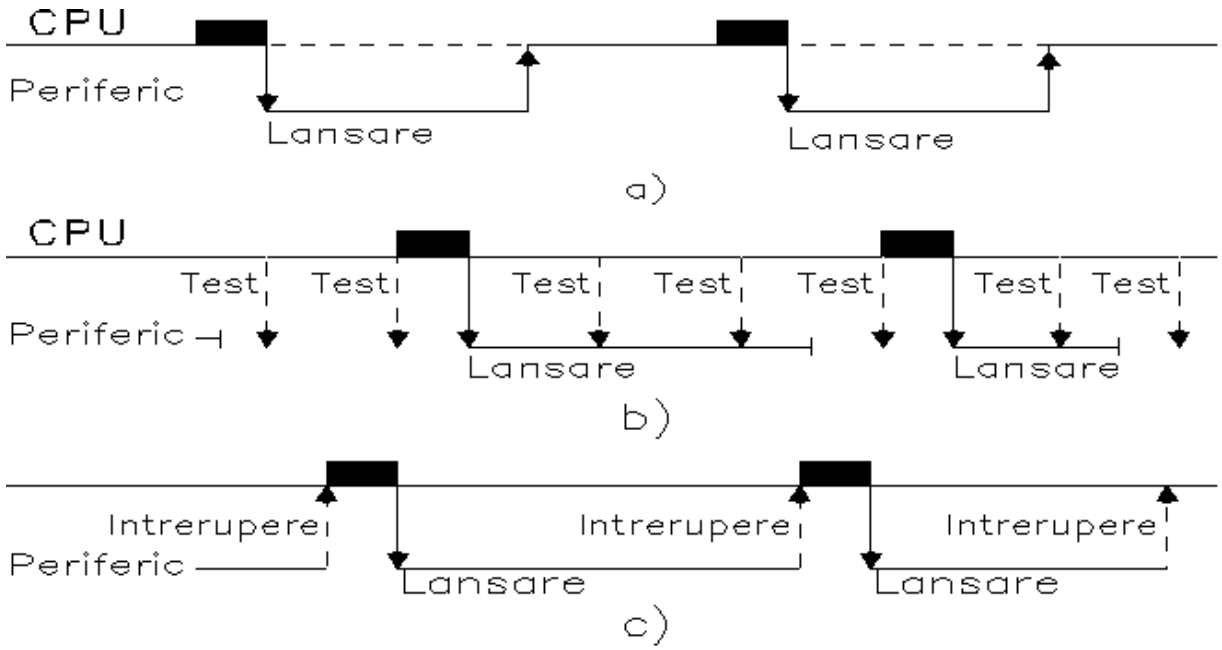


Figura 11.2 Interacţiuni dintre procesorul central şi canalul de intrare / ieşire

Cazul c) *întrerupere lansată de către periferic*. Prin intermediul întreruperilor se realizează comunicarea dintre **SC** şi dispozitivele periferice legate la el (discuri, benzi, imprimante etc.). Rutina de întrerupere determină perifericul care a emis întreruperea, execută rutina de I/O aferentă, după care, eventual, dă perifericului o nouă comandă de I/O.

Cazurile b) şi c) din fig. 11.2 pun în evidenţă situaţii în care procesorul şi canalul de I/O pot lucra în paralel. Pentru aceasta vom defini două funcţii cu care procesorul central se adresează canalului de intrare / ieşire:

```
STARTIO(operatie, periferic, adresa_de_memorie)
WAITIO(adresa_de_memorie)
```

`STARTIO` lansează o operaţie de intrare/ieşire (**Read** sau **Write**) de schimb a unui bloc de informaţie între un dispozitiv periferic şi o zonă de memorie. In cazul **Read** se depun octeţii blocului citiţi de la `periferic` în zona tampon indicată prin `adresa_de_memorie`. In cazul **Write** se iau octeţii blocului din zona tampon indicată de `adresa_de_memorie` şi sunt depuşi în blocul indicat de `periferic`.

`WAITIO` aşteaptă terminarea operaţiei lansate cu `adresa_de_memorie` indicată. Momentul în care `WAITIO` se termină este stabilit fie de unul din testele lansate de procesor (cazul b) din fig. 11.2), respectiv de către handlerul de întrerupere (cazul c) din fig. 11.2). Este unanim acceptat faptul că în timpul schimbului unui bloc între memoria internă şi un periferic,

atât *zona tampon cât şi spaţiul periferiferic rezervat blocului pe periferic sunt atribiute în mod exclusiv procesului de schimb*. Din această cauză pentru `WAITIO` este suficient arumentul `adresa_de_memorie`.

In prezent, **SC** folosesc două tipuri de canale: *canal selector* şi *canal multiplexor*. Canalul selector este destinat să realizeze schimbul dintre memorie şi perifericele rapide (discuri, benzi), care lucrează, la un moment dat, cu un singur periferic. Canalul multiplexor este capabil să lucreze simultan cu mai multe dispozitive periferice. Construcţia acestuia este mai complicată decât cea a canalului selector. Multiplexorul trebuie să decidă, pentru fiecare octet schimbat, care este traseul acestuia între memorie şi perifericele legate la multiplexor.

Apariţia canalelor de I/O a permis abordarea unor tehnici de exploatare eficientă a procesorului. Următoarele trei tehnici se bazează pe acest paralelism:

- utilizarea zonelor tampon multiple;
- multiprogramarea (vezi 9.2)
- conversiile off-line şi tehnica SPOOLING (vezi [10])

11.2.1.2 Utilizarea zonelor tampon multiple.

Folosirea *zonelor tampon* în diverse contexte a fost deja prezentată în 11.1.1. In cele ce urmează vom prezenta o nouă aplicaţie a acestei tehnici.

Presupunem că un program are de efectuat o serie de citiri de pe un periferic **`Dk:`**, iar rezultatul fiecărei citiri este folosit în calcule. Pentru o mai uşoară specificare, vom folosi două funcţii:

- `CALCULE(adresa_de_memorie)` indică procesorului central să efectueze calculele necesare cu informaţiile luate din zona tampon indicată de `adresa_de_memorie`.
- `EOF` este o funcţie booleană care întoarce valoarea `TRUE` s-au terminat toate citirile .

Pentru lucrul cu canalul de intrare / ieşire se vor folosi funcţiile `STARTIO` şi `WAITIO` descrise în secţiunea precedentă. In fig. 11.3 este dată prima soluţie a problemei de mai sus.

```
do {
  STARTIO('Read', 'Dk:', ZT);
  WAITIO(ZT);
  if (EOF) break;
  CALCULE(ZT);
} while (true);
STOP:
```

Figura 11.3 O primă soluţie a citirilor multiple

Programul din fig. 11.3 foloseşte o singură zonă tampon, `ZT` în care mai întâi citeşte şi apoi ia din ea informaţii pentru calcule. Evident, această rezolvare nu se exploatează deloc paralelismul. După caz, fie procesorul central aşteaptă după canal, fie canalul aşteaptă după procesorul central.

Pentru a exploata acest paralelism, cea mai simplă soluţie este folosirea a două zone tampon, `ZT1` şi `ZT2`. Astfel, după ce s-a efectuat o citire într-o zonă tampon, procesorul cere imediat canalului să efectueze, în avans încă o citire, în cealaltă zonă tampon. După lansarea noii citiri, procesorul trece la prelucrarea informaţiilor obţinute din prima citire. In momentul în care

canalul I/O şi-a încheiat noua misiune, iar procesorul solicită noi date, acestea sunt deja disponibile în a doua zonă tampon. Se lansează apoi în avans o nouă citire ş.a.m.d. In fig. 11.4 este ilustrat acest procedeu.

```
STARTIO('Read', 'Dk:', ZT1);
do {
  STARTIO('Read', 'Dk:', ZT2)
  WAITIO(ZT1);
  if (EOF) break;
  CALCULE(ZT1);
  STARTIO('Read', 'Dk:', ZT1);
  WAITIO(ZT2);
  if (EOF) break;
  CALCULE(ZT2)
} whle (FALSE);
STOP:
```

Figura 11.4 Citiri multiple cu două zone tampon

In mod analog, se petrec lucrurile dacă se cer efectuarea repetată a unor calcule urmate de scrieri pe periferic. Procedeul este ilustrat în fig. 11.5.

```
CALCULE(ZT1);
STARTIO('Write', 'Dk:', ZT1);
do {
  CALCULE(ZT2);
  if (EOF) break;
  STARTIO('Write', 'Dk:', ZT2);
  WAITIO(ZT1);
  CALCULE(ZT1);
  if (EOF) break;
  STARTIO('Write', 'Dk:', ZT2);
  WAITIO(ZT2);
} while (FALSE);
STOP:
```

Figura 11.5 Scrieri multiple cu două zone tampon

In fig. 11.5 funcţia EOF indică terminarea programului.

Extinderea procedeelor de mai sus la *n* zone tampon este relativ simplă. Zonele tampon se indexează de la 0 la n-1 şi se manevrează în manieră circulară. Lăsăm pe seama cititorului acest exerciţiu.

Controlul zonelor tampon multiple de către utilizator nu este o soluţie satisfăcătoare. Este evident că manipularea zonelor tampon multiple presupune multă atenţie. O manipulare neglijentă este în mod precis o cauză foarte ascunsă care poate provoca apariţia unei erori în funcţionarea programului. Recomandăm utilizarea acestei tehnici numai de către programatorii de elită şi numai în situaţii care reclamă în mod expres folosirea lor. In prezent, rutinele I/O din toate **SO** moderne au încorporate în ele puternice facilităţi de lucru cu zone tampon multiple. De exemplu, componenta SGF (gestiunea fişierelor) tratează schimbul cu fişierele prin intermediul zonelor tampon multiple aşa cum s-a văzut în 11.1.1. Astfel, utilizatorul beneficiază totuşi de facilităţile utilizării zonelor tampon multiple, dar cu sprijinul **SO.**

11.3 Elemente specifice lucrului cu discul

Dată fiind importanţa perifericelor de tip disc, în cle ce urmează vom aborda câteva elemente specifice nivelului fizic al intrărilor / ieşirilor.

11.3.1 Partiţionarea unui hard disc

Spaţiul unui hard disc poate fi împărţit în mai multe zone contigue, numite *partiţii disc*. O astfel de partiţie conţine un număr oarecare de cilindri consecutivi şi este privită de sistemul de operare ca şî un disc separat. . SO MS_DOS priveşte (începând cu versiunea 3.30) fiecare astfel de partiţie ca pe un disc separat. Este posibil, de asemenea, ca două partiţii diferite ale aceluiaşi hard disc să găzduiască două sisteme de operare diferite. De exemplu, una să găzduiască Windows XP, iar cealaltă o variantă de Linux.

De exemplu, să presupunem că avem un hard disc care are 614 cilindri. Acesta poate fi împărţit, din considerente organizatorice, în două partiţii astfel. O primă partiţie de 29 de cilindri va fi destinată fişierelor sistem importante, cum ar fi cele care conţin comenzile de bază ale SO. Ea va ocupa cilindrii 0 la 28 şi va putea fi supusă unui regim special de protecţie antivirus. A doua partiţie, de 585 cilindri, este destinată restului fişierelor. Ea va ocupa cilindrii 29 la 613 şi va putea fi folosită fără restricţii. (Evident, acesta este numai un exemplu, fiecare utilizator putând să-şi împartă hard discul cum doreşte). In exemplul de mai sus, SO va privi hard discul ca două discuri logice.

Operaţia de partiţionare se realizează cu ajutorul unui program specializat, numit FDISK. Există şi alte programe, cu interfeţe grafice realizează partiţionarea unui hard disc.

Este important de reţinut că operaţia de partiţionare are ca şi consecinţă pierderea tuturor informaţiilor de pe hard discul respectiv! Drept urmare, înainte de partiţionare vor fi salvate toate fişierele care se doresc a fi păstrate. Evident, acţiunea de partiţionare trebuie efectuată numai de către persoanele care posedă cunoştinţele necesare pentru partiţionare.

Să vedem cum este implementat mecanismul de partiţionare. Primul sector al hard discului (sectorul 1 de pe faţa 0 a cilindrului 0) poartă numele de sectorul de încărcare absolut sau sectorul MBR (Master Boot Record). El conţine un program de tip bootstrap şi o tabelă de descriere a partiţionării. O intrare pentru descrierea unei partiţii are structura din fig.

B	Hi	S+Ci	Sy	Hf	S+Cf	Sectreli	Sectrelf	
00	01	02	04	05	06	08	0C	0F

Figura 11.6 Descrierea unei partiţii hard disc

B este octetul de indicare a partiţiei active. Valoarea 80(hexa) indică o partiţie activă, iar valoarea 00 indică o partiţie inactivă. In orice moment, pe un hard disc trebuie să existe o singură partiţie activă.

Hi şi **S+Ci** indică adresa fizică de început a partiţiei. Se indică numărul feţei pe primul octet, iar pe următorii doi numărul sectorului şi al cilindrului de început.

Hf şi **S+Cf** indică, analog cu cele de mai sus, adresa fizică de sfârşit a partiţiei.

`Sectreli` şi **`Sectrelf`** indică numărul sectorului de început, respectiv al sectorului de sfârşit a partiţiei.

Cum are loc încărcarea unui SO aflat într-o partiţie activă? In momentul iniţializării, BIOS încarcă în memorie sectorul MBR şi dă controlul programului bootstrap de partiţionare. Acesta citeşte tabela de partiţii şi o caută pe cea activă. Având adresa fizică de început a acesteia, citeşte primul sector al ei. In acest sector se află programul bootstrap de lansare a încărcării SO respectiv.

Fiecare SO aflat pe hard disc dispune de un program similar lui `FDISK`. Prin intermediul lui se poate dezactiva o partiţie şi activa alta. Astfel, la o nouă încărcare, poate intra în lucru un alt SO.

Observaţie: Deşi structura tabelei de partiţii este cunoscută, nu este indicat să se acţioneze asupra ei decât cu programele oferite de firma constructoare a hard discului. Fiecare program bootstrap de partiţionare are anumite particularităţi care depind de construcţia hard discului. Intervenţia în această structură cu alte programe poate face discul inutilizabil [27].

11.3.2 Planificarea accesului la discul magnetic

11.3.2.1 Problematica planificării accesului la disc

Pentru a se avea acces la un anumit sector disc, unitatea de disc acţionează în trei etape:

- *poziţionarea* braţului pe cilindrul care conţine sectorul dorit;
- aşteptarea *rotaţiei* discului până când sectorul trece prin faţa capului de citire-scriere;
- *schimbul* propriu-zis de informaţie.

Dintre aceste etape, cea de poziţionare este cea mai lentă. La un moment dat, există o coadă de aşteptare pentru accese la disc, în special dacă discul este partajat între mai mulţi utilizatori. Ordinea de servire este decisă de către **SO**, astfel încât serviciile aceluiaşi utilizator să se facă în ordinea sosirii lor. Pe de altă parte, **SO** planifică serviciile între utilizatori astfel, încât timpul total necesar poziţionărilor să fie cât mai mic.

De asemenea, aşteptarea rotirii poate fi redusă folosind o serie de tehnici speciale.

Problema timpului necesar schimbului efectiv nu se pune, deoarece acesta este o constantă de construcţie a discului.

Pentru a avea o imagine asupra ordinelor de mărime şi a raportului între duratele etapelor, în tabelul următor prezentăm câteva caracteristici tehnice ale unui disc. Este vorba de un disc flexibil standard, dublă faţă şi dublă densitate.

Număr de cilindri:	40
Piste pe cilindru	2
Sectoare pe pistă	9
Sectoare pe dischetă	720
Octeţi pe sector	512
Octeţi pe dischetă	368640
Timp de poziţionare între doi cilindri vecini	6 msec

Timp de poziţionare în medie pe disc	77 msec
Durata unei rotaţii	200 msec
Timp pornire/oprire motor	250 msec
Durata schimbului: 1 sector	22 msec

O serie de îmbunătăţiri constructive ale discului aduc în mod implicit şi o reducere a timpului total de acces. Printre acestea amintim două.

- Multe drivere de disc au posibilitatea de a gestiona simultan mai multe discuri. Pe baza acestei facilităţi, se realizează *suprapunerea operaţiilor de poziţionare* la mai multe discuri simultan. Unele drivere extind această suprapunere şi la celelalte etape ale accesului la disc.
- Este ştiut faptul că sectoarele de pe pistele exterioare sunt mai lungi decât cele de pe pistele interioare, dar volumul de informaţie este acelaşi. Tehnologiile actuale, aplicate mai ales la hard-discuri, exploatează această diferenţă de lungime, folosind aşa-numita *Write Precompensation*, în vederea creşterii performanţelor.

Din punctul de vedere al planificării **SO**, sunt importante metodele de reducere, specifice planificării, a timpului de aşteptare a rotaţiei, dar mai ales a timpului de poziţionare. Evident, aceste probleme au rost numai la *sistemele multiutilizator*. Cererile de acces la disc ale unor utilizatori diferiţi pot fi puţin reordonate, dacă prin aceasta creşte cât de cât randamentul discului respectiv.

11.3.2.2 Reducerea accesului la sectoare vecine

Sectorul este unitatea de adresare a informaţiei pe disc. Orice bloc de informaţie este garnisit cu informaţii de control de paritate şi checksum, pentru a controla stabilitatea informaţiei din sector. Blocul este compus din unul sau mai multe sectoare vecine, eventual situate în aceeaşi pistă (pentru unele **SC**) sau în acelaşi cilindru (pentru alte **SC**).

Două sectoare sunt *vecine*, dacă adresele lor disc diferă printr-o unitate. Numerotarea sectoarelor în cadrul unei piste se face în două moduri: *fără întreţesere* şi *cu întreţesere (interleaving)*. In fig. 11.7 sunt date trei exemple de numerotare a sectoarelor de pe o pistă care conţine 8 sectoare:

- numerotare fără întreţesere, fig. 11.7a;
- numerotare cu factorul de întreţesere 1, fig. 11.7b;
- numerotare cu factorul de întreţesere 2, fig. 11.7c. (exemple preluate din [48]).

Numerotarea fără întreţesere are avantajul simplităţii în dauna timpului de acces. Numerotarea cu întreţesere favorizează prelucrările secvenţiale, permiţând trecerea de la un sector la sectorul vecin în cadrul aceleiaşi rotaţii. Aceast lucru este posibil deoarece, după cum ştim, timpul de acces propriu-zis este compus din timpul efectiv de scriere/citire, plus un *timp de regie* a schimbului. Acesta din urmă se referă la timpul necesar completării blocului cu informaţiile de control al parităţii sau de verificare a acestora, precum şi a altor pregătiri electronice ale unui schimb.

De exemplu, pentru a citi 8 sectoare de pe pistă din fig. 11.7a sunt necesare 8 rotaţii de disc. Dacă timpul de regie necesar unui schimb este mai mic decât timpul de trecere a unui sector prin faţa capului de citire, atunci pentru numerotarea din fig. 11.7b sunt necesare doar două rotaţii! Dacă timpul de regie nu depăşeşte timpul de trecere a două sectoare, atunci pentru

numerotarea din fig. 11.7c sunt necesare doar trei rotaţii! Acest mecanism de întreţesere este gestionat uneori prin hard, alteori prin intermediul unor drivere soft.

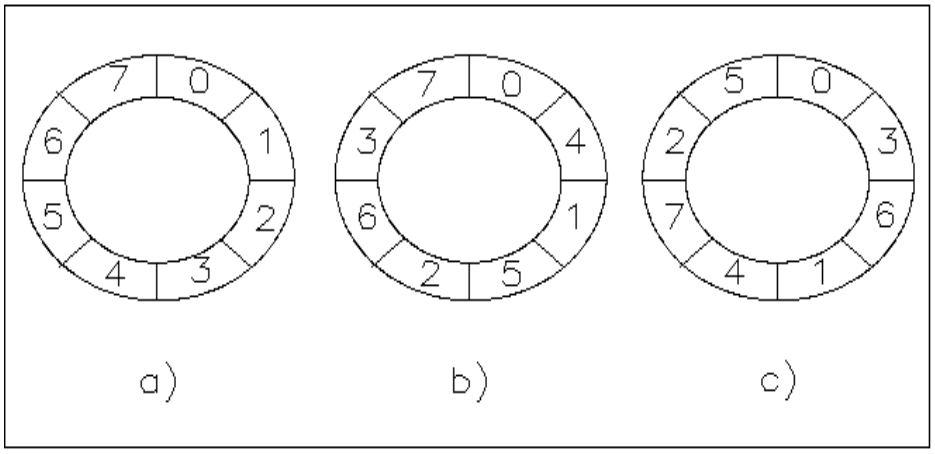


Figura 11.7 Numerotări ale sectoarelor, cu şi fără interleaving

11.3.2.3 Reducerea aşteptării rotaţiei

In **SO** mai pretenţioase, sau când este vorba de discuri cu capete fixe, se pune problema scurtării timpului de aşteptare pentru momentele când sectoarele defilează prin faţa capetelor. Apare astfel problema de ordonare a momentelor de lansare a transferurilor, pentru a se efectua toate schimburile solicitate, cu un număr redus de rotaţii ale discului.

Reducerea timpului de aşteptare a rotaţiei se poate face şi ordonând adecvat servirile cererilor existente la un moment dat pentru acelaşi cilindru. Cel mai răspândit algoritm de acest fel este algoritmul SLTF (Short Latency Time First) care funcţionează după regula "cel ce aşteaptă cel mai puţin va fi servit primul". Să luăm ca exemplu cele trei cereri din fig. 11.8.

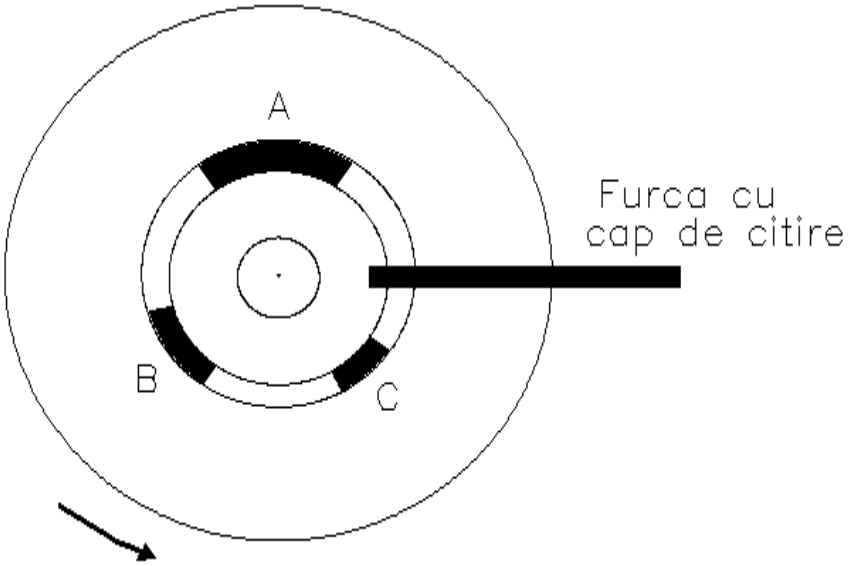


Figura 11.8 Cereri de sectoare de pe aceeaşi pistă

In exemplul din fig. 11.8 sunt ilustrate trei cereri la sectoare de pe aceeaşi pistă. Sosirea în timp a cererilor a 270 a fost:

A, B, C

Dacă servirea se face în această ordine, atunci cele trei cereri sunt terminate după trei rotiri ale discului.

Algoritmul SLTF va programa servirea în ordinea:

C, B, A

după timpul pe care îl mai au până la trecerea prin faţa capului de citire. Drept rezultat, cele trei cereri sunt rezolvate într-o singură rotire a discului.

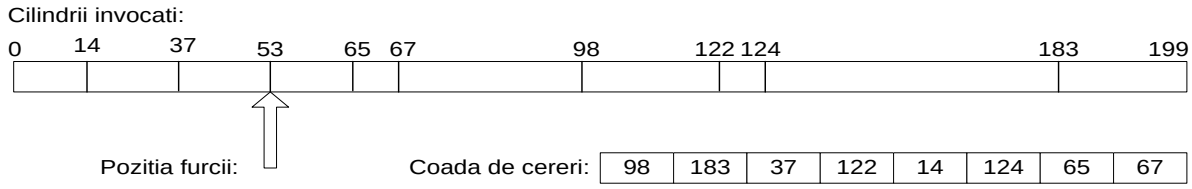
11.3.2.4 Reducerea timpului de poziţionare

Există mai mulţi algoritmi de planificare a cererilor de accese la diverşi cilindri disc. Alegerea pentru implementare în **SO** a unui anumit algoritm este însă o problemă dificilă. Trebuie spus din capul locului că o serie de elemente care nu ţin efectiv de planificare pot influenţa randamentul discului. Iată două exemple:

- Structura informaţiilor pe disc influenţează serios timpii de poziţionare. Pentru accesul la un fişier secvenţial, alocat într-o zonă contiguă pe disc, este necesar mult mai puţin timp decât pentru accesul la un fişier secvenţial indexat, unde trebuie accesate şi tebele de indexi de pe disc.
- La **SO** interactive, eventual cu multiacces, apar multe fişiere, în general de dimensiuni mici. Aici sunt necesare multe accese la directorii de fişiere (în principal pentru deschiderea şi închiderea acestor fişiere). Plasând, spre exemplu, directorii în mijlocul discului (nu la început şi nici la sfârşit), se poate obţine o reducere substanţială a timpului de poziţionare.

Revenind acum la planificarea propriu-zisă a poziţionărilor, să prezentăm câţiva algoritmi mai des folosiţi. Pentru a-i ilustra, să considerăm situaţia de cereri din fig. 11.9. La un moment dat, furca cu capetele de citire-scriere este poziţionată pe cilindrul 53 şi există opt cereri de poziţionare, în ordinea solicitărilor, pe următorii cilindri: 98, 183, 37, 122, 14, 124, 65, 67.

Figura 11.9 Cereri la disc la un moment dat



Următorii patru algoritmi sunt cel mai mult folosiţi pentru scurtarea timpului total pentru poziţionări.

Algoritmul FCFS (First Come First Served - primul venit, primul servit), execută cererile în ordinea sosirii acestora. Este un algoritm simplu, dar nu cel mai eficient. Pentru exemplu de mai sus, servind cererile aşa cum sunt ele în coadă, sunt necesare mişcări de poziţionare însumând 640 *cilindri.*

Algoritmul SSTF (Shortest Seek Time First), execută de fiecare dată operaţia care solicită cea mai scurtă poziţionare, faţă de locul curent. In exemplul nostru, ordinea de servire este:

65, 67, 37, 14, 98, 122, 124, 183

Această ordine necesită poziţionări peste numai 236 *cilindri*. Comparaţia acestui rezultat cu cel dat de algoritmul FCFS dă câştig de cauză algoritmului SSTF. Dezavantajul algoritmului este că, teoretic, poate amâna indefinit anumite cereri, deoarece coada de aşteptare se modifică permanent! Dacă, spre exemplu, după servirea de la cilindrul 14 apar numai cereri până la cilindrul 97, ultimele 4 cereri aşteaptă indefinit!

Algoritmul SCAN (elevator). Pentru acest algoritm, mişcarea capetelor începe de la ultimul cilindru, mergând spre primul şi servind toate cererile pe care le întâlneşte. După ce a servit cererea de la cea mai mică adresă de cilindru solicitat, capetele îşi schimbă sensul de mers, servind cererile apă- rute ulterior până la servirea cererii de pe cilindrul solicitat cu cea mai mare adresă, apoi iar se schimbă sensul ş.a.m.d. Dacă în exemplu nostru, servirea cilindrului 53 s-a făcut la mişcarea spre adrese mici, atunci servirea se face în ordinea:

```
37, 14, 65, 67, 98, 122, 124, 183
```

Ca urmare, vor rezulta deplasări totale de 192 *cilindri.* Este posibil ca unele cereri să aştepte două parcurgeri ale discului până sunt servite. Pentru a se uniformiza timpii de aşteptare în vederea servirii, se foloseşte varianta circulară, pe care o prezentăm în continuare.

Algoritmul C-SCAN - evaluare circulară. Spre deosebire de algoritmul SCAN, aici servirea se face numai când capetele avansează de la adrese mici spre cele mari. In momentul în care s-a ajuns la ultimul cilindru, capetele sunt retrase automat pe cilindrul 0 (de regulă cu mare viteză), după care se reîncep servirile apărute în coadă. In exemplul considerat, ordinea de servire este:

```
65, 67, 98, 122, 124, 183; 199;  retragere  la 0; 14, 37
```

La unele tipuri de unităţi de disc, retragerea capetelor pe cilindrul O durează mai puţin decât o poziţionare obişnuită. In acest caz, se poate considera că, practic, fiecare cerere este servită *cel mult după parcurgerea în întregime a cilindrilor discului.*