Gestiunea memoriei

Introducere. În conformitate cu arhitectura von Neumann, memoria primară împreună cu regiştrii CPU şi memoria „cache” formează **memorie executabilă**, deoarece aceste componente sunt implicate în execuţia unui proces.

CPU poate încărca instrucţiunile acestuia numai din memoria primară.

Unităţile de memorie externă (secundară) sunt utilizate pentru a stoca date pentru o mai lungă perioadă de timp. Fişierele executabile, pentru a deveni procese, precum şi informaţiile prelucrate de acestea, trebuie să fie încărcate în memoria primară.

Aministratorului memoriei interne este responsabil de alocarea memoriei primare proceselor şi de a acorda asistenţă programatorului în încărcarea/salvarea informaţiilor din/în memoria secundară.

Astfel, partajrea memoriei interne de către mai multe procese şi minimizarea timpului de acces sunt obiective de bază ale administratorului memorie interne.

Spaţiul de adrese al unui proces

- Componentele unui program sursă sunt reprezentate folosind identificatori, etichete şi variabile, ce reprezintă nişte nume simbolice şi formează **spaţiul de nume** al programului sursă.
- **Faza de compilare** transformă un text sursă într-un modul obiect, adică fiecare nume simbolic este translatat într-o adresă relativă la modulul obiect.
- **Faza de editare de legături** grupează mai multe module, formând un fişier, numit modul absolut, stocat pe un suport extern până când se cere execuţia lui. **Editorului de legături** transformă adresele din cadrul modulelor în aşa-zisele **adrese relocabile.**
- **Faza de translatare (relocare)** a adresei constă în transformarea adreselor relative la fişierul executabil, în adrese de memoria internă, realizându-se astfel **imaginea executabilă** a programului. Acest lucru este realizat de o componentă a SO, numită **încărcător**(loader).
- **Spaţiul de adrese al unui proces** este mulţimea locaţiilor alocate acestuia, atât din memoria primară, cât şi din cea secundară, servicii ale SO şi resurse.

- Altor obiecte referenţiate de către un program le sunt asociate adrese de memorie internă. Spaţiul de adrese defineşte toate entităţile logice folosite de către un proces şi specifică o adresă prin care ele sunt referenţiate.
- Un program poate fi gândit ca o specificare a unei activităţi(algoritm) care urmează să fie realizată de către un proces. El conţine un set de instrucţiuni care urmează să fie executate şi o mulţime de variabile pe care le utilizează. Când un program sub formă de fişier executabil este gata de execuţie, se realizează o corespondenţă care defineşte unde vor fi plasate procedurile şi datele în spaţiul de adrese al procesului.

Încărcarea programului

- Înainte ca un program să fie executat, trebuie să-i fie alocat un spaţiu din memoria primară. Dacă se lucrează în regim de multiprogramare, este posibil ca în memorie să fie prezente simultan mai multe programe. Fiecare program foloseşte zona(zonele) de memorie alocată(alocate) lui.
- De asemenea, pe durata execuţiei unui program, necesarul de memorie variază. Odată ce sistemul cunoaşte care locaţii de memorie urmează a fi folosite pentru execuţia programului, poate să realizeze corespondenţa dintre adresele de memorie primară alocate procesului respectiv şi spaţiul său de adrese.
- Deci programul executabil este translatat într-o formă finală accesibilă unităţii de control a CPU şi încărcat în memoria primară, la o anumită adresă de memorie.
- Când **contorul de program** este iniţializat cu adresa primei instrucţiuni executabile(principalul punct de intrare din program), CPU începe să execute programul.

- În anumite SO, administratorul memoriei poate să şteargă o parte din programul executabil din memoria primară, să-l salveze în memoria secundară şi să elibereze zona de memorie ocupată, astfel încât aceasta să poată fi alocată altor procese. Astfel, chiar după ce un program a fost convertit într-o formă executabilă în memoria internă, el poate fi stocat şi în memoria externă, atâta timp cât este necesar.
- Totuşi, odată cu execuţia procesului, imaginea sa din memoria internă se schimbă, fără a se modifica şi copia existentă pe disc.
- De exemplu, copia unei date din memoria secundară va fi modificată numai dacă procesul execută o comandă de scriere, prin care în mod explicit modifică zona de pe disc ocupată de data respectivă.

Funcţiile administratorului; obiective ale gestiunii memoriei

Funcţiile administratorului

- Alocarea de spaţiu de memorie internă proceselor;
- Realizarea corespondenţei dintre spaţiul de adrese al procesului şi locaţii de memorie internă;
- Minimizarea timpului de acces la locaţiile de memorie.

- Realizarea acestor funcţii este condiţionată atât de componenta hardware, cât şi de cea software, conţinută în SO. Odată cu evoluţia componentelor hardware ale SC, s-au schimbat şi strategiile de administrare a memoriei, pentru a se valorifica aceste îmbunătăţiri. Reciproc, strategiile privind gestiunea memoriei au evoluat în timp, ceea ce a condus la evoluţia componentelor hardware ale sistemelor de calcul.
- Principalele **obiective** ale gestiunii memoriei sunt:

- calculul de translatare a adresei(relocare);
- protecţia memoriei;
- organizarea şi alocarea memoriei operative;
- gestiunea memoriei secundare;
- politici de schimb între procese.

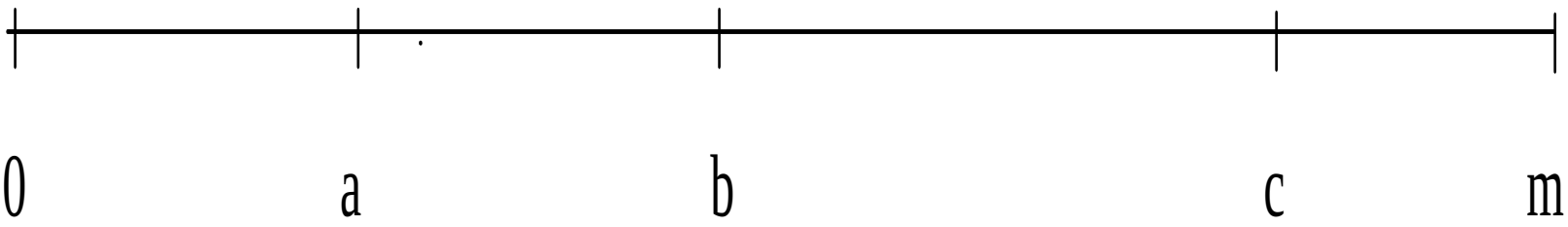
Metode clasice de alocare a memoriei

- În cazul sistemelor cu **prelucrare în loturi** (**monoprogramare**), spaţiul de memorie internă este împărţit în trei zone (figura urm):

  - o zonă este alocată nucleului sistemului de operare (spaţiul de adrese `0,.., a-1`);

  - următoarea zonă este alocată job-ului în curs de execuţie (spaţiul de adrese `a, a+1,..,c-1`);

  - zona cuprinsă între adresele c şi `m-1`, reprezintă un spaţiu nefolosit (memoria are capacitatea de `m` locaţii).



- La un moment dat, există un singur job în execuţie, care are disponibil întreg spaţiul de memorie, care începe la adresa a.
- Această metodă este specifică generaţiei a II-a de calculatoare. Memoria internă a calculatoarelor din această generaţie avea o dimensiune mică.
- Gestiunea spaţiului de adrese a, ..,c-1 cade în sarcina utilizatorului:

  - pentru a-şi putea rula programele de dimensiune mare, el foloseşte **tehnici de suprapunere** (overlay).

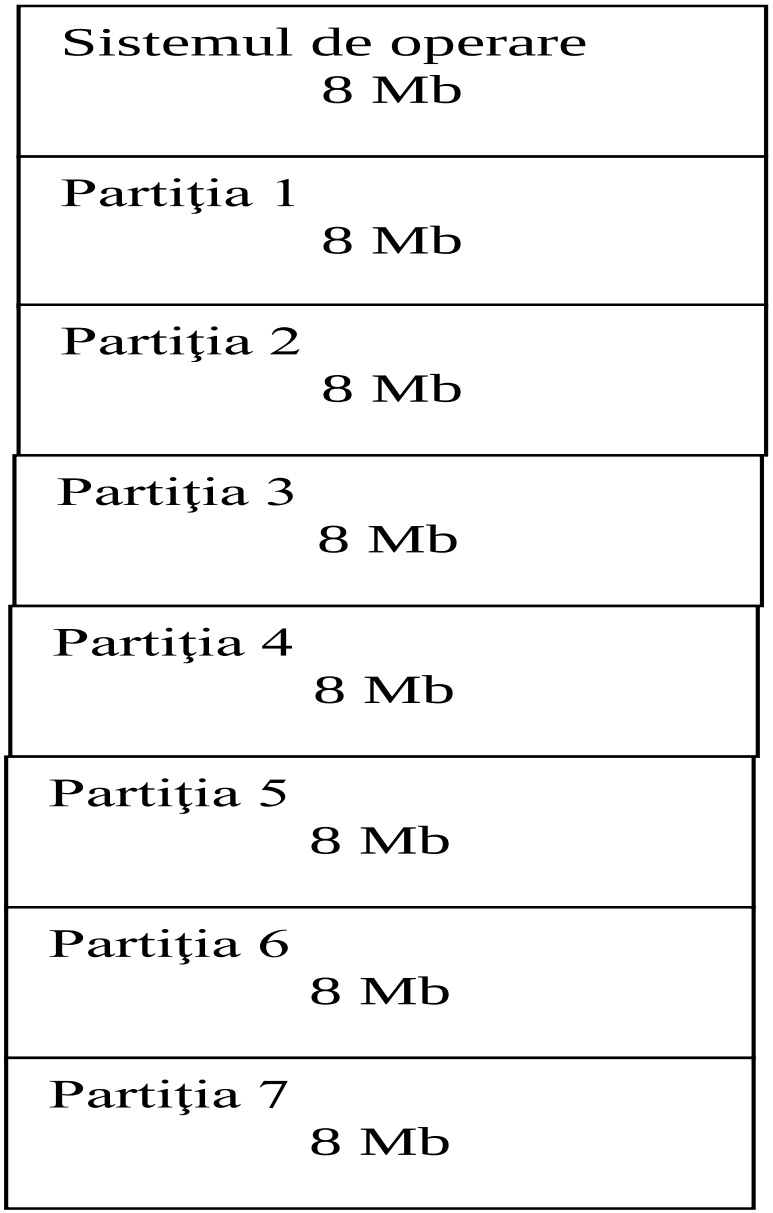
  - Orice program este format dintr-o **parte rezidentă**, care este prezentă în memorie pe întreaga durată a execuţiei (adresele a, ..,b-1) şi o parte de suprapunere (adresele b, ..,c-1), în care se aduc părţi ale programului (segmente), de care este nevoie la momentul respectiv.

  - Programatorul dispune de comenzi pentru definirea segmentelor.
- **Exemplu:** Să presupunem că avem un program care calculează suma şi produsul a două matrici precum şi norma matricilor. Partea rezidentă, va fi alocată unei funcţii principale în care se fac iniţializări (citirea componentelor matricilor), se memorează şi se afişează valorile obţinute. Vom avea câte un modul pentru cele trei funcţii carecalculează valorile indicate. La un moment dat, numai unul dintre module se poate afla în partea rezidentă.

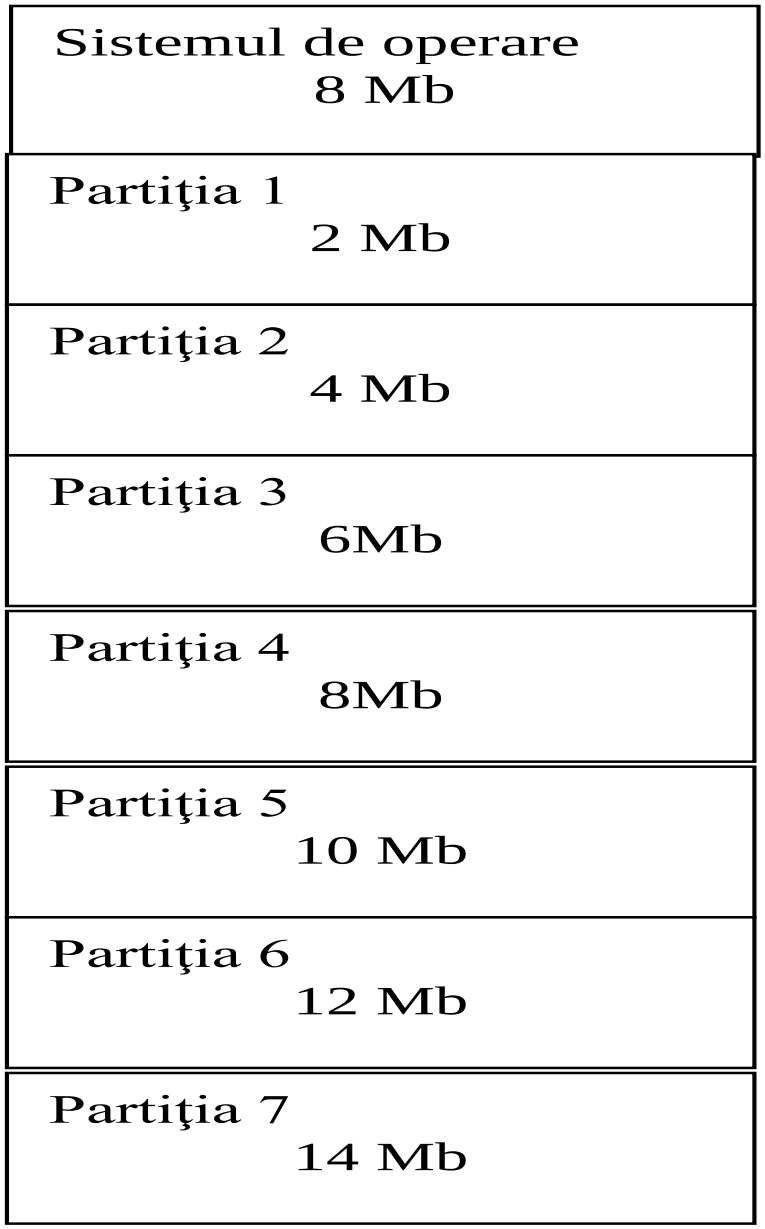
Alocarea cu partiţii fixe

- **Alocarea cu partiţii fixe** (MFT-Memory Fix Tasks sau alocare statică).
- Este prima metodă introdusă pentru sistemele care lucrează în regim de multiprogramare.
- Se presupune că memoria este împărţită în *N* zone disjuncte, de lungime fixă numite **partiţii** şi fiecare partiţie este identificată cu un număr *i (i=1,..,N)*.
- Presupunem că o partiţie i este de lungime N_i şi este alocată unui proces pe toată durata execuţiei lui, indiferent dacă o ocupă complet sau nu.
- Editorul de legături pregăteşte programele pentru a fi rulate într-o zonă de memorie prestabilită.
- Partiţiile pot avea acceaşi dimensiune sau dimensiuni diferite.
- **Exemplu.** În figura urm. avem două tipuri de partiţionări. În ambele situaţii, dimensiunea memoriei este de 64 de Mb şi sistemului de operare îi este alocată o partiţie de 8 Mb. În cazul a) toate partiţiile au aceeaşi lungime iar în cazul b) partiţiile au dimensiuni diferite.

- a) Partiţii cu aceeaşi lungime



b) Partiţii cu lungimi diferite



- Dacă un proces k are nevoie de n_k unităţi de memorie el, poate fi încărcat în oricare dintre partiţiile `i`, pentru care $N_i \geq n_k$.
- În timpul execuţiei procesului, un spaţiu de dimensiune rămâne neutilizat. Acest fenomen se numeşte **fragmentare internă**.
- Problema care se pune este să se aleagă partiţia astfel încât porţiunea de memorie nefolosită să aibă o dimensiune cât mai mică, adică să se minimizeze diferenţele de forma $N_i - n_k$.
- Dacă un proces nu încape în nici una dintre partiţiile existente, el nu poate fi executat.

- Una dintre problemele cele mai dificile este fixarea acestor dimensiuni.
- Alegerea unor dimensiuni mai mari scade probabilitatea ca unele procese să nu poată fi executate, dar scade şi numărul proceselor active din sistem.
- În cazul în care există job-uri în sistem care aşteaptă să fie executate, dar toate partiţiile libere existente la momentul respectiv sunt prea mici, apare fenomenul de **fragmentare externă a memoriei**.

Selectarea job-urilor care urmează să fie executate se face de către **planificator**, în funcţie de necesarul de memorie(pe baza informaţiilor transmise de către utilizator sau determinate automat de către sistem) şi de partiţiile disponibile existente la momentul respectiv. În general, există două moduri de legare a proceselor la partiţii:

- Fiecare partiţie are coadă proprie; legarea la o anumită partiţie a proceselor se va face pe baza necesităţii diferenţei minime între dimensiunea partiţiei şi a procesului (**best fit**-cea mai bună potrivire).

- O singură coadă pentru toate partiţiile; SO va alege pentru procesul care urmează să intre în lucru, în ce partiţie se va executa.

Selectarea lucrării se poate face prin:

- o strategie de tip FCFS(**F**irst **C**ome **F**irst **S**erved), care are dezavantajul că o anumită lucrare trebuie să aştepte în coadă, chiar dacă există o partiţie disponibilă în care ar încăpea, iar în faţa lui în coadă se află job-uri care necesită partiţii mai mari;

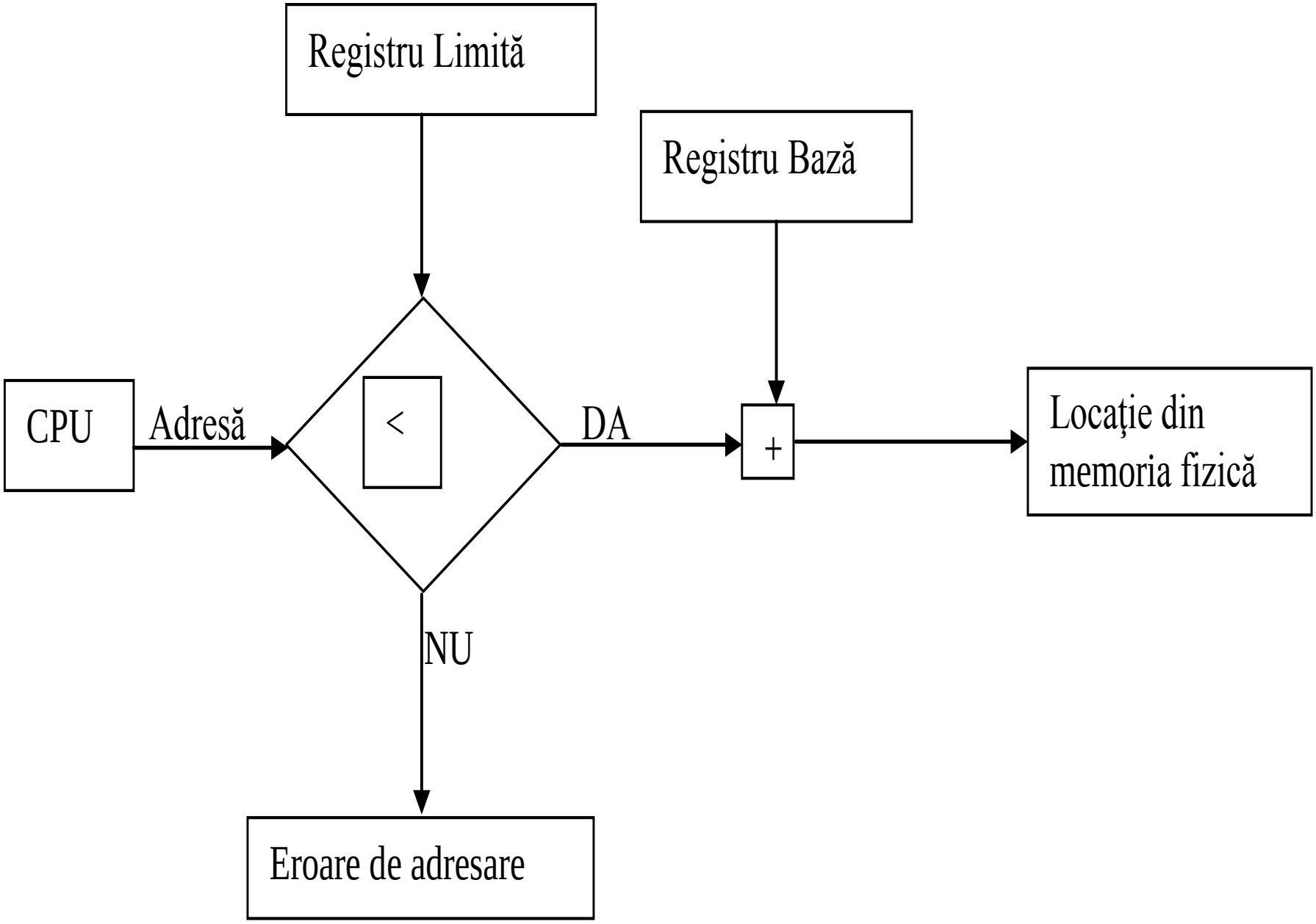
- pe baza împărţirii job-urilor în clase de priorităţi, în funcţie de importanţa lor, care poate avea dezavantajul prezentat mai sus;

- pe baza celei mai bune potrivirii între dimensiunea job-ului cu dimensiunea partiţiei.

Metodele prezentate pot fi combinate. De exemplu, dacă avem mai multe job-uri în sistem care au aceeaşi prioritate, va fi ales cel care se potriveşte cel mai bine peste partiţia care devine disponibilă.

Legarea prin cozi proprii partiţiilor este mai simplă din punctul de vedere al SO; în schimb, legarea cu o singură coadă este mai avantajoasă din punctul de vedere al fragmentării mai reduse a memoriei.

- Deoarece în memorie există mai multe job-uri în execuţie, trebuie rezolvate două probleme: **relocarea** şi **protecţia memoriei.**
- O soluţie a rezolvării ambelor probleme este ca CPU să conţină două registre speciale: **registrul de bază** şi **registrul limită** (figura urm.).
- Într-un fişier executabil, locaţiile au adrese relative la începutul fişierului (prima locaţie are adresa 0, a doua 1 ş.a.m.d.). Când lucrarea este planificată pentru execuţie, în registrul de bază este încărcată adresa primei instrucţiuni din fişierul executabil, iar registrul limită va conţine adresa ultimei locaţii din partiţia respectivă.
- Când o locaţie din fişierul executabil trebuie relocată, adresa ei relativă se adaugă la adresa conţinută în registrul de bază; dacă valoarea astfel obţinută depăşeşte conţinutul registrului limită, atunci are loc o eroare de adresare, altfel se obşine adresa unei locaţii din memoria internă, în care va fi încărcată locaţia din fişierul executabil.
- Alocarea cu partiţii fixe a fost folosită la sistemele generaţiei a III-a de calculatoare(IBM 360, Felix C256/512/1024), dar ea nu este recomandată pentru utilizarea în cadrul sistemelor unde nu se cunoaşte dinainte de ce spaţiu de memorie are nevoie procesul pentru a fi executat, aspect întâlnit adesea în cadrul sistemelor de operare moderne.


Registru Limită
Registru Bază
CPU
Adresă
<
DA
+
Locaţie din memoria fizică
NU
Eroare de adresare

- **Interschimbarea job-urilor(job-swapping)** apare în cazul sistemelor cu organizarea memoriei în partiţii fixe, din necesitatea ca la anumite momente unele dintre ele să fie evacuate din memorie iar altele să fie introduse în memorie.
- De **exemplu**, dacă se execută un job şi apare un alt job de prioritate mai înaltă, jobul de prioritate mai slabă va fi evacuat pe disc.
- În mod normal, un job care a fost evacuat va fi readus în aceeaşi partiţie, restricţie impusă atât strategia de alocare, cât şi de metoda de relocare.
- Dacă relocarea se face în momentul asamblării sau în momentul încărcării(relocare statică), job-ul nu poate fi transferat într-o altă partiţie; dacă se foloseşte relocarea dinamică(cu registru de bază şi registru limită, de exemplu) acest lucru este posibil.
- Interschimbarea joburilor necesită o memorie externă cu acces direct şi rapid, care să poată îngloba copii ale tuturor imaginilor de memorie utilizator.

- Toate procesele ale căror imagini de memorie se află pe disc şi care sunt gata să intre în execuţie se grupează într-o coadă, în timp ce procesele existente în memorie la momentul respectiv formează altă coadă.
- Atunci când planificatorul doreşte să lanseze în execuţie un proces, el apelează dispecerul care verifică dacă procesul se află în memorie.
- Dacă nu şi dacă nu există nici o partiţie liberă, dispecerul evacuează din memorie unul dintre procese, introduce în locul său procesul dorit, reîncarcă registrele şi transferă controlul procesului selectat.
- O acţiune de acest fel presupune şi cea de salvare a contextului procesului în execuţie (a conţinuturilor regiştrilor utilizaţi de către acesta), acţiune care este destul de complexă.

Alocarea cu partiţii variabile

- **Alocarea cu partiţii variabile** (alocare dinamică sau alocare MVT – Memory Variable Task).
- Reprezintă o extensie a alocării cu partiţii fixe, care permite o exploatare mai eficientă a memoriei SC.
- În cazul multiprogramării cu partiţii fixe, problema cea mai dificilă este optimizarea dimensiunii partiţiilor, astfel încît să se minimizeze fragmentarea memoriei.
- De asemenea, se presupune că joburile au o dimensiune cunoscută, ipoteză care nu este în general adevărată.
- Aceste inconveniente pot fi rezolvate dacă se admite **modificarea dinamică a dimensiunii partiţiilor**, în funcţie de solicitările adresate sistemului şi de capacitatea de memorie încă disponibilă la un moment dat.
- Prin folosirea acestei metode, numărul şi dimensiunea partiţiilor se modifică în timp.
- În momentul în care procesul intră în sistem, el este plasat în memorie într-un spaţiu în care încape cea mai lungă ramură a sa.
- Spaţiul liber în care a intrat procesul, este acum descompus în două partiţii: una în care se află procesul, iar cealaltă într-un spaţiu liber care poate fi alocat altui proces.
- De asemenea, când un proces îşi termină execuţia, spaţiul din memorie ocupat de el este eliberat, urmând a fi utilizat de către un alt proces. Apare, deci o alternanţă a spaţiilor libere cu cele ocupate.

- Pentru a se obţine spaţii libere de dimensiune cât mai mare, SO va declanşa operaţia de **alipire a unor spaţii libere vecine sau de compactare a memoriei (relocare a adreselor),** adică de deplasare a partiţiilor active către partiţia ocupată de către nucleul SO, pentru a se concatena toate fragmentele de memorie neutilizate.
- De regulă, operaţia de compactare este complexă, presupunând efectuarea de operaţii de modificare a adreselor; în practică se aleg soluţii de compromis, cum ar fi:

  - Se lansează **periodic** compactarea, la un interval de timp fixat, indiferent de starea sistemului. Procesele care nu au loc în memorie aşteaptă compactarea sau terminarea altui proces.

  - Se realizează o **compactare parţială** pentru a asigura loc numai procesului care asteaptă.

  - Se încearcă numai mutarea unuia dintre procese, cu concatenarea spaţiilor rămase libere.
- **Strategii de administrare a spaţiului din memoria internă.** Aşa cu am menţionat anterior, la un moment dat memoria se prezintă ca o alternanţă a spaţiilor libere cu cele ocupate.
- Cele libere vor fi alocate proceselor care cer memorie, iar cele ocupate, când sunt eliberate trebuie, eventual să fie concatenate cu alte spaţii libere, pentru a obţine zone contigue de dimensiune cât mai mare.
- Deci, sunt necesare metode prin care să se ţină evidenţa spaţiilor libere şi a celor ocupate şi să se aloce spaţiile de memorie solicitate.

- **Administrarea memoriei folosind liste înlănţuite**.
- Vom presupune că întreaga cantitate de memorie solicitată la un moment dat este formată dintr-un **şir de octeţi consecutivi, care se alocă proceselor** dintr**-**un rezervor de memorie (numit **heap**), de unde se ia acestă memorie.
- De asemenea, presupunem că există două rutine, una pentru **a aloca** o zonă de memorie şi de a întoarce adresa ei de început şi o a doua rutină pentru **a elibera** spaţiul alocat anterior, în vederea refolosirii lui.
- Fiecare zonă liberă începe cu un **cuvânt de control,** care conţine un **pointer** către următoarea porţiune liberă şi un camp care conţine lungimea zonei respective. La fel se întâmplă în cazul unei **zone ocupate.**
- O zonă ocupată(respectiv liberă) este reperată după cuvântul ei de control. În timp, eate posibil ca două zone libere să devină adiacente. Sistemul conţine o **procedură de comasare** a două zone libere adiacente.
- În momentul în care un proces cere o anumită cantitate de memorie, sistemul caută o zonă liberă de unde să se ocupe o anumită porţiune.

- Pentru aceasta se folosesc următoarele strategii:

  - Metoda primei potriviri (First-Fit). Esenţa metodei constă în aceea că partiţia solicitată este alocată în prima zonă liberă în care încape.Principalul avantaj al metodei este simplitatea căutării de spaţiu liber.

  - Metoda celei mai bune potriviri (Best-Fit). Esenţa metodei constă în căutarea acelei zone libere care lasă după alocare cel mai puţin spaţiu liber. **Avantajul** metodei constă în economisirea zonelor de memorie mai mari. **Dezavantajul** este legat de timpul suplimentar de căutare şi generarea blocurilor de lungime mică, adică **fragmentarea internă excesivă**.

  Primul neajuns este eliminat parţial, dacă lista de spaţii libere se păstrează nu în ordinea crescătoare a adreselor, ci în ordinea crescătoare a lungimilor spaţiilor libere; în acest caz algoritmul s-ar complica foarte mult.

  - Metoda celei mai rele potriviri (Worst-fit) este dual㠹 metodei Best-Fit. Esenţa ei constă în căutarea acelei zone libere care lasă după alocare cel mai mult spaţiu liber. Deşi numele metodei sugerează că este vorba despre o metodă mai slabă, în realitate nu este chiar aşa. Faptul că după alocare rămâne un spaţiu liber mare, este benefic, deoarece în spaţiul rămas poate fi plasată în viitor o altă partiţie.

Metoda alocării prin camarazi (Buddy-system)

- **Metoda alocării prin camarazi (Buddy-system)** se bazează pe reprezentarea binară a adreselor şi faptul că dimensiunea memoriei interne este un multiplu al unei puteri a lui 2. Presupunem că se dimensiunea memoriei interne este de forma $cx2^n$, iar unitatea de alocare a memoriei este de forma 2^m.
- **Exemplul 1.** Dacă sistemul are o memorie internă de 32 Mo, atunci c=1 şi n=25. Dacă dimensiunea memoriei interne este de 192 Mo, atunci c=3 şi n=26. De asemenea, se poate considera că unitatea de alocare este de 256 Ko, adică m=18.
- Ţinând cont de proprietăţile operaţiilor cu puteri ale lui 2, atât dimensiunile spaţiilor alocate, cât şi ale celor libere sunt de forma 2^k, cu $m \leq k \leq n$. În concluzie, sistemul va păstra liste separate ale adreselor spaţiilor disponibile, în funcţie de dimensiunea lor exprimată ca putere a lui 2. Vom numi lista de ordin k, lista tuturor adreselor unde încep spaţii libere de dimensiune 2^k. Vor exista astfel n-m+1 liste de spaţii disponibile.
- **Exemplul 2**. Dacă considerăm că dimensiunea memoriei interne este de 192 Mo, vom avea 17 posibile liste: lista de ordin 8, având dimensiunea unui spaţiu de 256 octeţi; lista de ordin 9, cu spaţii de dimensiune 512 ş.a.m.d.

- Presupunem că, fiecare spaţiu liber(ocupat) de dimensiune 2^k, are adresa de început un multiplu de 2^k. Două spaţii libere se numesc **camarazi de ordinul** k, dacă adresele lor A1 şi *A2* verifică una dintre proprietăţile următoare:

  - $A1<A2$, $A2=A1+2^k$ şi $A1 \bmod 2^{k+1}=0$

  - $A2<A1$, $A1=A2+2^k$ şi $A2 \bmod 2^{k+1}=0$

- Această definiţie, exprimă o proprietate fundamentală: Atunci când într-o listă de ordinul k apar doi camarazi, sistemul îi concatenează într-un spaţiu de dimensiune 2^{k+1} şi reciproc, un spaţiu de dimensiune 2^{k+1} se poate împărţi în două spaţii de dimensiune 2^k.

- **Algoritmul de alocare de memorie**.

  Pas 1. Fie o numărul de octeţi solicitaţi. Se determină:

  $$\min\{p/m \le p \le n, o \le 2^p\}.$$

  Pas 2. Se determină:

  $k=\min\{i/p \le i \le n$ şi lista de ordin i este nevidă$\}$.

  Pas 3. Dacă k=p, atunci aceasta este alocată şi se şterge din lista de ordinul p altfel se alocă primii 2^p octeţi, se şterge zona din lista de ordinul k şi se creează în schimb alte k-p zone libere, având dimensiunile 2^p, 2^{p+1},...,2^{k-1}.
- **Observaţie.** Pasul 3 al algoritmului se bazează pe egalitatea

  $$2^k-2^p=2^p+2^{p+1}+...+2^{k-1}.$$
- **Exemplul 3.** Se doreşte alocarea a 1000 octeţi, deci p=10. Presupunem ca nu s-au găsit zone libere nici de dimensiunea 2^{10}, nici 2^{11} şi nici 2^{12}. Presup. ca prima zonă liberă de dimensiune 2^{13} are adresa de început $5x2^{13}$ şi o notăm cu I. Ca rezultat al alocării a fost ocupată zona A de dimensiune 2^{10} şi au fost create încă trei zone libere: B de dimensiune 2^{10}, C de dimensiune 2^{11} şi D de dimensioune 2^{12}. Zonele *B*, *C* şi D se trec respectiv în listele de ordine 10, 11 şi 12, iar zona I se şterge din lista de ordin 13.

- **Algoritmul de eliberare.**

  Pas 1. Fie 2^p dimensiunea zonei eliberate. Se introduce zona respectivă în lista de ordinul p.

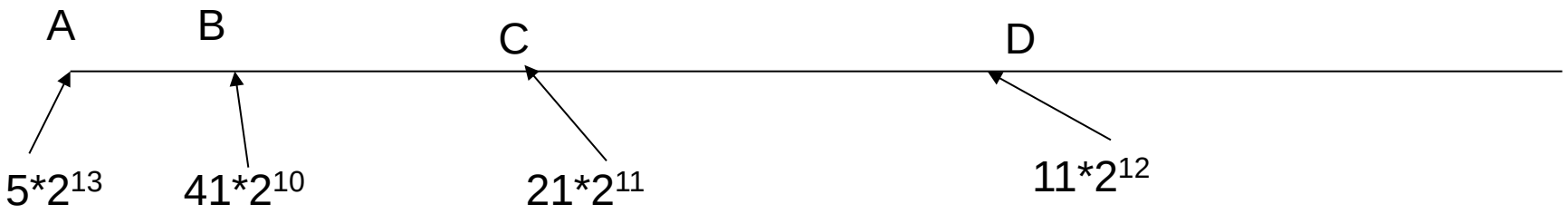
  Pas 2. k:=p

  Pas 3. Verifică dacă există camarazi de ordin k:

  - Dacă **da**, efectuează **comasarea** lor; **Şterge** cei doi camarazi; **Introdu** noua zonă liberă de dimensiune 2^{k+1} în lista de ordin k+1.

  Pas 4. k:=k+1; goto **Pas 3**.

- **Exemplul 4.** Să presupunem, de exemplu că la un moment dat zonele A , C şi D de adrese $5x2^{13}$, respectiv $21x2^{11}$ şi $11x2^{12}$ sunt libere, iar zona B de adresa $41x2^{11}$ este ocupată (fig urmatoare); avem: $41*2^{10}-5*2^{13}=2^{10}$; $21*2^{11}-41*2^{10}=2^{10}$; $11*2^{12}-21*2^{11}=2^{11}$.



- Se observa ca zonele sunt adiacente, în ordinea A, B, C, D. In momentul cind se eliberează B, in conformitate cu paşii descrişi mai sus, se execută următoarele acţiuni:

  - Se trece zona B în lista de ordin 10.

  -Se observă că zonele A şi B sunt camarazi; cele două zone sunt comasate şi formează o nouă zonă X. Zona X se trece în lista de ordin 11, iar zonele A şi B se şterg din lista de ordin 10.

  - Se observă că zonele X şi C sunt camarazi; ele sunt comasate şi se formează o zonă Z care se trece în lista de ordin 12, înlocuind zonele X şi C din lista de ordin 11.

  - Se observă că Z şi D sunt camarazi; ele sunt şterse din lista de ordin 12, iar în lista de ordin 13 se introduce rezultatul comasării lor.