

Curs 4. Managementul memoriei

Intr-un sistem monoprosesor, memoria principala este impartita in 2 parti, o parte pentru sistemul de operare (monitorul rezident, kernel) iar cealalta pentru programul care se afla in executie.

Intr-un sistem multiptogram, partea din memorie alocata userilor trebuie sa fie impartita astfel incat sa fie atribuita fiecarui proces corespunzator userului respectiv. Taskul de impartire si gestionare al memoriei poarta numele de managementul memoriei. Managementul memoriei este un task vital sistemelor de operare deoarece in momentul in care procesele asteapta evenimente de la dispozitivele de I/O sau datorita faptului ca procesorul e ocupat, memoria trebuie sa fie foarte bine gestionata pentru a putea incarca cat mai multe procese.

In mod normal, programatorul nu cunoaste in avans care alte programe vor fi rezidente in memoria principala in timpul executiei propriului program. In plus, ne-ar place sa putem face un swap anumitor procese pentru a mari spatiul pentru cele aflate in starea de READY. Dupa ce un proces a fost swapat el va reveni inapoi in memoria principala la o alta adresa. Aceasta reprezinta relocalizarea. In acest caz, referintele de memorie trebuie translatate in cod la noile adrese fizice.

Datorita acestui fapt apar anumite aspecte tehnice ilustrate in figura urmatoare. Figura

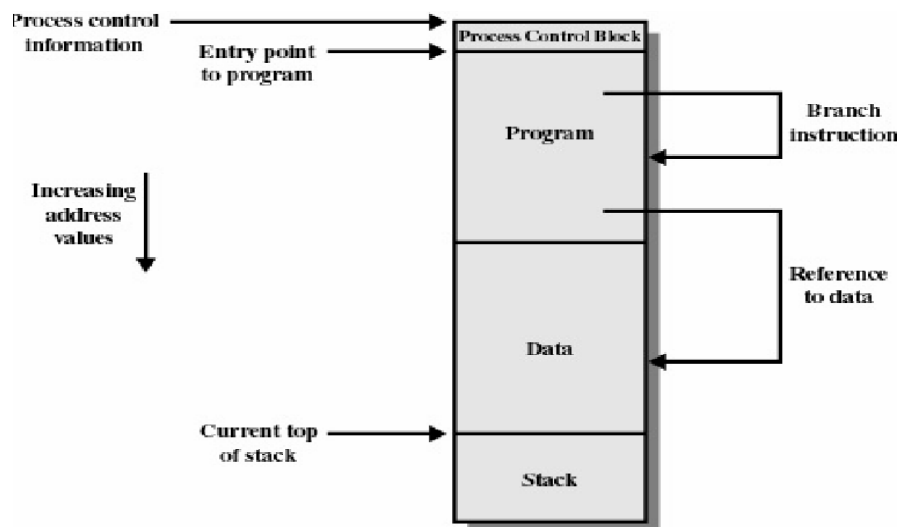


Figura 4.1. Imaginea unui proces in memorie

descrie imaginea unui proces. Pentru simplitate vom considera ca imaginea procesului ocupa o zona continua in memoria principala. Sistemul de operare trebuie sa cunoasca adresa unde gaseste informatia de control a procesului si a stivei ca si punct de intrare pentru executia programului corespondent acestui proces. Deoarece sistemul de operare este responsabil pentru aducerea proceselor in memoria principala, aceste adrese sunt usor de obtinut. In plus, procesorul trebuie sa foloseasca referintele de memorie din program. Din aceasta cauza, hardware-ul (dat de procesor) si software-ul (dat de sistemul de operare) trebuie sa fie capabile sa translateze referintele de memorie gasite in program in adrese fizice reflectand locatia curenta a programului in memoria principala.

Fiecare proces trebuie protejat impotriva nedoritelor interferente ale altor procese, provocate accidental sau intentionat. Astfel, programele reprezentand alte procese trebuie sa nu fie capabile de a accesa referinte de memorie ale altui proces pentru citire sau scriere fara o permisiune in acest sens. Deoarece localizarea programului in memoria principala este necunoscuta, e imposibila verificarea adreselor absolute pentru asigurarea protectiei. Imaginea procesului prezentata in figura anterioara ilustreaza cerintele legate de protectie. Normal, un proces utilizator nu poate accesa o zona a sistemului de operare, program sau date. Apoi un program al unui proces nu poate fi continuat cu o instructiune a altui proces in mod uzual. Nici zona de date nu poate fi accesata intre procese fara a exista un protocol in acest sens. In general in cerintele legate protejarea memoriei sunt satisfacute de procesor.

Fiecare mecanism de protectie trebuie sa fie flexibil astfel incat sa permita mai multor procese sa acceseze o aceiasi zona de memorie. Procesele care coopereaza pentru indeplinirea unui task trebuie sa detina un acces partajat asupra unei zone comune de date. Sistemul de management al memorie trebuie sa permita un acces partajat la aceiasi zona de memorie fara a incalca protectia datelor.

Programele sunt organizate in module in timp ce memoria are o organizare logica data de secvente de biti. O serie de avantaje pot aparea daca sistemul de operare vede programele si datele organizate sub forma unor module:

- Modulele pot fi scrise si compilate independent cu toate referintele dintre ele rezolvate in momentul rularii;
- Pot fi atribuite diverse grade de protectie diferitelor module destul de usor;

- Se poate realiza o distribuire usoara a modulelor catre mai multi utilizatori.

Memoria calculatorului este organizata in cel putin 2 nivele: memoria principala, scumpa dar foarte rapida si memoria secundara, mai ieftina dar inceata. Un task important pentru managementul memoriei il constituie organizarea memoriei pe cele doua nivele.

Una dintre principalele operatii pentru sistemul de operare este aducerea programelor in memoria principala pentru a fi executate de catre procesor. In sistemele moderne aceasta este cunoscuta sub numele de memorie virtuala. Memoria virtuala se bazeaza pe una din urmatoarele tehnici de organizare: segmentare sau paginare. In cazul partiilor de dimensiune egala (partitionare fixa) un proces poate fi incarcat intr-o anumita partitie daca dimensiunea lui este mai mica sau egala cu dimensiunea partitiei. In cazul in care toate partiile sunt pline procesul poate fi swapat. In cazul in care un program nu poate intra intr-o partitie acest lucru trebuie specificat. Practic, sistemul nu este eficient deoarece un program cat de mic va ocupa o intreaga partitie- duce la fragmentare interna.

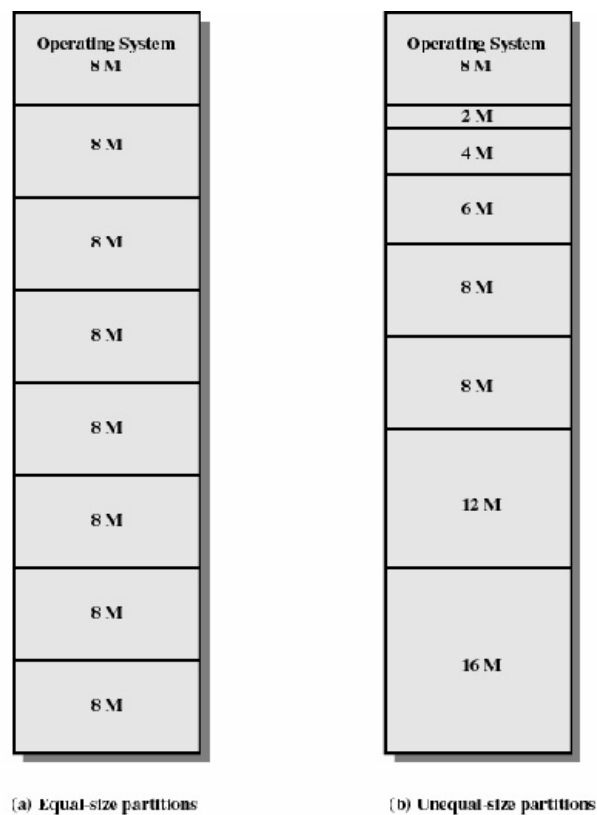


Figura 4.2. Exemple de partitionare fixa

In figura 4.2. avem 2 exemple pentru partitionare fixa. Prima ne prezinta partitiile de dimensiuni egale in timp ce a doua partitionare are dimensiuni diferite pentru partitiile care o compun.

Procesul de partitionare fixa are multe neajunsuri. Datorita partițiilor fixe nu conteaza care dintre ele este utilizata. In cazul in care avem partitiile de dimensiuni diferite exista 2 cai de atribuire a proceselor. Cea mai simpla metoda este atribuirea fiecarui proces celei mai mici partitii in care acesta intra. Apare o coada de asteptare pentru fiecare partiție pentru ca la un moment dat e posibil ca mai multe procese de dimensiuni apropiate sa doreasca aceiasi partiție. Avantajele constau in faptul ca este minimizata cantitatea de memorie irosita. Dar aceasta modalitate este optima din punct de vedere al unei partiții dar nu si din punctul de vedere al intregului sistem.

In figura urmatoare observam cele 2 mecanisme care sunt discutate. Presupunand ca la un moment dat o zona de memorie de 8 Megabytes nu este ocupata de nici un proces dar exista cozi de asteptare pentru portiuni mai reduse de memorie ne dam seama ca acest prim mecanism poate duce la diverse intarzieri care ar putea fi evitate daca s-ar permite unui proces mai mic sa foloseasca si zone mai mari de memorie. Astfel avem o a

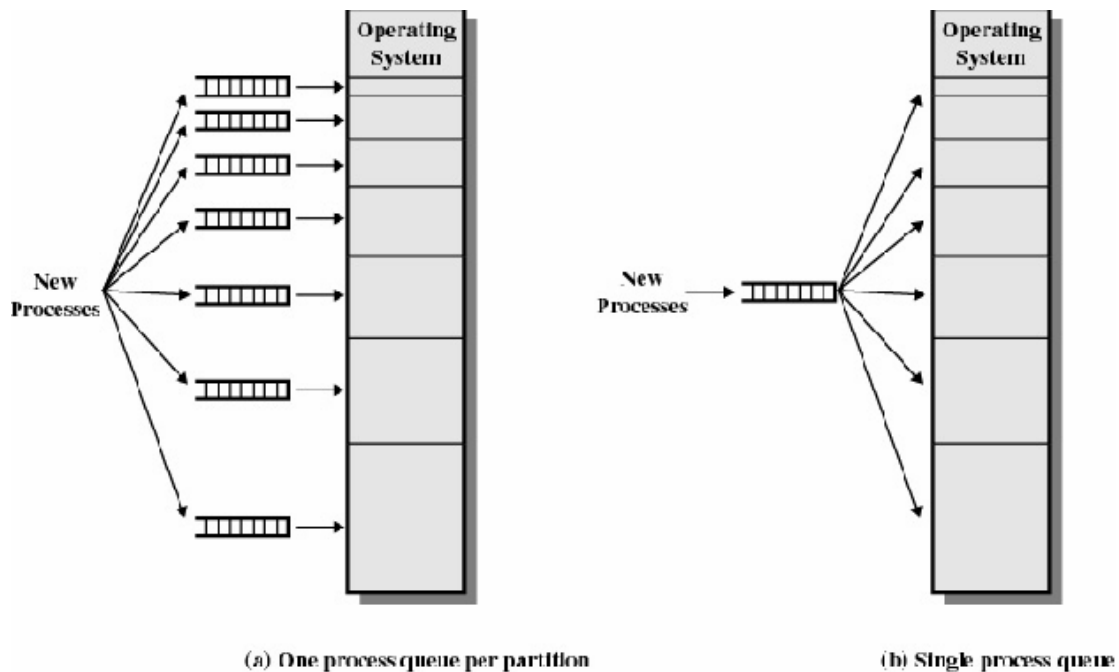


Figura 4.3. Metode de partitionare a memoriei

doua abordare cu o singura coada de asteptare pentru toate procesele. Pentru un proces va fi aleasa cea mai mica partiție pe care acesta o poate ocupa. In cazul in care toate partițiile

sunt ocupate atunci va fi luata o decizie de swapare. In acest caz pot fi luati diferiti factori precum prioritatea sau alte criterii. Partitionarea prin aceasta metoda ofera o relativa flexibilitate dar mai apar o serie de dezavantaje:

- numarul de partitii limiteaza numarul de procese active din sistem;
- ocuparea partiilor nu este eficienta.

Astazi acest mod de organizare nu mai este folosit, face parte doar din istoria calculatoarelor si a sistemelor de operare.

In cadrul partitionarii dinamice, partiile sunt de lungime si numar variabile. Fiecare proces are alocata exact atata memorie cat are nevoie. In figura urmatoare initial memoria principala este goala in afara de partea ocupata de sistemul de operare. Primele 3 procese sunt incarcate incepand cu locatia la care se termina sistemul de operare si ocupa doar spatiul suficient pentru fiecare proces. Apoi sistemul de operare face un swap la 2 procese lasand suficient loc pentru altele datorita faptului ca la acel moment nici un proces nu e in stare de Ready. Deoarece procesul 4 este mai mic ca si ocupare in memorie decat procesul 2 apare un spatiu liber in memorie. Daca se doreste aducerea din nou a procesului 2 nu mai este posibil datorita faptului ca acesta ocupa prea mult loc. Pentru a putea fi adus se face un swap la procesul 1. Aceasta ilustrare ne conduce la aparitia unei fragmentari externe datorata aparitiei spatiilor libere in memorie. O metoda de imbunatatire a situatiei se numeste compactare. Din timp in timp sistemul de operare face un shift pentru procese astfel ca acestea sa ocupe cat mai bine spatiul de memorie pe care il au la dispozitie.

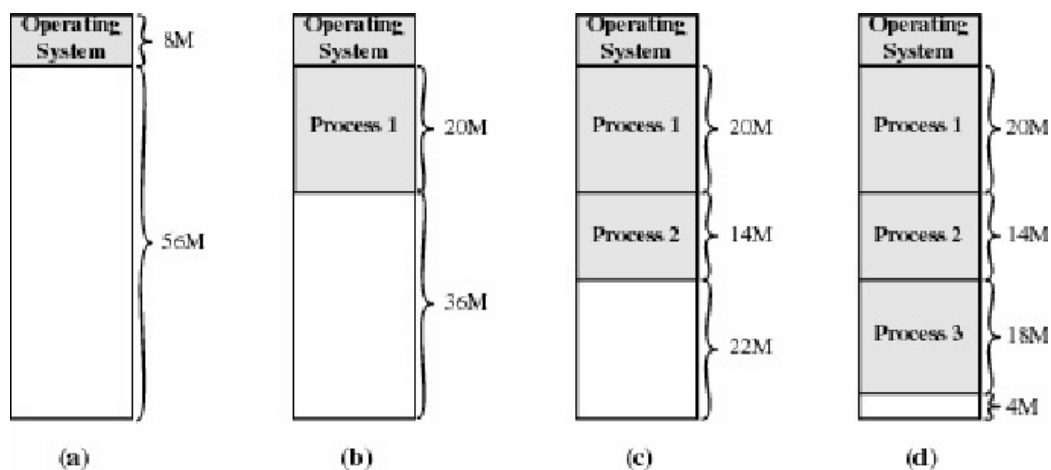


Figura 4.4. Partitionare dinamica

Datorita faptului ca procesul de compactare este unul consumator de timp, sistemul de operare trebuie sa stie cum sa atribuie procese spatiului de memorie. Trei algoritmi de plasare pot fi folositi. Primul dintre acestia se numeste “best-fit” si alege blocul care este cel mai apropiat ca dimensiune de dimensiunea procesului. Algoritmul al doilea poarta denumirea de “first-fit” si porneste cautarea de la inceputul spatiului de memorie alegand blocul de memorie care este suficient de “incapator” pentru proces. Ultimul, “next-fit” scaneaza zona de memorie incepand de la ultimul bloc asignat pana gaseste unul corespunzator pentru proces. In cele 2 reprezentari putem observa diferentele dintre cei trei algoritmi. Astfel, pentru first-fit este gasita prima zona de memorie libera in care putem pune procesul respectiv, pentru best-fit este gasita cea mai buna zona iar pentru next-fit zona de memorie incepand de la ultima alocare. Care dintre cei algoritmi este mai bun depinde si de modul de aparitie sau de swapare al proceselor in ultimele doua cazuri. In general cel mai utilizat este next-fit si pentru ca se raporteaza la un spatiu de memorie care a fost ultimul ocupat. Best-fit este cel mai bun din punct de vedere al ocuparii memoriei dar si cel mai incet in timp ce first-fit este cel mai simplu dar cel mai incet.

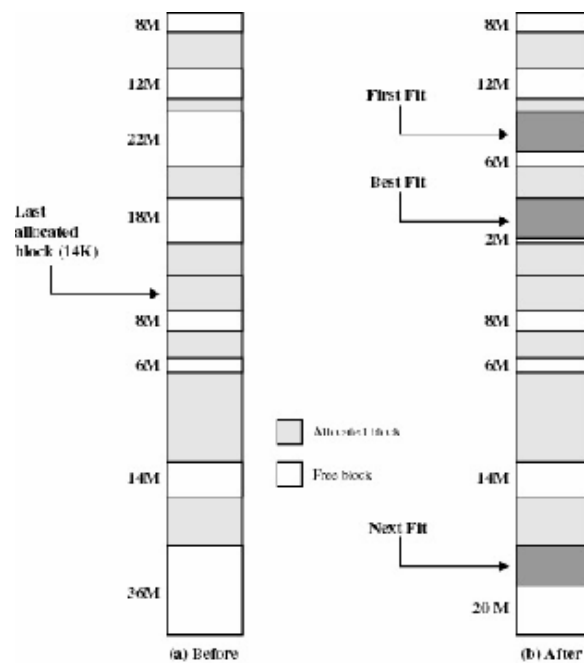


Figura 4.5 Comparatie intre cei 3 algoritmi

Intr-un sistem de operare cu multiprogramare se ajunge la un moment dat in situatia in care toate procesele din memoria principala sunt in starea blocat si chiar si dupa compactare exista prea putina memorie pentru a asigura spatiul pentru alte procese. Atat partitionarea fixa cat si cea dinamica au problemele lor. O partitionare fixa limiteaza numarul proceselor si poate duce la o folosire ineficienta a spatiului de memorie in timp ce una dinamica e mai complicat de realizat si apar probleme la compactare. Un compromis interesant il reprezinta sistemele de tip buddy. Intr-un astfel de sistem, spatiul disponibil pentru alocare este vazut ca blocuri de dimensiunea 2^k unde poate lua valori intre 1 si u obtinandu-se astfel blocuri de marime maxima sau minima.

Pentru inceput, intreg spatiul este tratat ca un singur bloc cu dimensiunea 2^u . Daca este facuta o cerere mai mica decat aceasta valoare atunci intregul bloc este alocat. Altfel, intregul bloc este impartit in 2 de dimensiuni 2^{u-1} . Daca unul dintre ele neimpartit la 2 poate contine procesul atunci acesta este incarcat. Daca nu, algoritmul continua pana se ajunge la dimensiunea dorita. La fiecare moment de timp sistemul mentine o lista de blocuri de dimensiune inferioara care nu sunt alocate. Un spatiu liber din memorie poate fi impartit in 2 si trecut intre blocurile de dimensiune corespunzatoare care pot fi ocupate cu procese. In momentul in care o pereche de blocuri ramine libere ele pot fi unite si obtine un bloc cu o dimensiune mai mare.

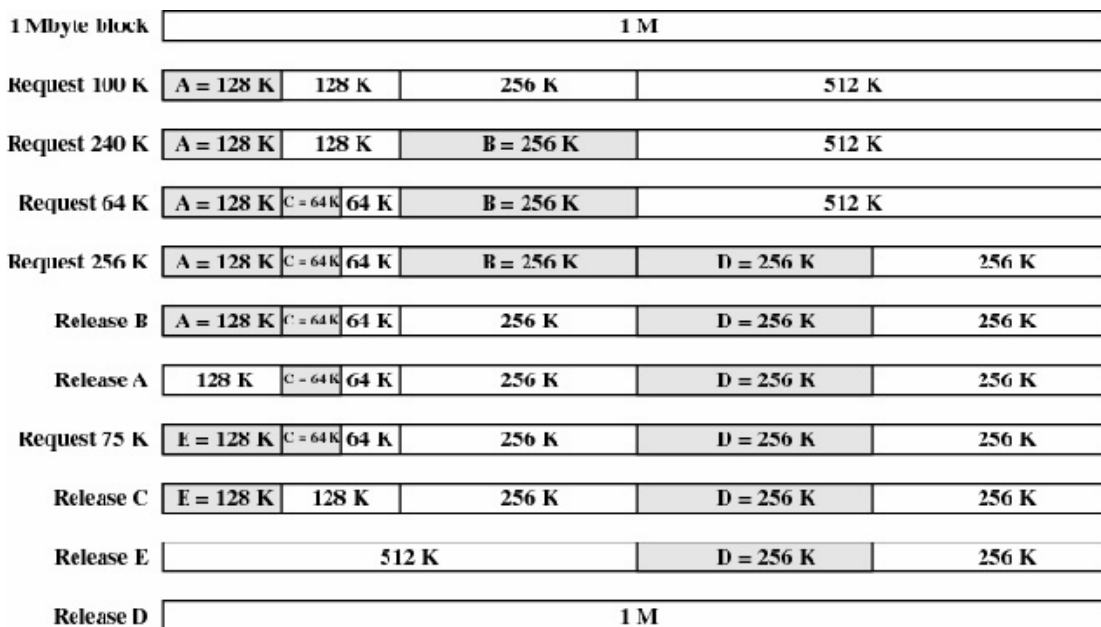


Figura 4.6. Sistemul de tip buddy

In figura consideram un bloc initial de 1 megabyte. Prima cerere, A este de 100 kbytes pentru care alocam un bloc de 128 kbytes. Blocul initial este impartit de 3 ori pentru a obtine segmentul dorit. Urmatoarea cerere, B ne solicita un bloc de 256K. Acest bloc este deja disponibil si poate fi alocat fara probleme. Procesul continua. Observam ca in momentul in care nu mai avem nevoie de blocul E acesta este imediat unit cu jumatatea lui.

Sistemul buddy are o alocare de tip arborescent prezentata in figura 4.7. Frunzele arborelui corespund partitionarii curente a memoriei. Daca 2 parti de memorie sunt frunze insemna ca cel putin una trebuie alocata. In caz contrar ele vor fi unite intr-un bloc mai mare. Cu toate ca sistemul imbunatateste partitionarea fixa sau dinamica, in sistemele moderne folosirea paginarii sau segmentarii la memoria virtuala da rezultate

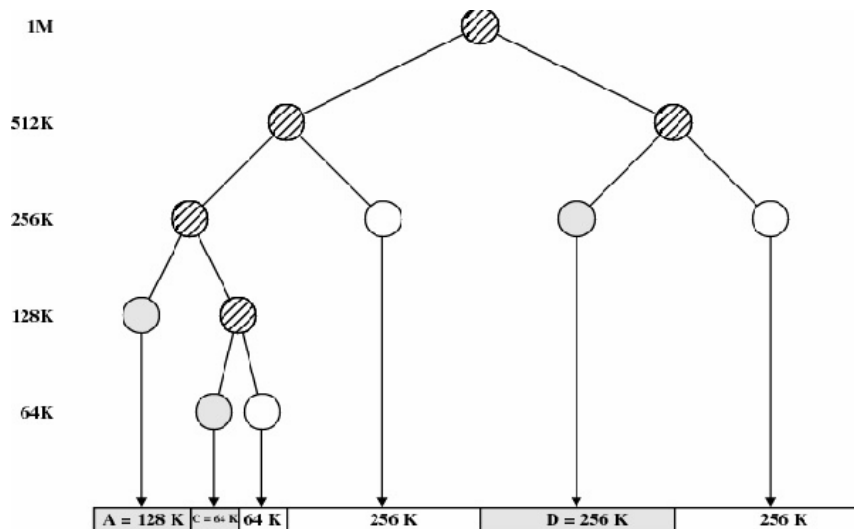


Figura 4.7. Structura arborescenta a sistemului de tip buddy

imbunatatite. Doar Unix-ul mai foloseste o forma modificata a acestui algoritm dar cu tendinte de schimbare.

Am mai prezentat faptul ca un proces nu va avea totdeauna aceiasi locatie in memorie. In timpul executiei acesta va ocupa mai multe locatii absolute din memorie. Deasemenea compactarea poate conduce la ocuparea de diverse locatii de catre un proces. Pentru a vedea cum sunt gasite alte locatii trebuie sa facem o distinctie intre adrese logice si fizice. Daca cele fizice sunt cele absolute, cele logice referă spre locatii de memorie

independente de localizarea datei in memorie. Cele 2 tipuri de adrese impreuna cu cele relative au fost studiate la asamblare.

Figura urmatoare ne prezinta modul de definire al adreselor. Cand un proces primeste starea de Running un registru special, de obicei denumit base register, este incarcat cu adresa de starta procesului. Un alt registru cunoaste adresa locatiei de final a procesului. In timpul executiei procesului luam in calcul adrese relative. Astfel fiecare adresa va fi manipulata in 2 pasi de procesor: in primul rand este adunata valoarea din

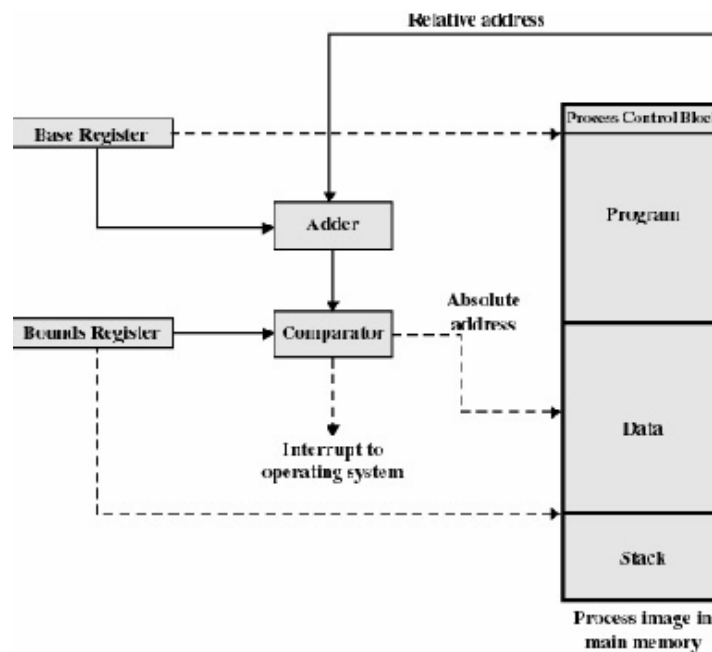


Figura 4.8. Modul de definire al adreselor

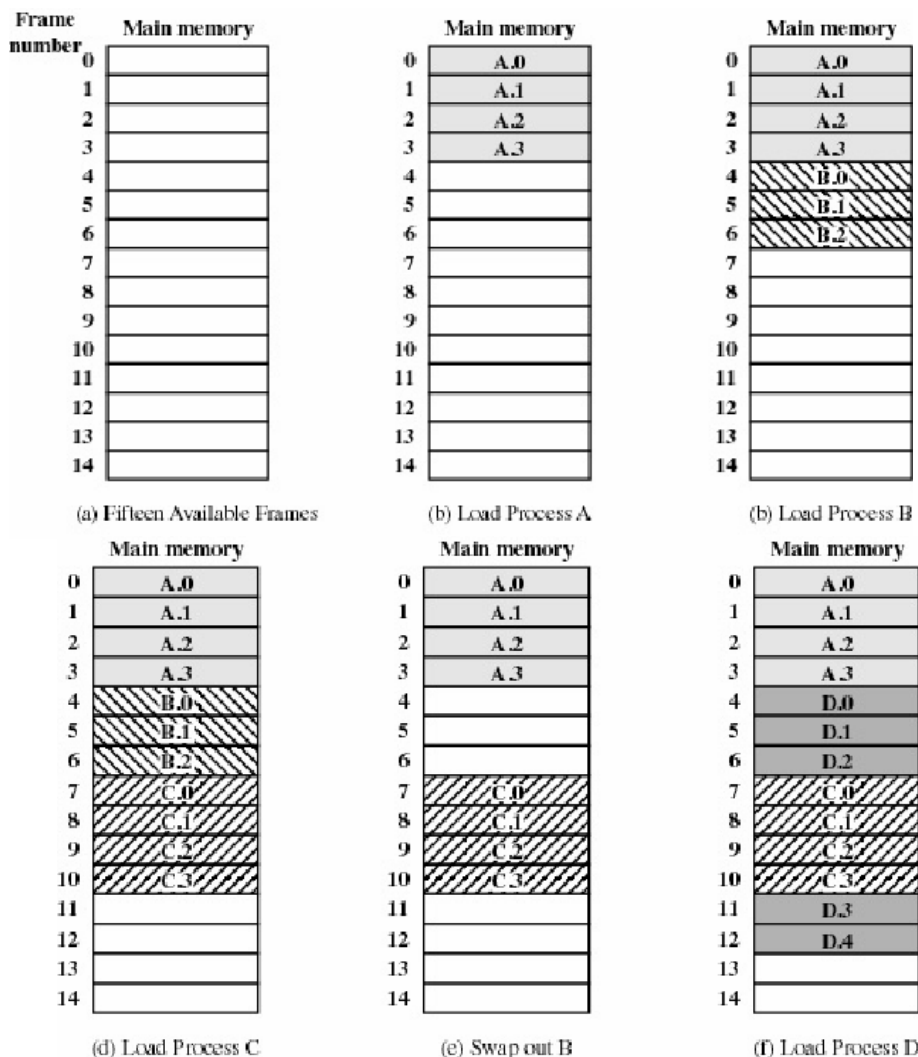
base register pentru a face trecerea de la adresa relativa la adresa absoluta. Apoi este comparata cu adresa din registrul care indica finalul spatiului alocat procesului. In cazul in care o depaseste va fi generata o intrerupere. In caz contrar este executata instructiunea.

O modalitate foarte raspandita pentru managementul memoriei este cea de paginare. Prin aceasta intelegem impartirea in bucati mici a memoriei si impartirea fiecarui proces intr-un numar de entitati de aceasta dimensiune. In cazul procesului partile lui se numesc pagini iar cele de memorie se numesc frame-uri. Sistemul de

operare intretine un tabel pentru fiecare proces. Acest tabel contine locatii pentru fiecare pagina a procesului si adrese de memorie cu numarul paginii si un ofset in pagina.

Odata cu paginarea convertirea din adrese logice in adrese fizice este realizata de catre procesor. In figurile urmatoare am prezentat incarcarea proceselor in memorie. Procesul A contine 4 pagini. Procesul B este incarcat dupa procesul A si are doar 3 pagini. Apoi este incarcat si procesul C cu 4 pagini. Procesul B este suspendat si swappat di memoria principala. Procesul D care este adus in memoria principala ocupa 5 pagini si va ocupa locatiile lasate libere de procesul B si inca 2 in plus.

In figura urmatoare avem tabelele corespunzatoare fiecarui proces cu frame-urile ocupate de catre acesta precum si lista de frame-uri libere. Observam ca paginarea simpla inseamna de fapt o impartire fixa a memoriei cu diferenta ca paginile sunt entitati mici si un proces poate fi impartit in mai multe astfel de entitati.



Segmentarea este alternativa paginarii. Deosebirea consta ca segmentele pot fi de dimensiuni diferite dar exista o dimensiune maxima a unui segment. Adresarea consta din 2 parti – un numar de segment si un offset. Atata timp cat segmentele nu sunt egale segmentarea este similara cu partitionarea dinamica.