Universitatea Politehnica din Bucuresti
Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Implementarea gestiunii memoriei

Studenti: **Moraru Diana (431A)**
Sarbu Lucia Ioana (431A)
Serban Andrei Gabriel (432A)

Cuprins

1. Introducere - Memoria virtuala (*Sarbu Lucia-Ioana*)

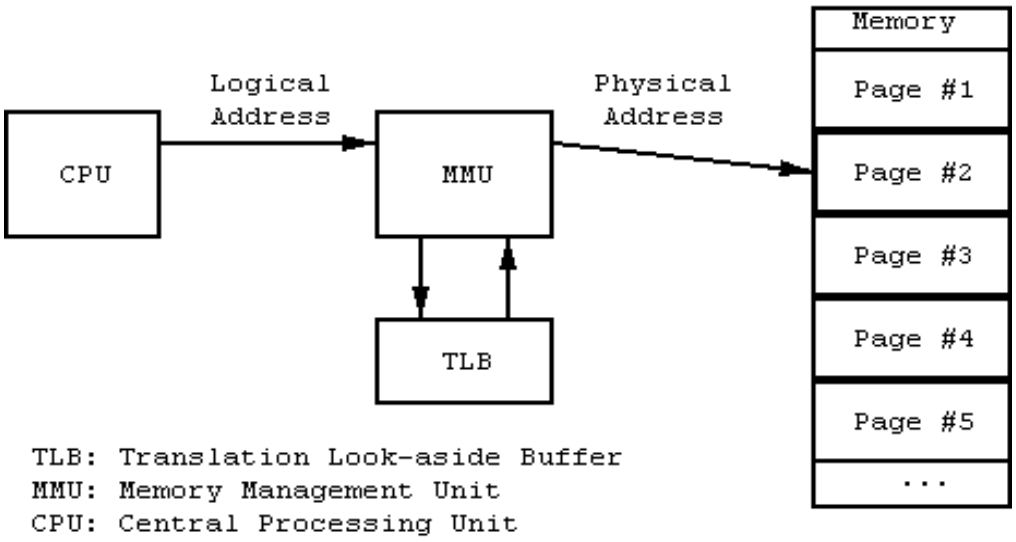
Memoria virtuala constituie un set de tehnici si mecanisme ce se utilizeaza pentru a mari capacitatea a memoriei principale RAM (Random Access Memory). In acest mod, capacitatea memoriei fizice instalata este depasita, ca posibilitati de adresare, prin numarul de biti din instructiune care specifica adresa. In aceste conditii, din programul stocat in memoria externa vor fi preluate in memoria principala numai zonele active la un moment dat. Procesul nu este vizibil utilizatorului si se desfasoara in mod automat. Spatiul de adresare explorat de bitii de adresa din instructiunile furnizate de procesor corespunde capacitatii memoriei virtuale. Sub aspect fizic memoria virtuala va fi organizata in memoria secundara.

Avantajele impelementarii memoriei virtuale sunt:

- Este implementat in hardware cu ajutor din partea Sistemului de Operare - accesul Central Processing Unit – Memoria RAM si Memoria RAM – Hard Disk are timpi de intarziere foarte mici, neglijabili.
- Procesele pot avea spatii de adrese diferite – se evita astfel problemele de suprascriere intre datele accesate de procese concurente si adiacente in memorie
- Spatiul de adrese al procesului poate avea dimensiuni mai mari decât memoria fizic prezenta – reusind astfel permiterea rularii a procese multiple, daca e nevoie, in acelasi timp sau a unor procese mai rapid prin oferirea a mai mult spatiu de stocat date necesare acestuia
- Memoria folosita de proces poate sa nu fie continua – astfel este rezolvata problema proceselor care sunt spre sfarsitul memoriei RAM sau care nu au loc de extindere datorita suprapunerii peste altele
- Unele zone de memorie pot fi partajate de catre mai multe procese – se aloca unor procese atributii de management a memoriei si li se ofera un rol mai important in rulare
- Se poate utiliza dinamic si se poate reconfigura oricare suport fizic aflat la dispozitie - problema unor insuficiente resurse de memorie ramane si se rezolva imediat printr-un upgrade de disk sau RAM

2. Adresarea memoriei virtuale dupa principiul paginarii (*Sarbu Lucia-Ioana*)

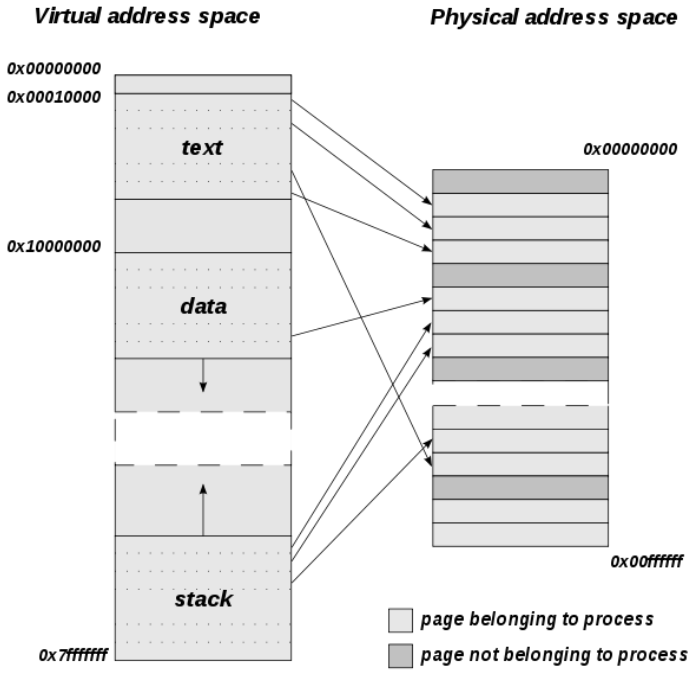
Sistemele cu memorie virtuala folosesc sistemul numit “Paginare”. In fiecare sistem de calcul gasim un set de adrese de memorie pe care programele pot sa le genereze. Atunci cand un program da o comanda de stocare a unei valori se produce generarea o adresa numita “adresa virtuala” care apartine spatiului adreselor virtuale. Daca memoria virtuala nu se poate folosi atunci adresa este pusa direct pe magistrala bus de memorie si adresa respectiva din memoria fizica este scrisa sau citita dupa caz. Cand memoria virtuala este disponibila, adresa nu este trimisa automat pe magistrala de memorie ci este trimisa la Unitatea de gestiune a memoriei (MMU – Memory Management Unit) care transforma adresa virtuala in adresa fizica cum ne este aratat in figura urmatoare.



Pentru a intelege mai bine functionarea sistemului, vom exemplifica cu urmatorul caz: avem un calculator ce poate genera 16 biti de adrese virtuale (de la 0 la 64 Kb). Memoria fizica are 32 Kb, deci nu pot fi translatate toate adresele virtuale in adrese fizice in acelasi timp. O copie a programului de 64 KB trebuie sa existe mereu pe disc pentru ca fragmente din el sa se aduca in memoria fizica atunci cand este nevoie de ele. Spatiul adreselor virtuale este impartit in “pagini”. Entitatile corespunzatoare in memoria fizica se numesc “page frames”. Aceste doua unitati au intotdeauna aceeasi dimensiune. Cand un program trimite o cerere de preluare a unei valori din memorie, el de fapt face o cerere pentru o adresa virtuala, care este trimisa mai departe la Unitatea de gestiune a memoriei, care este, la randul ei, schimbata in adresa fizica si acea data de la adresa fizica este pusa pe magistrala de memorie. Memoria fizica nu stie nimic despre Unitatea de gestiune a memoriei, ea doar primeste comanda de a pune o valoare dintr-o celula pe magistrala de memorie.

Abilitatea de a putea translata 16 unitati de memorie virtuala oricand in cele 8 unitati de memorie fizica nu rezolva problema ca spatiul adreselor virtuale este mai mare decat spatiul adreselor fizice.

Exista doar 8 adrese virtuale in legatura cu memoria fizica la un moment dat. Restul adreselor virtuale sunt marcate cu adresa fizica lipsa. Daca un program vrea sa acceseze o de locatie de memorie fara corespondenta in memoria fizica atunci se genereaza un mesaj catre SO (numit “page fault”) care preia continutul unei locatii de memorie fizica ce nu este folosita, o copiaza pe disc si apoi aloca spatiul nou creat adresei virtuale cerute de program. In momentul in care se incearca accesarea unei valori careia nu ii corespune o valoare din memoria fizica si aceasta nici nu se afla pe disc, atunci inseamna ca acea pagina nu mai exista sau a aparut o eroare si atunci acea intrare va fi eliminata din memorie.



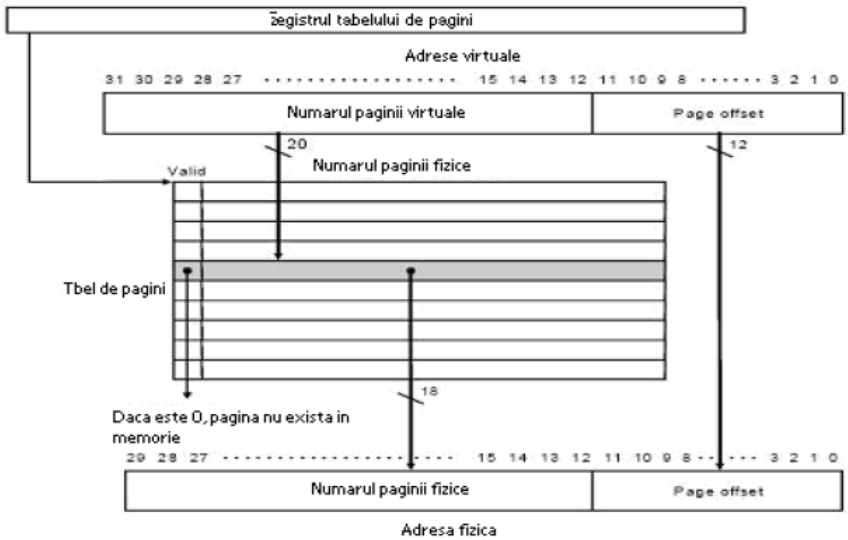
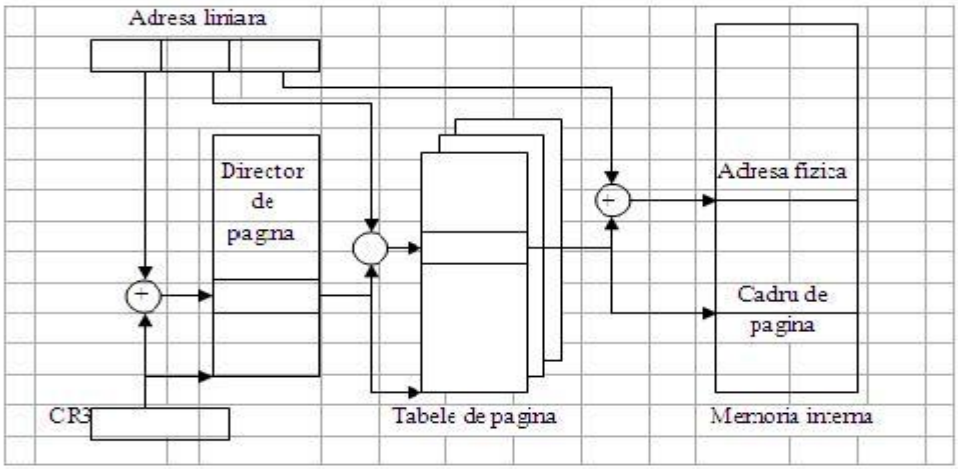


Fig.4

2.1. Paginarea in Windows (*Sarbu Lucia-Ioana*)

Procedeul de paginare are la baza principiul localizarii temporale si spatiale a informatiilor accesate (date si instructiuni). Evidenta paginilor este realizata prin tabele de pagina. O linie de intrare in aceasta tabela contine:

- adresa de "page frame" – 20 de biti ce arata amplasarea paginii in memoria interna,
- bitul P- specifica daca pagina este prezenta memoria interna
- bitul D-(Dirty) – arata daca s-au facut scrieri in pagina respectiva
- bitul U/S – (utilizator/supervizor) – controleaza accesul la pagina in cele doua moduri: utilizator si supervizor
- bitul R/W – (read/write) - indica tipurile de operatii (citire/scriere) ce sunt permise



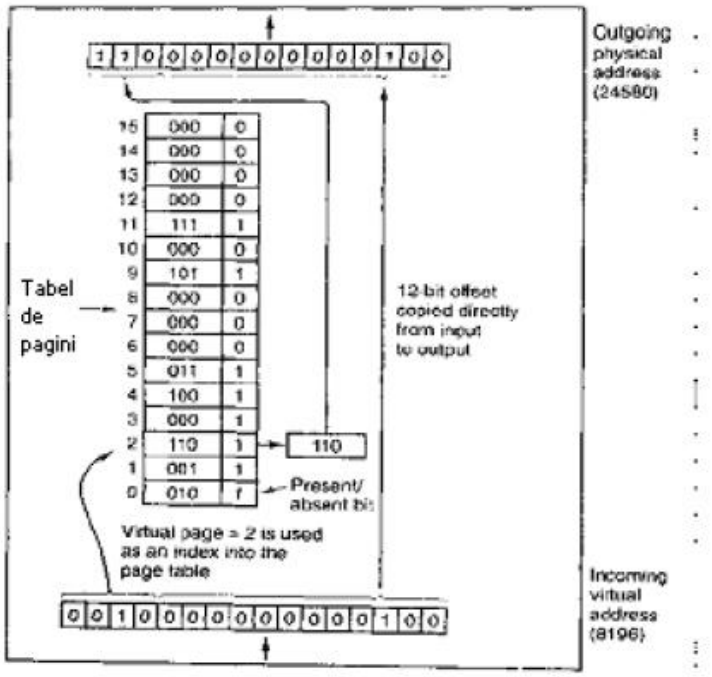
Generata de modulul de segmentare (cu dimensiunea de 32 biti), adresa liniara generala este descompusa in doua campuri: o adresa de decalaj ("offset") si un selector de pagina (de dimensiune - 20 de biti). Din selector, primii 10 biti sunt folositi drept index in directorul de pagina (necesar pentru a determina tabela de pagina), in timp ce restul de 10 biti sunt utilizati pentru regasirea paginii, in calitate de index in tabela de pagina. Adresa de inceput a directorului de pagina se poate gasi in registrul de control al procesorului (CR3). Pentru a ilustra acest lucru, in figura de mai jos este reprezentat mecanismul de conversie prin care, folosind mecanismul de paginare, adresa fizica este convertita in adresa liniara.

In scopul cresterii vitezei de conversie a adreselor se utilizeaza o memoria cache adresabila prin continut, care este de mare viteza si care stocheaza in memorie informatiile ultimelor 32 de pagini .

O mapare a adreselor virtuale in adrese fizice se realizeaza astfel: in primul rand adresa virtuala este impartita in doua, anume bitii high-order si bitii low-order (in primii este inclus numarul paginii virtuale, iar in ultimii decalajul).

In cazul unei adrese de 16 biti, impreuna cu o pagina de 4 kb, una din cele 16 pagini poate fi specificate de primii 4 biti, iar decalajul octetului de urmatorii 12. Pentru

numarul paginii se mai poate realiza o impartire cu 3 sau 5 biti. Impartirile se diferentiaza in functie de marimile diferite ale paginilor.



Operatiile interne ale MMU cu 16 pagini a 4KB

Numarul paginii virtuale actioneaza ca index in cadrul tabelei de pagini, in vederea gasirii intrarii pentru acea pagina virtuala. Tot de acolo, in cazul in care exista, va fi gasit si numarul frame-ului paginii. Acesta este adaugat la capatul high-order al decalajului, formandu-se adresa fizica prin inlocuirea numarului paginii virtuale, prima fiind trimisa catre memorie.

Tabela de paginii are ca scop maparea paginiilor virtuale spre frame-uri de pagina. Transpusa in termeni matematici, tabela de pagina reprezinta o functie, avand numarul paginii virtuale ca argument, in timp ce numarul frame-ului fizic este rezultatul. Prin folosirea rezultatului, un camp al cadrului paginii ar putea inlocui campul paginii virtuale, rezultatul fiind formarea unei adrese fizice de memorie. Doua lucruri trebuie luate in considerare, ca potentiale probleme: anume faptul ca tabela de pagina poate fi foarte extinsa, iar maparea trebuie realizata rapid.

In ce priveste extinderea tabelei de pagina, aceasta este cauzata de adresele virtuale de 32 de biti folosite de PC-urile moderne. Spatiul adreselor virtuale de 32 de biti are 1.000.000 pagini- cu o pagina de 4 kb, in timp ce in cazul adreselor de 64kb, spatiul lor ocupa mai mult. Tabela de pagina trebuie sa aiba 1.000.000 intrari atunci cand exista 1.000.000 pagini in spatiul adreselor virtuale. Fiecare proces are nevoie de o tabela de pagina proprie.

Cea de-a doua problema (maparea care trebuie realizata rapid) se datoreaza necesitatii translatarii virtuale-fizice, proces ce trebuie realizat la fiecare referinta la memorie. O instructiune cuprinde deobicei o comanda instructiune si un operand, astfel

incat va fi nevoie sa se faca referirea la tabla de pagina o data sau de mai multe ori la fiecare instructiune. Astfel, pentru a se evita un blocaj major, este necesar ca o privire in tabela de pagina sa se faca in mai putin de o nano-secunda, in cazul in care o instructiune se face in 4 nano-secunde. Acest lucru face ca maparea rapida si extinsa sa devina o constrangere pentru alcatuirea computerelor.

Aceasta este o problema importanta pentru toate tipurile de computere, indiferent de nivelul lor de performanta.

In ceea ce priveste designul tabelei de pagina, cel mai simplu astfel de design este reprezentat de o tabela de pagina unica, avand in componenta doar o matrice de registre rapide, fiecare avand o intrare virtuala, fiind indexata in functie de numar.

La pornirea unui proces, registrii cu tabela de pagina a procesului sunt incarcati de catre sistem (aceasta este preluata dintr-o copie din memoria RAM principala).Nu este necesar sa se acceseze din nou memoria pentru tabela de pagina atunci cand executia procesului este in curs, acesta fiind un avantaj in plus impreuna cu simplitatea sa. In cazul unei tabele mari, insa, aceasta este o metoda costisitoare, o incarcare a tabelei in fiecare context fiind un obstacol pentru performanta.

Alt exemplu este constituit de tabela de pagina retinuta total in memoria principala. In acest caz, un registru unic ar fi suficient pentru indicarea inceputului tabelei de pagina. Acest lucru face posibila schimbarea, intr-un anumit context, a maparii memoriei, prin incarcarea unui registru. In schimb, vor fi necesare mai multe accesari, pe durata executiei fiecarei instructiuni, ale memorie in scopul citirii intrarilor tabelei de pagine.

2.1.1 Tabele de pagini inversate (*Sarbu Lucia-Ioana*)

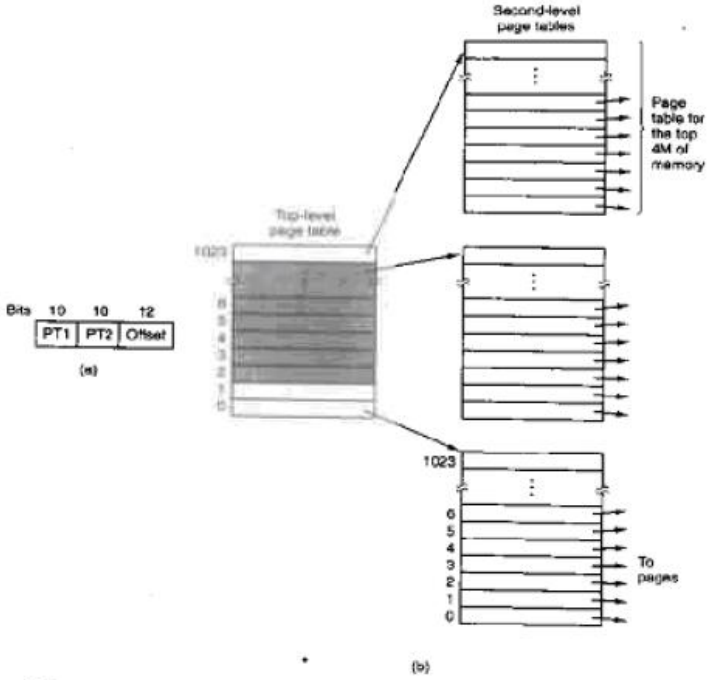
Tabelul de pagini inversate (IPT) poate fi conceput ca o extensie TLB, folosind RAM-ul sistemului normal. In IPT sunt combinate, intr-o unica structura de date, un tabel de pagina si unul de cadru.Nefiind un tabel de pagini adevarat, nu este obligatoriu capabil sa detina toate maparile actuale.Baza lui este formata de un tabel de dimensiuni fixe, fiecare cadru in memorie avand numarul de randuri asociat. La 2000 de cadre, in IPT ar fi 2000 de randuri. Fiecarui rand ii corespunde o intrare (pentru numarul paginii virtuale si paginii fizice), impreuna cu alte date si un mijloc necesar pentru crearea unui lant de coliziune.

Este necesara utilizarea unui tabel de cartografiere a adreselor virtuale pentru a desfasura cautarea in toate intrarile IPT, intrucat structura de baza nu este eficienta. Utilizarea tabelului de cartografiere se va face la un indice in ITP, atunci cand este folosit lantul de slabiciune. Tabelul de distribuire poarta numele de hash. Functia hash nu este, de regula, optimizata, mai dezirabila fiind viteza de prime. Tabelele hash au coliziuni. Datorita hashing-ului, sunt posibile multe coliziuni in utilizare, fiind necesara verificarea fiecarei intrari in tabelul VPN, pentru a cerceta cauza lor- aceasta putand fi o coliziune sau o intrare.

2.1.2 Tabele de pagini pe mai multe niveluri (*Sarbu Lucia-Ioana*)

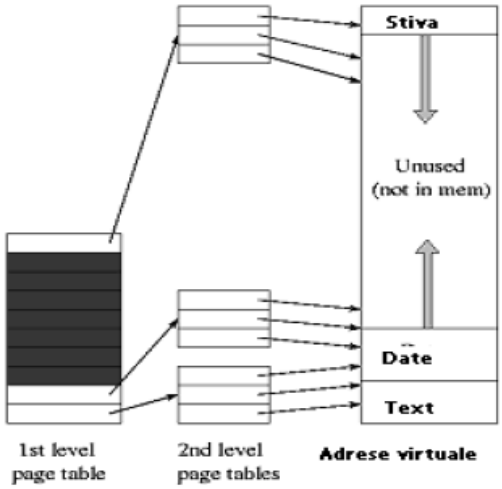
O lista de mapare instalata este pastrata, in IPT, pentru cadrele in memoria fizica, cu toate ca acest lucru ar putea fi risipitor. In schimb este posibila crearea unei structuri de pagini de tabel, care sa cuprinda mapari pentru paginile virtuale, posibilitate ce ar putea fi realizata prin mentinerea tabelelor care contin mai multe pagini ce acopera un anumit bloc de memorie virtuala.

Se poate vedea un exemplu in imaginea de mai jos.. In figura a. avem o adresa de 32 de biti, care este impartita in mai multe campuri. anume cate 10 biti pentru PT1 si PT 2, in timp ce pentru decalaj sunt 12, din decalaj rezultand faptul ca pagina va avea 4 kb.



(a) Adresa pe 32 de biti cu doua campuri pentru tabel
(b) Tbel al paginilor pe doua nivele

Nu este recomandata stocarea tuturor tabelelor de pagina in memorie intotdeauna, atunci cand se foloseste tabela multinivel. In cazul unui proces ce utilizeaza 12MB, primii 4 de jos, din memorie, vor fi distribuiti pentru textul programului, urmatorii pentru date si ultimii pentru stiva.



Campul gri inchis reprezinta zona care nu este folosita din spatiul adreselor virtuale. In cazul in care PTE-urile gri inchis sunt accesate va fi cauzat un page fault, acestea neavand corespondenta catre frame-uri.

In figura b. de mai sus este ilustrat modul in care functioneaza o tabela pe doua nivele. Pe partea stanga apare tabela de pagina cu 1024 de intrari (top-level), aceasta corspunz and campului PT1, care are 10 biti.

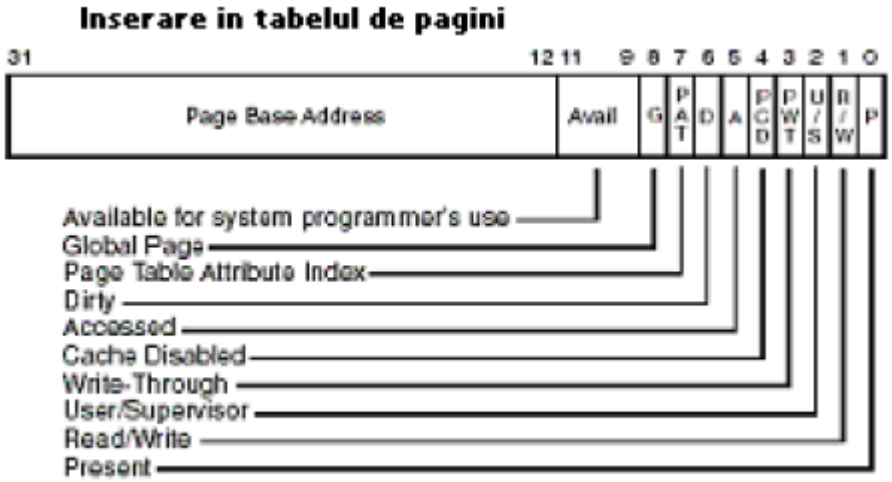
La prezentarea MMU a unei adrese virtuale, este extras, in primul rand, PT1 iar apoi este folosita aceasta valoare drept index in tabela top-level. Fiecare intrare din cele 1024 reprezinta 4M, intregul spatiu de 4 GB al adreselor virtuale fiind desfacut in cadre de cate 1024 octeti.

Adresa sau numarul cadrului de pagina a tabelei de nivel doi este data de intrarea al carei localizare este aflata prin indexarea in tabela de pagina cea mai de sus.

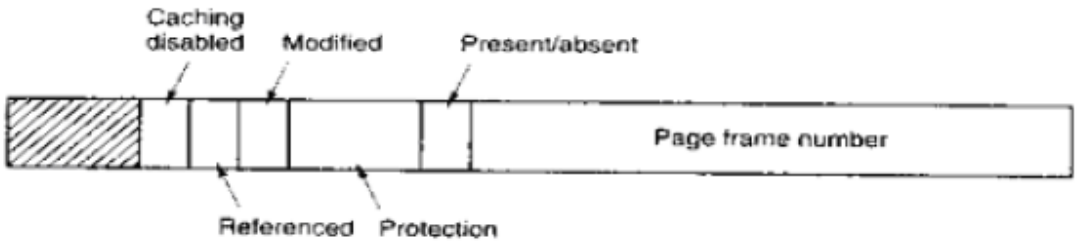
Tabela de pagina pentru textul programului va fi indicata de intrarea 0 (din tabela top-lvel), cea pentru date de catre intrarea 1, tabela de pagina pentru stiva fiind indicata de intrarea 1023, celelalte intrari ramanand neutilizate.

In ce priveste gasirea frame-ului de pagina pentru pagina insasi, indexul de pagina de nivel 2 selectata va folosi campul PT2.

2.1.3 Structura unei intrari in tabela de pagina (*Sarbu Lucia-Ioana*)



Structura unei intrari in tabela de pagina depinde de masina, deci difera foarte mult de la masina la masina. In general, dimensiunea unei intrari in tabela este de 32 biti, dar ea poate varia.



Inserare in tabelul de pagini

Asadar, dupa cum se observa in figura, campul cel mai mare este numarul de Page Frame. Acest camp este si cel mai important, deoarece scopul maparii paginilor este de a afla aceasta valoare.

Urmatorul bit este cel de “Prezent/Absent” (Present/Absent), care este 1 daca intrarea este valida si 0 daca pagina virtuala, cea care corespunde intrarii, nu este prezenta in memoria curenta. In momentul in care se incearca accesarea unei intrari de tabela care are acest bit 0, este generat automat un “page fault”.

Bitii de “Protectie“ (Protected) indica tipul de acces permis. In structurile cele mai simple, acest camp ocupa un bit – (0 pentru citire/scriere si 1 numai pentru citire). Dar campul poate avea, in unele cazuri, si mai mult de un bit. O varianta folosita este cea in care campul are 3 biti - cate un bit pentru fiecare tip de acces: citire, scriere si executie. Bitii de “Modificare” si de “Referentiere” (Modified, Referenced) evidentiaza folosirea si accesarea paginii.

Bitul de “Modificare” este setat automat din partea hardware, in momentul in care se scrie in pagina si este folosit atunci cand sistemul revendica un “page frame”. Daca pagina a fost modificata intre timp, aceasta este considerata “murdara” (Dirty) si trebuie

rescrisa pe disc. Altfel, daca aceasta nu a suferit nicio modificare, ea este considerata "curata" si este ramane la fel, intrucat copia sa de pe disk este valabila.

Bitul de "Referentiere" (Referenced) este 1 de fiecare data cand pagina este accesata sau referentiata (pentru scriere sau pentru citire). Rolul bitului este evidentiat in momentul in care SO trebuie sa aleaga o pagina dar se afla in "page fault". In acest moment, paginile nefolosite sunt considerate valide, mai bune decat paginile folosite, deci bitul de referentiere este important mai ales in sistemele si algoritmii de inlocuire a paginilor.

Bitul "Caching" se poate seta pentru a elimina optiunea de "caching" pentru pagina. Aceasta optiune este mai importanta si mai folositoare in paginile care mapeaza catre registrele diferitelor aparate, decat pentru memorie.

Un lucru important ce trebuie specificat este ca adresa de disc, care este folosita sa stocheze pagina atunci cand nu este memorie, nu este inclusa in tabela de pagina. Tabela de pagina stocheaza numai acele informatii necesare partii hardware pentru a translata adresa virtuala in adresa fizica. Informatia necesara sistemului pentru a se ocupa de "page fault" este retinuta in tabele soft in SO.

2.1.4 Limitele memoriei virtuale si a celei fizice (*Sarbu Lucia-Ioana*)

Dimensiunea memoriei virtuale si a celei fizice de care dispune un sistem de calcul ce foloseste un SO Windows depinde de configuratia hardware si de varianta Windows folosita. Pe un hardware de 32 de biti, dimensiunea adresei virtuale este de 4 GB si catitatea maxima de memorie fizica se cuprinde intre 2 si 128 GB. Pe un hardware de 64 de biti, dimensiunea adresei virtuale este de 16 TB si cantitatea maxima de memorie fizica se cuprinde intre limitele de 64 GB si 1 TB.

In urmatorul tabel, sunt specificate, pentru fiecare editie de Windows:

- cantitatea de memorie virtuala
- cantitatea maxima de memorie fizica

Operating system version	Edition	Virtual memory	Maximum physical memory
Windows Server 2003	Standard	4 GB	4 GB
	Web	4 GB	2 GB
	Enterprise	4 GB	32 GB, if hardware supports PAE
	Enterprise (64-bit)	16 terabytes	64 GB
	Datacenter	4 GB	128 GB, if hardware supports PAE
	Datacenter (64-bit)	16 terabytes	512 GB
Windows XP	Home	4 GB	4 GB
	Professional	4 GB	4 GB
	64-bit Edition Version 2003	16 terabytes	128 GB
Windows 2000	Professional	4 GB	4 GB
	Server	4 GB	4 GB
	Advanced Server	4 GB	8 GB
	Datacenter Server	4 GB	32 GB, if hardware supports PAE

2.2 Paginarea in Linux (*Sarbu Lucia-Ioana*)

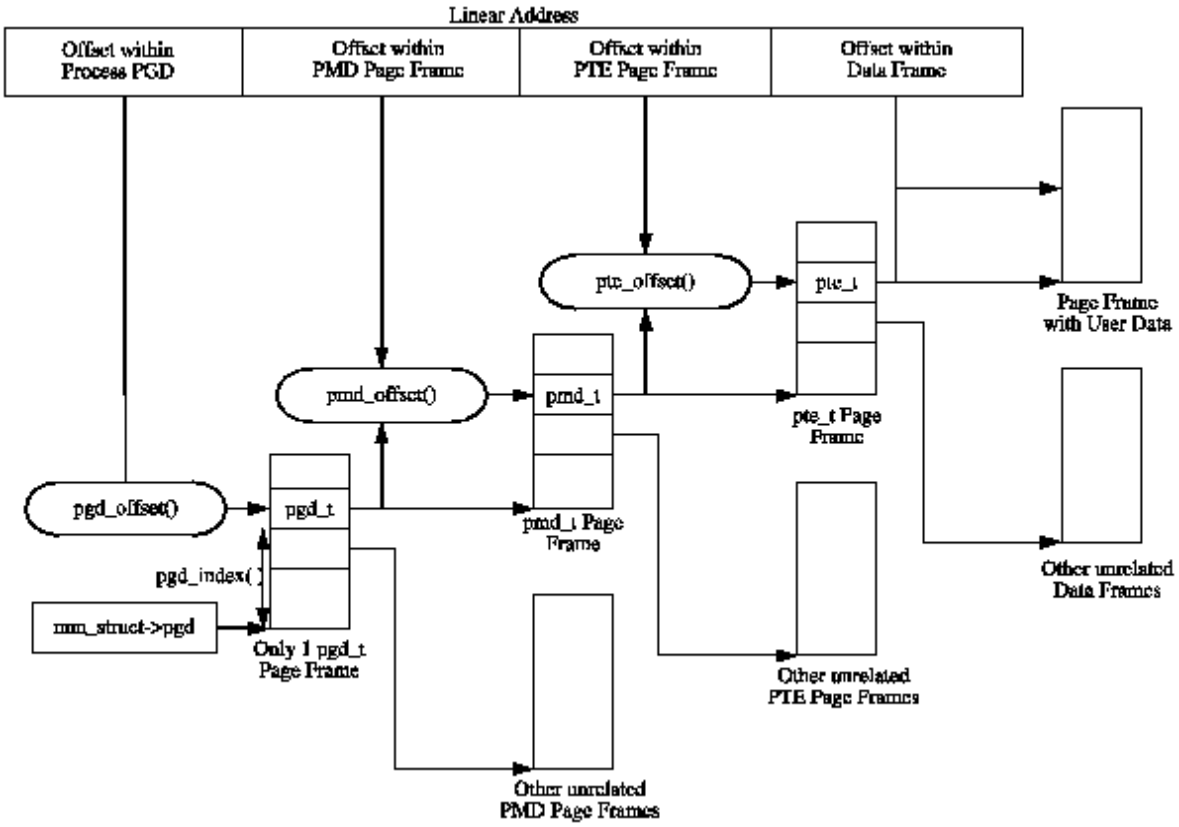
Nivelele sistemului de operare Linux sunt (in)dependente intr-un mod neobisnuit, in comparatie cu alte masini. Alte SO au obiecte care gestioneza paginile de suport fizic (exemplu: obiect PMJP in BSD). Linux retine conceptul de tabel de pagini pe 3 niveluri in codul de arhitectura independent, chiar daca arhitectura care sta la baza nu-l suporta. In timp ce aceasta structura este usor de inteles, conceptual, diferenta dintre diversele tipuri de pagini este foarte vaga. Aceste tipuri de pagini sunt recunoscute prin fanioanele („flags”) lor.

Acest capitol incepe prin descrierea modului in care este amenajat tabelul de pagini si a tipurilor folosite pentru a reprezenta cele 3 niveluri diferite de tabele de pagini. Urmeaza apoi modul in care o adresa virtuala este impartita in partile sale componente si structura lor. Apoi se va discuta despre despre modul de intrare in nivelul cel mai scazut, pagina de intrare in tabel (PTE) si bitii utilizati de catre partea hardware. Se va aminti apoi despre macrocomenzile utilizate pentru a naviga in tabelul de pagini, despre stabilirea si verificarea atributelor, despre modul in care in tabelul de pagini este populat si felul in care paginile sunt alocate si eliberate.

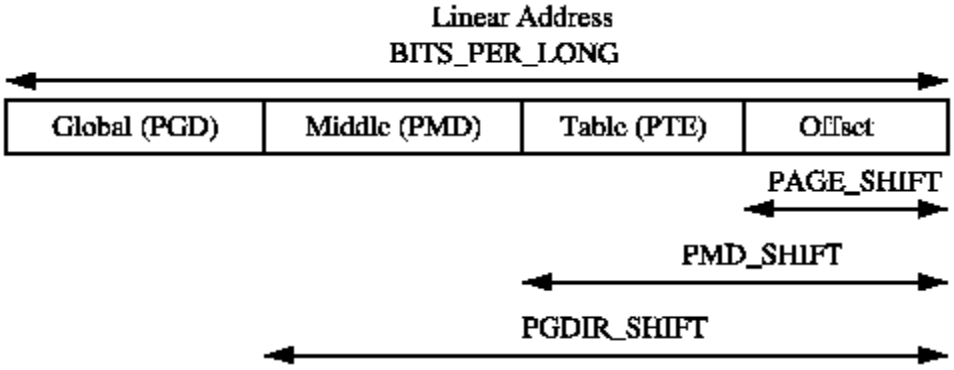
2.2.1 Descrierea paginii directoare (*Sarbu Lucia-Ioana*)

Fiecare pagina directoare are in vedere un pointer (mm_struct → PGD) la propria pagina Global Directory (PGD), care este un cadru fizic. Acest cadru contine o serie de tipuri pgd_t - tip de arhitectura specifica definita in <asm/page.h>. Tabelele de pagini sunt alocate in mod diferit in functie de tipul arhitecturii. Pe masina x86, pagina procesului curent este incarcata la copierea mm_struct → PGD in registrul CR3, rezultand stergerea TLB-ului. De fapt, acesta este functia __flush_tlb () ce se foloseste in codul de arhitectura dependenta.

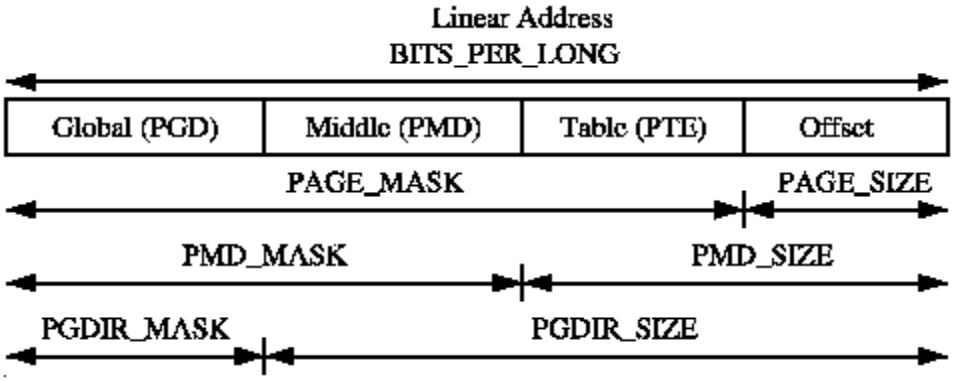
Fiecare intrare ce este activa din tabelul PGD indica spre un cadru de pagina care contine o serie de Page Middle Directory (PMD) si intrarile tip pmd_t care, la randul lor, indica spre cadrele care includ Page Tabelul Entries (PTE) de tipul pte_t . Acestea indica, si ele, catre cadrele care includ datele efective ale utilizatorului. Intrarea swap se stocata in PTE in momentul in care pagina este vazuta drept suport de depozitare si ea poate fi utilizata prin do_swap_page () in momentul in care pagina nu gaseste intrarea swap ce contine datele ei. In imaginea urmatoare este reprezentat formatul tabelului de pagini:



Pentru a obtine compensatii pe cele 3 niveluri din tabelul de pagini si un decalaj in cadrul paginii reale, o adresa liniara indicata poate fi impartita in mai multe parti. Pentru descompune adresa liniare in partile sale componente, sunt prevazute cateva macrocomenzi in 3 perechi, cate o pereche pentru fiecare nivel in tabelul de pagini: un SHIFT, o dimensiune si o macrocomanda MASK. Prima pereche, “SHIFT”, indica, pentru fiecare nivel din tabelele de pagini, cati biti (lungimea) sunt mapati pe acestea.



Valorile macrocomenzii “MASK” pot fi utilizate in cadrul operatiei “AND” cu o adresa liniara pentru a masca toti bitii superiori. Aceste valori sunt utilizate des pentru a determina daca o adresa liniara apartine unui nivel anume din tabel. Macrocomenzile “SIZE” indica numarul de octeti adresati de fiecare intrare a fiecarui nivel. Relatia intre ultimele doua macrocomenzi discutate este reprezentata in figura urmatoare:



Pentru calculul fiecareia dintre cele trei perechi, este importanta numai macrocomanda SHIFT (SIZE si MASK sunt calculate pe baza acesteia). De exemplu, cele trei macrocomenzi pentru nivelul de pagina de pe x86 sunt:

```
5 #define PAGE_SHIFT      12
6 #define PAGE_SIZE       (1UL << PAGE_SHIFT)
7 #define PAGE_MASK       (~(PAGE_SIZE-1))
```

PAGE_SHIFT reprezinta dimensiunea in biti a partii decalate (offsetul) a spatiului adresei lineare (12 biti pentru x86). Dimensiunea unei pagini se poate calcula usor pe baza codului de mai sus. In final, masca se calculeaza prin negarea bitilor care alcatuiesc PAGE_SIZE - 1. Daca o pagina trebuie sa se alinieze la marginea unui pagini, PAGE_ALIGN () este folosit. Acesta macrocomanda adauga PAGE_SIZE - 1 la adresa inainte de a aplica operatia si logica cu PAGE_MASK.

PMD_SHIFT reprezinta numarul de biti din adresa liniara (cei mapati de partea a doua a nivelului tabelului). PMD_SIZE si PMD_MASK sunt calculate intr-un mod similar cu macrocomenzile nivelelor de pagina.

PGDIR_SHIFT este numarul de biti care sunt mapate in top, sau primul nivel, din tabelul de pagini. PGDIR_SIZE si PGDIR_MASK sunt calculate in acelasi mod ca mai sus.

2.2.2 Descrierea unei intrari in tabelul de pagini (*Sarbu Lucia-Ioana*)

Asa cum am mentionat, fiecare intrare este descrisa de structs pmd_t pte_t, si pgd_t pentru PTE-uri, PMD-uri si respectiv PGD-uri. Chiar daca acestea sunt de multe ori doar numere intregi fara semn, ele sunt definite ca structs din doua motive. Primul este pentru protectie a tipului, astfel incât acestea nu vor fi utilizate necorespunzator. Al doilea este pentru caracteristici cum ar fi PAE pe x86 in cazul in care apare o suplimentare de 4 biti pentru adresarea de mai mult de 4GiB de memorie. Pentru a stoca biti de protectie, este definit pgprot_t care detine fanioane relevante si este, de obicei stocat in intrarea in tabelul de pagini unde se afla bitii inferiori.

Pentru nominalizarea tipului, 4 macrocomenzi sunt furnizate in asm / page.h, care iau tipurile de mai sus si returneaza partea relevanta din structuri. Acestea sunt: pte_val (), pmd_val (), pgd_val () si pgprot_val (). Pentru a inversa tipul, 4 macrocomenzi sunt furnizate: __pte (), __pmd (), __pgd () si __pgprot ().

Pentru scop ilustrativ, vom examina cazul unei arhitecturi x86 fara PAE activate, dar cu aceleasi principii ce se aplica in toate arhitecturile. Pe o x86 cu nici un PAE, pte_t este pur si simplu un intreg pe 32 de biti intr-o structura. Fiecare pte_t face referire la o adresa a unui cadru de pagina si toate adresele subliniate sunt garantate a fi aliniate cu pagina. Prin urmare, exista PAGE_SHIFT (12) biti, in care valoarea pe 32 de biti, este libera fata de bitii de stare de la intrarea in tabelul de pagini. Un numar de biti de protectie si de stare sunt enumerati in tabelul urmator dar semnificatia bitilor variaza intre arhitecturi.

Bit	**Function**
_PAGE_PRESENT	Pagina este prezenta dar nu schimbata
_PAGE_PROTNONE	Pagina este prezenta dar nu poate fi accesata
_PAGE_RW	Setare in cazul in care pagina va fi scrisa
_PAGE_USER	Setare daca pagina e accesibila din spatial userului
_PAGE_DIRTY	Setarea in cazul in care pagina e scrisa
_PAGE_ACCESSED	Setarea in cazul in care pagina e accesata

Acesti biti sunt auto-explicativi, cu exceptia _PAGE_PROTNONE de care vom discuta mai departe. Pe un x86 cu Pentium III si mai mare, acest bit se numeste atribut Page Table (PAT), in timp ce arhitecturi mai vechi, cum ar fi Pentium II au avut acest bit rezervat. Bitul PAT este folosit pentru a indica dimensiunea paginii PTE. Intr-o intrare PGD, acest bit este numit Dimensiunea exceptata a paginii (PSE), evident, acesti biti sunt destinati sa fie utilizati in conjunctie.

Linuxul nu utilizeaza bitul PSE pentru pagini de utilizator, bitul PAT este liber in PTE pentru alte scopuri. Exista o cerinta pentru a avea o pagina rezidenta in memorie, dar inaccesibila la procesele din spatiul utilizator, cum ar fi in momentul in care o regiune este protejata cu mprotect () cu pavilion PROT_NONE. In momentul in care regiunea se vrea a fi protejata, _PAGE_PRESENT bit este sters si _PAGE_PROTNONE bit este setat. Macrocomanda pte_present() verifica daca oricare dintre acesti biti sunt setati pentru ca kernel-ul sa stie ca PTE este prezent, doar inaccesibil in mod utilizator, care este un punct subtil, dar important. Ca bit de hardware _PAGE_PRESENT este NULL, un defect pagina va avea loc in cazul in care pagina este accesata astfel incat Linux sa poata aplica protectia din timp.

2.2.3 Folosirea intrarilor a tabelelor de pagini (*Sarbu Lucia-Ioana*)

Macrocomenzile definite in <asm/pgtable.h> sunt importante in navigarea si examinarea de intrari in tabelul de pagini. Pentru a naviga in directoarele de pagina, trei macrocomenzi sunt furnizate. Pgd_offset () preia o adresa, mm_struct pentru procesul de intrare si returneaza PGD care acopera adresele solicitate. pmd_offset () ia o intrare PGD si o adresa si returneaza PMD-uri relevante. pte_offset () ia PMD-ul si returneaza un PTE relevant. Restul de adresa liniara este compensata in cadrul paginii.

A doua runda de macrocomenzi determina daca intrarile din tabelul de pe pagina sunt prezente sau pot fi utilizate.

- pte_none (), pmd_none () si pgd_none () intoarce 1 daca intrarea corespunzatoare nu

exista;
- pte_present (), pmd_present () si pgd_present () intoarce 1 daca intrarile corespunzatoare in tabelul de pagini au stabilit ca bitul e prezent;
-pte_clear (), pmd_clear () si pgd_clear () va sterge mentiunea corespunzatoare in tabelul de pagini;
- pmd_bad () si pgd_bad () sunt utilizate pentru a verifica intrarile in momentul in care a trecut ca parametri de intrare pentru functii care pot schimba valoarea intrarilor. Fie ca returneaza 1 aceasta variaza intre putine arhitecturi care definesc aceste macrocomenzi, dar pentru cei care definesc de fapt, asigura ca pagina de intrare este marcata ca fiind prezenta si accesata , acestea fiind cele mai importante doua controale.

Exista mai multe parti ale memoriei virtuale care sunt pline in tabelul de pagini de coduri si este important sa fie recunoscute. Un exemplu foarte simplu de o plimbare in tabelul de pagini este functia follow_page() ce se afla in / memory.c mm. Urmatoarele intructiuni fac parte dintr-un fragment din aceasta functie (partile ce nu au legatura cu parcurgerea tabelului de pagini au fost omise):

```
pgd_t *pgd;
pmd_t *pmd;
pte_t *ptep, pte;

pgd = pgd_offset(mm, address);
if (pgd_none(*pgd) || pgd_bad(*pgd))
        goto out;

pmd = pmd_offset(pgd, address);
if (pmd_none(*pmd) || pmd_bad(*pmd))
        goto out;

ptep = pte_offset(pmd, address);
if (!ptep)
        goto out;

pte = *ptep;
```

Se folosesc pur si simplu cele trei macrocomenzi de offset (decalaj) pentru a naviga prin pagina de tabele si macrocomenzile _NONE () si _bad () pentru a asigura ca tabelul de pagina este valid.

Al treilea set de macrocomenzi examineaza si seteaza permisiunile de intrare. Permisiunile determina ce proces se poate sau nu se poate face cu o anumita pagina. De exemplu, intrarile din tabelul de pagina Kernet nu sunt niciodata citite printr-un proces in mod utilizator.

• permisiuni de citire pentru o intrare sunt testate cu pte_read (), setate cu pte_mkread (), si compensate cu pte_rdprotect ();
• permisiuni de scriere sunt testate cu pte_write (), setate cu pte_mkwrite (), si compensate cu pte_wrprotect ();
• permisiuni de executie sunt testate cu pte_exec (), setate cu pte_mkexec (), si

compensate cu pte_exprotect ();
• permisiunile pot fi modificate la o noua valoare cu pte_modify (), dar utilizarea aceasta este aproape inexistenta. Este folosite doar in functia change_pte_range () in mm /mprotect.c.

Al patrulea set de macrocomenzi examineaza si stabileste starea a unei intrari. Exista doar doi biti importanti in Linux - bitul “murdar” si bitul “accesat”. Pentru a verifica acesti biti sunt utilizate macrocomenzile pte_dirty () si pte_young (). Pentru a seta bitii sunt folosite macrocomenzile pte_mkdirty () si pte_mkyoung (). Pentru a le sterge sunt diponibile macrocomenzile pte_mkclean si pte_old () .

3. Algoritmi de inlocuire a paginilor(Windows si Linux) (*Serban Andrei-Gabriel*)

Cand apare o eroare in mecanismul de paginare,sistemul de operare trebuie sa aleaga o pagina pe care sa o sterga din memorie pentru a face loc unei pagini noi.Daca pagina in cauza a fost modificata in memorie,trebuie rescrisa pe disk.Pagina noua este scrisa in spatiul eliberat din memorie.

Este posibil sa alegem o pagina aleatoriu si o sa o mutam din memorie dar este de preferat sa eliberam memoria de paginile mai putin folosite.Daca mutam o pagina folosita mult de sistemul de operare,riscam sa crestem numarul operatiilor de lucru cu memoria.Pentru o alegere optima s-au facut multe studii teoretice cat si experimentale.In paragrafele urmatoare sunt descrisi cei mai importanti algoritmi.

Pentru a determina ce paginile pot fi inlaturate din memoria fizica, Linux utilizeaza tehnica Least Recent Used (LRU). Astfel, fiecarei pagini din memorie ii este asociata o varsta, ce se modifica la fiecare accesare a paginii. Cu cat ea este accesata mai des cu atat ea este mai “tanara”. Cu cat pagina este mai “ in varsta” cu atat devine un candidat mai bun pentru swapping. In cazul unui sistem de operare care foloseste paginarea pentru managementul memoriei, algoritmii de inlocuire a paginilor decid ce pagina din memorie sa scrie in swap sau ce sa stearga in momentul in care este nevoie ca memoria sa fie alocata altei pagini.

Calitatea unui algoritm este data de timpul in care se face inlocuirea. Cu cat timpul este mai scurt cu atat consideram algoritmul mai bun. Un algoritm de inlocuire a paginilor incearca sa determine ce pagina ar trebui sa fie inlocuita astfel incat sa aiba un timp de acces cat mai rapid si in asa fel incat acea pagina sa nu fie imediat folosita. Algoritmii pot fi locali sau globali. Atunci cand algoritmul selecteaza sa inlocuiasca o pagina in emoria locala a procesului cu o alta pagina din aceeasi memorie, algoritmul este de tip local. In cazul n care pagina aleasa din orice parte a memoriei, algoritmul este de tip global. lgoritmii locali partitioneaza memoria in asa fel incat sa se poata determina cate pagini sa fie ferite unui proces sau unui grup de procese. Cele mai cunoscute forme de partitionare sunt “fixed artitioning” si “balanced set”, algoritmi bazati pe modelul working set. Marele avantaj al acestor lgoritmi il reprezinta stabilitatea: fiecare proces isi poate face un management al paginilor ndependent de orice structura globala de date. Multi algoritmi returneaza doar un pointer catre pagina in care urmeaza sa fie scrisa noua pagina. Din pacate aceasta poate fi si “murdara”, caz in care durata scrierii noi pagini ar fi mult mai mare. De aceea in calculatoarele moderne exista ideea de precuratare. Acest mecanism incepe procesul de stergere asupra paginilor ce urmeaza a fi inlocuite. Ideea este aceea ca pana in momentul in care va fi nevoie de pagina respectiva din memorie , ea va fi deja curata. Acest algoritm trebuie sa determine ce pagini vor urma sa fie inlocuite, pentru a nu sterge in mod inutil.

First-in, First-out :

FIFO este cel mai simplu algoritm de inlocuire a paginilor.Este un algoritm care necesita cele mai putin resurse din partea sistemului de operare. Sistemul de operare creeaza o coada in functie de intrarea paginilor. In momentul in care este nevoie sa se incarce o noua pagina, se elimina primul loc din fata, toate paginile din coada fiind deplasate cu o pozitie la dreapta, ramanand un loc liber in partea stanga pentru noua

pagina. Acest algoritm nu este prea eficient motiv pentru care in practica este si foarte rar folosit.

O imbunatatire a algortmului FIFO este"FIFO with second chance.El verifica daca bitul de adresare R(referenced bit) este setat.Daca da,pagina este trecuta in capul cozii,altfel este mutat din memorie.

Exemplu FIFO:

```
int fifo_alg() {
int save_reg,page,frame,earliest_time;
  save_reg = FTVR;
  earliest_time = INFINITY;
  do {
    FTVR++;
    if (FTVR >= FTLR) FTVR = 0;
    page = FTBR->entries[FTVR].page;
    if (page == NIL) return(FTVR);
    if (PTBR->entries[page].first_time <
earliest_time) {
      earliest_time = PTBR-
>entries[page].first_time;
      frame = FTVR;
    }
  } while (FTVR != save_reg);
  FTVR = frame;
  return(FTVR);
}
```

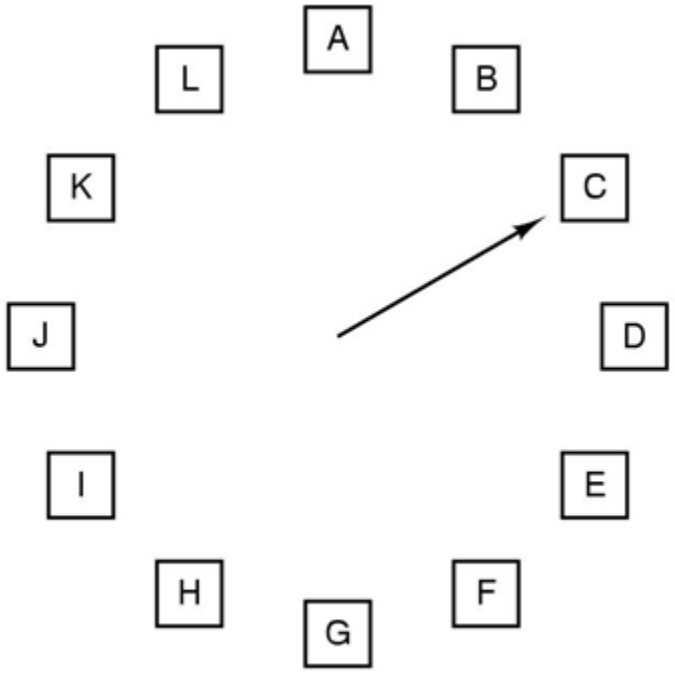
Algoritmul Not Recently Used (NRU)

Algoritmul Not Recently Used (NRU) ,favorizeaza pastrarea paginilor folosite recent. Principiul este urmatorul: cand se adreseaza o pagina i se atribuie un bit, bit de adresare. La fel cand se aduc modificari asupra paginii i se atribuie un nou bit, de modificare. La un anumit interval de timp se verifica bitii fiecarei pagini, si paginile se impart in urmatoarele categorii:
clasa0 : neadresata, nemodificata
clasa1: neadresata, modificata
clasa2: adresata, nemodificata
clasa3: adresata, modificata

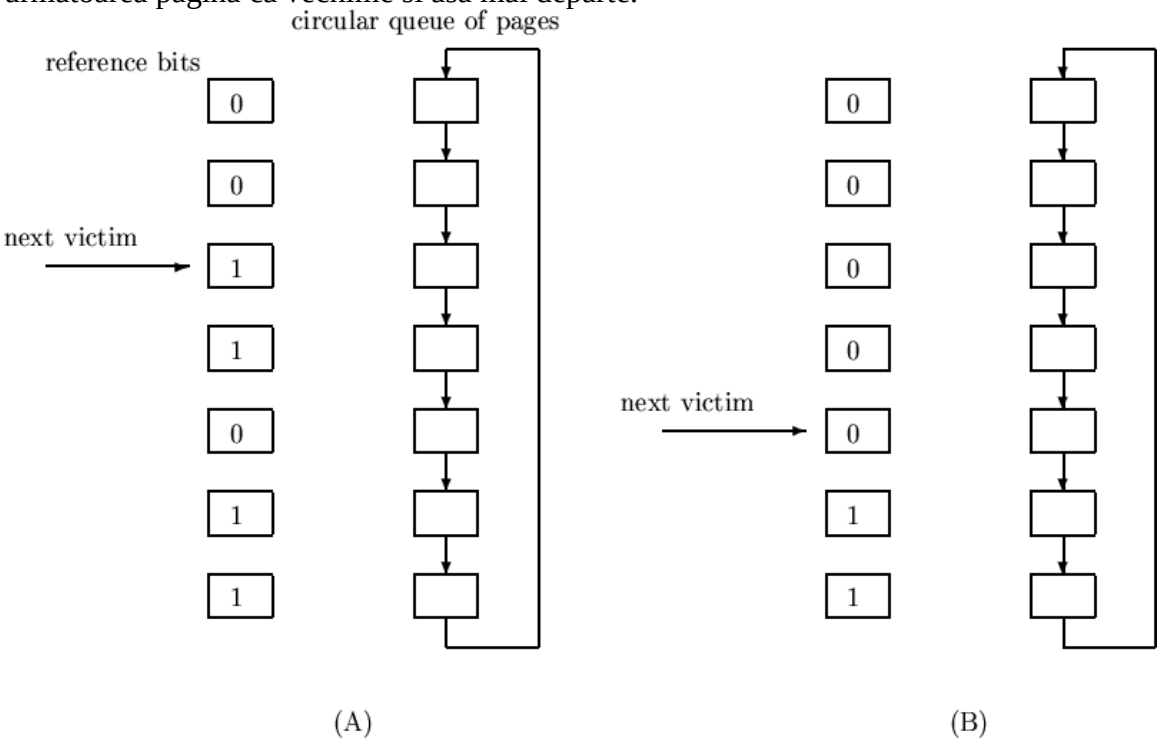
Algoritmul NRU alege la intamplare o pagina din clasele inferioare pentru inlocuire. In acest algoritm, o pagina adresata este mai importanta decat una modificata.

Clock

Cu toate ca “second chance” este un algoritm rezonabil,el este deseori ineficient pentru ca muta constant pagini in lista.O mai buna abordare este sa pastram paginile intr-o lista circulara in forma de ceas.Sageata arata spre cea mai veche pagina.



Cand este nevoie de o pozitie noua algoritmul inspecteaza intai cea mai veche pagina si in cazul in care bitul de adresare este 0 o inlocuieste. Daca nu, se trece la urmatoarea pagina ca vechime si asa mai departe.



Least Recently Used (LRU)

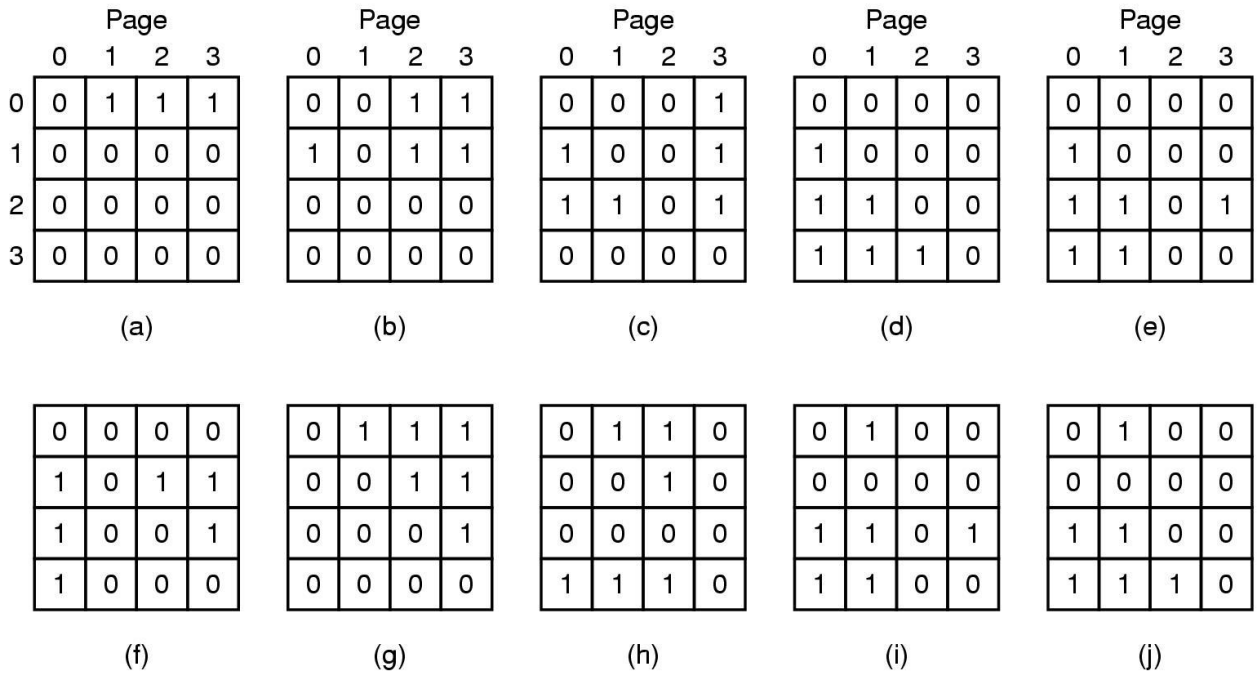
Paginile ce au fost foarte folosite in ultima perioada de timp (determinata) vor fi folosite mult si in viitor deci ele ar trebui pastrate. Teoretic acesta este unul dintre cei mai eficienti algoritmi numai ca este foarte costisitor.

Exista totusi cateva forme ale acestui algoritm care incearca sa reduca din costuri, pastrand in acelasi timp performanta. Cea mai costisitoare dintre acestea este metoda listelor inlantuite care dispune de un set de liste ce contin toate paginile din memorie.La capatul lor se afla pagina cel mai rar folosita. Costurile ridicate sunt date de faptul ca listele se vor muta la fiecare accesare a memoriei, lucru ce necesita foarte mult timp. Un mare defect al LRU este acela ca daca el gestioneaza la un moment dat N pagini iar o aplicatie lucreaza cu un loop de N+1 pagini, la ultima operatie va rezulta o eroare.

Versiunile derivate ale LRU, incerca, pe de-o parte sa rezolve problema costului ridicat si pe de alta parte sa elimine aceste erori.

Exemple:

LRU-K reprezinta o imbunatatire fata de LRU in ceea ce priveste consumul de timp. ARK reprezinta un LRU ce pastreaza si o istorie a pagnilor pe care le-a sters *Algoritmul RANDOM – aletor* Dupa cum spune si numele, paginile vor fi eliminate din memoria virtuala aletor. Acest algoritm elimina necesitatea determinarii peginii ce va fi eliminate. Pentru OS/390 ,atunci cand LRU nu mai face fata se apeleaza la RANDOM. Exemplu LRU cu matrici:



Pagini adresate in ordinea: 0,1,2,3,2,1,0,3,2,3

Not Frequently Used

Algoritmul NFU necesita un numarator si fiecarei pagini ii este asociat un contor, in starea initiala zero. Lafiecare perioada a ceasului paginile la care s-a facut referire sunt incrementate cu 1. Ca urmare, numaratorul stie exact cat de frecvent a fost utilizata o pagina si astfel, pagina cu contorul cel mai scazut poate fi inlocuita.
Marea problema a acestuia este ca tine cont de frecventa de accesare a paginii dar nu si de durata cat a fost ea folosita, el lovindu-se de o reala problema la boot-area unui sistem de operare. In acest caz, unele pagini sunt accesate n primul pas si folosite pe o durata lunga de timp, dupa care, in pasul al doilea, alte pagini sunt folosite frecvent insa pentru durate scurte de timp. La o cerere de inlocuire algoritmul va inlocui, deci o pagina ce a fost folosita la boot-are lucru ca va rezulta, desigur intr-o eroare.

Imbatranire(aging)

Urmasul algoritmului NFUeste Aging.Imbunatatirea adusa fiind ca in acest caz este luata in calcul si durata de folosire a unei pagini. In acest caz, in loc de o simla incrementare a contorului, el este intai mutat la dreapta cu o unitate. Daca bitii de referinta ai unei pagini arat asa pe durata a 6 perioade ale ceasului: 100110, atunci contorul a fost incrementat astfel: 10000000, 01000000, 00100000, 10010000, 11001000, 01100100.

Aceasta metoda face ca paginile folosite mai recent dar mai putin frecvent sa aiba o prioritate ami ridicata dcat paginile accesate frecvent in trecut. Cand este nevoie ca o pagina inlocuita se va allege pagina cu cel mai mic contor.

Problema acestui algoritm este ca poate monitoriza doar ultimele 16 sau 32 de intervale, astfel doua pagini pot avea contorul 00000000, chiar daca una a fost accesata cu 9 intervale in urma iar alta cu 100. Totusi, cunoasterea folosirii in ultimele 16 intervale este suficienta pentru o decizie aproape optima astfel, Aging fiind considerat un algoritm aproape perfect pentru un pret moderat.

4.Gestionarea memoriei fizice (*Moraru Diana*)

Exista trei cerinte de baza ale memoriei interne:

•Timpul de acces la memoria interna trebuie sa fie cat mai mic posibil; acest lucru serealizeaza prin proiectarea adecvata atat a componentei hardware cat si a celei softwareimplicate in gestiunea memoriei. Calculatoarele moderne folosesc memoria "cache", cereprezinta un tip de memorie cu acces foarte rapid si care contine informatiile cele mairecent utilizate de catre CPU.

•Dimensiunea memoriei adresabile trebuie sa fie cat mai mare posibil, ceea ce se poaterealiza prin conceptual de memorie virtuala.

•Memoria interna trebuie sa aiba un cost relativ scazut.

Functiile administratorului memoriei interne sunt:

•Alocarea de spatiu de memorie interna proceselor.

•Realizarea corespondentei dintre spatiul de adrese al procesului si locatii de memorieinterna.

•Minimizarea timpului de acces la locatiile de memorie

Realizarea acestor functii este conditionata atat de componenta hardware, cat si de cea software,continuta in SO.

Principalele obiective ale gestiunii memoriei sunt:

•calculul de translatare a adresei(relocare);
•protectia memoriei;
•organizarea si alocarea memoriei operative;
•gestiunea memoriei secundare;
•politici de schimb intre procese, memoria operativa si memoria secundara

4.1 Subsistemul de gestiune a memoriei fizice la Windows (*Moraru Diana*)

Subsistemul de gestiune al memoriei din cadrul unui sistem de operare este folosit de toate celelalte subsisteme: planificator, I/O, sistemul de fisiere, gestiunea proceselor, networking. Memoria este o resursa importanta, de aceea sunt necesari algoritmi eficienti de utilizare și gestiune a acesteia. Rolul subsistemului de gestiune a memoriei este de a tine evidenta zonelor de memorie fizica ocupate sau libere, de a oferi proceselor sau celorlalte subsisteme acces la memorie si de a mapa paginile de memorie virtuala ale unui proces peste paginile fizice.

Nucleul sistemului de operare ofera un set de interfete (apeluri de sistem) care permit alocarea/dezalocarea de memorie, maparea unor regiuni de memorie virtuala peste fisiere, partajarea zonelor de memorie. Din pacate, nivelul limitat de intelegere a acestor interfete si a actiunilor ce se petrec in spate conduc la o serie de probleme foarte des intalnite in aplicatiile software: memory leak-uri, accese invalide, suprascrieri, buffer overflow, corupere de zone de memorie.

Este, in consecinta, fundamentala cunoasterea contextului in care actioneaza subsistemul de gestiune a memoriei si intelegerea interfetei pusa la dispozitie de sistemul de operare programatorului.

In functionarea sistemului Windows trebuie avuta in vedere corelatia a trei componente: Kernel, User, GDI.

KERNEL este nucleul mediului Windows. Rolul lui este de a rezolva toate sarcinile elementare ale unui sistem de operare: incarca programele si realizeaza administrarea memoriei, regleaza impartirea timpului de calcul intre aplicatii, controleaza task-urile distincte. Programele KRNL386.EXE si KERNEL32.DLL constituie nucleul sistemului de operare, oferind suportul pentru functiile la nivelul jos de care are nevoie o aplicatie pentru a rula. Kernel nu este responsabil cu functiile de intrare/iesire si de interfata cu utilizatorul, proprii unui sistem de operare. Denumirea de nucleu al sistemului de operare provine din faptul ca el interactioneaza numai cu Windows.

USER prin componentele sale USER.EXE si USER32.DLL nu se refera la utilizator, ci la comanda tuturor ferestrelor si administrarea acestora: continutul de informatie al ferestrei, structura de baza a lor, toate informatiile din meniuri si submeniuri. In afara ferestrelor modulul USER se ingrijeste si de alte elemente cum sunt casetele de dialog sau structurile de control apelabile prin butoane sau combinatii de taste. In acelasi timp, USER se ocupa cu incarcarea driverelor, perifericelor, supravegherea comunicatiei intre task-urile distincte, a ferestrelor, a iconurilor si aplicatiilor; realizeaza comanda cursorului si iconurilor. In cele din urma acesta gestioneaza intr-o oarecare masura resursele aplicatiilor: iconuri, meniuri, cimpuri de dialog pe care Windows le retine in memoria RAM.

GDI (graphic device interface) prin GDI.EXE si GDI32.DLL asigura toate afisarile pe ecran, imprimanta sau orice alt periferic contribuind la realizarea unei legaturi intre Windows si mediul exterior. Se administreaza procesul grafic la modul general, independent de dispozituvul folosit pentru afisare.

Atit pentru modulul USER, cit si pentru cel GDI, exista resurse de memorii speciale; daca ele sunt depasite atunci sistemul se poate bloca.

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 partitiilor 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 partitiile 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.

Procesul de partitionare fixa are multe neajunsuri. Datorita partitiilor fixe nu conteaza care dintre ele este utilizata. In cazul in care avem partitii 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 partitie pentru ca la un moment dat e posibil ca mai multe procese de dimensiuni apropiate sa doreasca aceiasi partitie. Avantajele constau in faptul ca este minimizata cantitatea de memorie irosita. Dar aceasta modalitate este optima din punct de vedere al unei partitii dar nu si din punctul de vedere al intregului sistem. Astazi acest mod de organizare nu mai este folosit, face parte doar din istoria calculatoarelor si a sistemelor de operare.

In cadrul partitionarii dinamice, partitiile sunt de lungime si numar variabile.Fiecare proces are alocata exact atata memorie cat are nevoie.Dar, metoda 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. 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:

- Algoritmul prima potrivire (first fit):

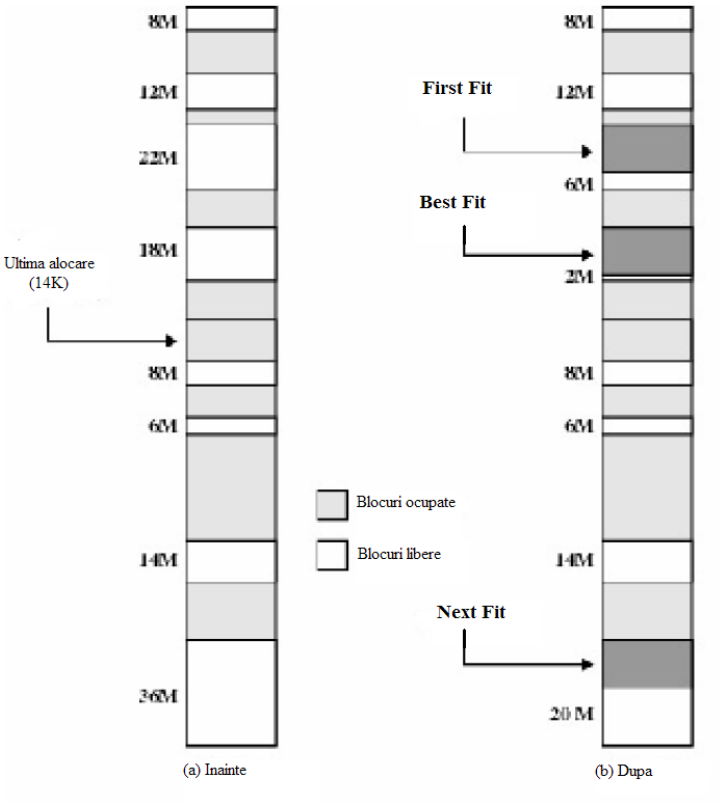
-Gestionarul de memorie cauta in lista de segmente un spatiu liber suficient de mare
-Spatiul este spart in doua parti, una pentru proces iar cealalta va ramane ca memorie nefolosita, doar daca nu se potriveste exact

- Algoritmul urmatoarea potrivire (next fit):

- Memoreaza pozitia la care a gasit un spatiu suficient de mare
-La urmatoarea cautare va incepe sa caute in lista de la pozitia la care a terminat inainte, in loc sa inceapa mereu de la inceput ,asa cum face mereu algoritmul First fit
-Este un pic mai lent decat algoritmul First fit

- Algoritmul cea mai buna potrivire (best fit):

-Cauta in toata lista si alege cel mai mic spatiu liber care este potrivit
-Este mai lent
-Creaza multe spatii mici ceea ce duce la fragmentarea memoriei



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.

4.2 Algoritmul prietenului la Linux (*Moraru Diana*)

Metoda alocarii prin prieteni (Buddy-system) se bazeaza pe reprezentarea binara a adreselor si faptul ca dimensiunea memoriei interne este un multiplu al unei puteri a lui 2. Presupunem ca sedimensiunea memoriei interne este de forma c x 2^n, iar unitatea de alocare a memoriei este de forma 2^m.

Daca sistemul are o memorie interna de 32 Mo, atunci c=1 si n=25. Daca dimensiunea memoriei interne este de 192 Mo, c=3 si n=26. De asemenea, se poate considera ca unitatea dealocare este de 256 Ko, adica 2^8.

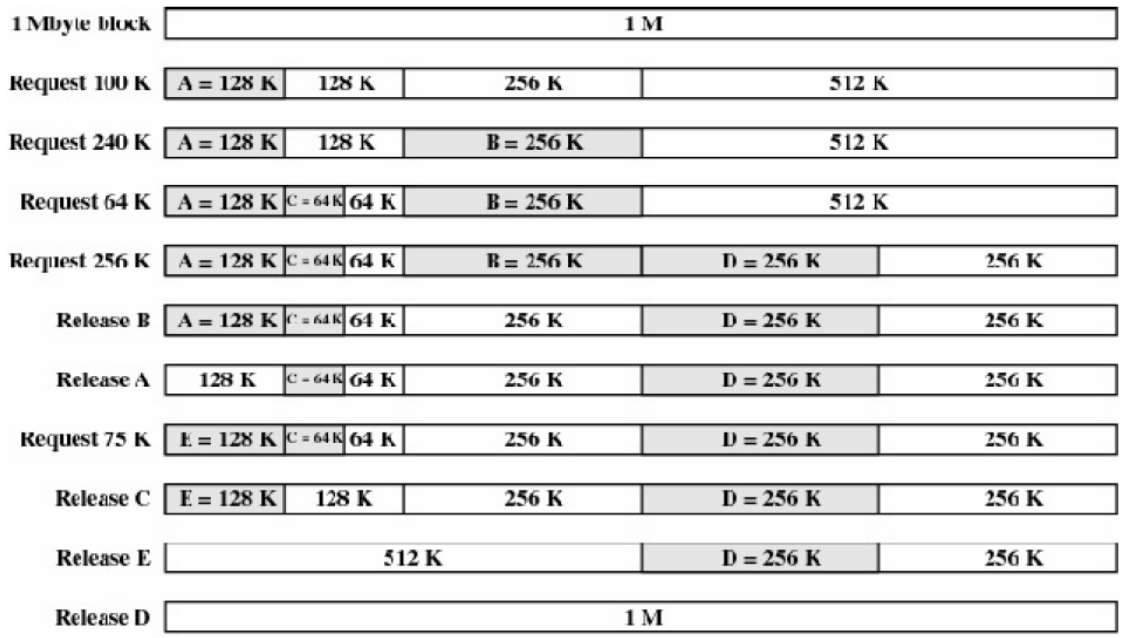
Tinand cont de proprietatile operatiilor cu puteri ale lui 2, atat dimensiunile spatiilor alocate, cat siale celor libere sunt de forma 2^k, cu k≤m≤n. In concluzie, sistemul va pastra liste separate aleadreselor spatiilor disponibile, in functie de dimensiunea lor exprimata ca putere a lui 2. Se numeste lista de ordin k, lista tuturor adreselor unde incep spatii libere de dimensiune 2^k. Vor exista astfel n-m+1 liste de spatii disponibile.

Daca consideram ca dimensiunea memoriei interne este de 192 Mo, vom avea 17 liste: lista de ordin 8, avand dimensiunea unui spatiu de 256 octeti; lista de ordin 9, cu spatii dedimensiune 512 s.a.m.d.Presupunem ca, fiecare spatiu liber(ocupat) de dimensiune 2^k, are adresa de inceput un multiplu de 2^k.

Doua spatii libere se numesc prieteni de ordinul k, daca adresele lor A1 si A2 verifica una dintre proprietatile urmatoare:
A1<A2, A2=A1+ 2^k si A1 mod 2^{k+1}=0
A2<A1, A1=A2+ 2^k si A2mod 2^{k+1}=0

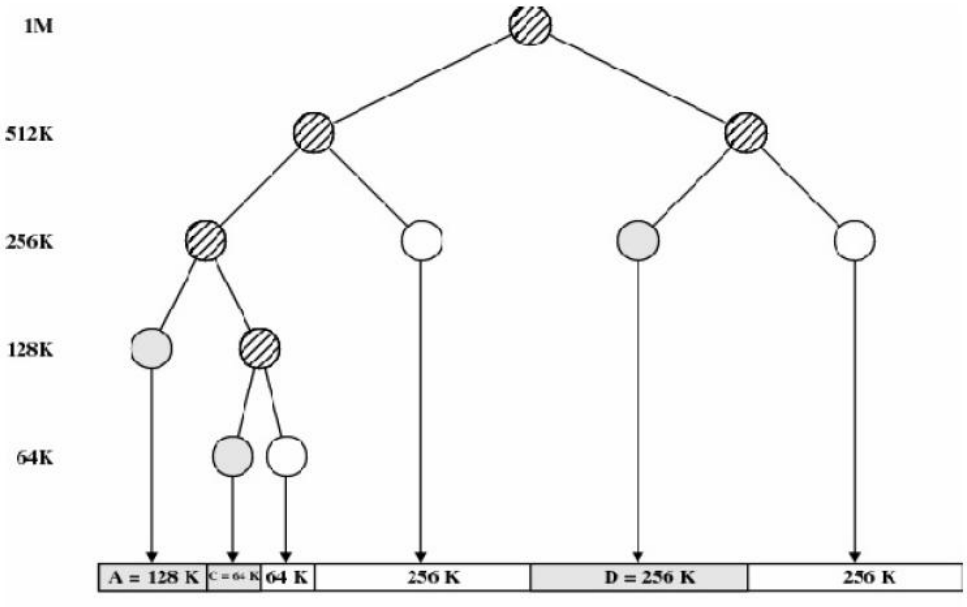
Aceasta definitie, exprima o proprietate fundamentala: Atunci cand intr-o lista de ordinul k apar doiprieteni, sistemul ii concateneaza intr-un spatiu de dimensiune 2^{k+1} si reciproc, un spatiu dedimensiune 2^{k+1} se poate imparti in doua spatii de dimensiune 2^k.

Exemplu



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 urmatoare. Frunzele arborelui corespund partitionarii curente a memoriei. Daca 2 parti de memorie sunt
frunze inseamna ca cel putin una trebuie alocata. In caz contrar ele vor fi unite intr-un bloc mai mare.



Algoritmul de alocare de memorie.

Pas 1. Fie o numarul de octeti solicitati. Se determina $\min\{p / m \le p \le n, o \le 2^p\}$.
Pas 2. Se determina $k = \min\{i / p \le i \le n$ si lista de ordin i este nevida$\}$.
Pas 3.Daca k=p, atunci aceasta este alocata si se sterge din lista de ordinul p altfel se aloca primii 2^p octeti, se sterge zona din lista de ordinul k si se creeaza in schimb alte k-p zone libere, avand dimensiunile 2^p, 2^{p+1},..., 2^{k-1}.

Pasul 3 ai algoritmului se bazeaza pe egalitatea: $2^k - 2^p = 2^p + 2^{p+1} + ... + 2^{k-1}$.

De exemplu, se doreste alocarea a 1000 octeti, deci p=10. Nu s-au gasit zone libere nici de dimensiunea 2^{10}, nici 2^{11} si nici 2^{12}.Prima zona libera de dimensiune 2^{13}. are adresa de inceput 5x 2^{13} si o notam cu I. Ca rezultat al alocarii a fost ocupata zona A de dimensiune 2^{10} si au fost create incatrei zone libere: B de dimensiune 2^{10}, C de dimensiune 2^{11} si D de dimensioune 2^{12}.

Zonele B, C si D se trec respectiv in listele de ordine 10, 11 si 12, iar zona I se sterge din lista de ordin 13.

Algoritmul de eliberare.

Pas 1. Fie 2^p dimensiunea zonei eliberate. Se introduce zona respectiva in lista de ordinul p.
Pas 2.k=p
Pas 3.Verifica daca exista prieteni de ordin k:Daca da, efectueaza comasarea lor; Sterge cei doi prieteni;Introdu noua zona libera de dimensiune 2^{k+1} in lista de ordin k+1.
Pas 4. k=k+1; go to Pas 1.

Sa presupunem, de exemplu ca la un moment dat zonele A, C si D de dimensiuni $5x\,2^{13}$,respectiv $21x\,2^{11}$ si $11x\,2^{12}$ sunt libere, iar zona B de dimensiune $41x\,2^{11}$ este ocupata, zonele fiind adiacente, in ordinea A, B, C, D. In conformitate cu pasii descrisi mai sus, se executa urmatoarele actiuni:

-Se trece zona B in lista de ordin 10.

-Se observa ca zonele A si B sunt prieteni. Drept urmare, cele doua zone sunt comasate si formeaza o noua zona X. Zona X se trece in lista de ordin 11, iar zonele A si B se sterg din lista de ordin 10.

-Se observa ca zonele X si C sunt prieteni; ele sunt comasate si formeaza o zona Z care se trece inlista de ordin 12, inlocuind zonele X si C din lista de ordin 11.

-Se observa ca Z si D sunt prieteni; ele sunt sterse din lista de ordin 12, iar in lista de ordin 13 se introduce rezultatul comasarii lor.

Bibliografie generala:

Sarbu Lucia-Ioana:

"Modern Operating Systems", Andrew Tanenbaum
"Understanding The Linux Virtual Memory Manager", Mel Gorman
http://www.linux-tutorial.info/index.php
http://www.intellectualheaven.com/Articles/WinMM.pdf
http://webster.cs.ucr.edu/AoA/Windows/HTML/MemoryArchitecturea3.html
http://www.brokenthorn.com/Resources/OSDev18.html

Serban Andrei-Gabriel:

http://en.wikipedia.org/wiki/Page_replacement_algorithm
http://www.liralab.it/teaching/OS/files/sample-4.pdf
http://www.sci.csueastbay.edu/~billard/cs4560/node15.html
http://www.ics.uci.edu/~bic/courses/JaverOS/pr4.pdf

Moraru Diana:

Sisteme de Operare Moderne, Andrew Tanenbaum
AWeighted Buddy Method for Dynamic Storage Allocation :Kenneth K. Shen and James L. Peterson,Digital Systems Laboratory Stanford University :
http://delivery.acm.org/10.1145/370000/361164/p558-shen.pdf?ip=84.232.208.192&acc=ACTIVE%20SERVICE&CFID=82489288&CFTOKEN=82817049&__acm__=1337010718_21edd6ae3b18085c7179ccc5311dc98b
http://www.linux-tutorial.info/modules.php?name=MContent&pageid=308
http://elf.cs.pub.ro/so/wiki/laboratoare/laborator-04
http://www.scribd.com/doc/13597643/Sisteme-de-Operare-
http://www.scritube.com/stiinta/informatica/GESTIUNEA-MEMORIEI32136.php