Implementarea gestiunii memoriei

Cuprins :

1. Introducere

1.1 Memoria Virtuala

Cu mai multi ani in urma ne-am confruntat pentru prima data cu programe care erau mai mari decat memoria fizica disponibila. Solutia gasita a fost impartirea programelor in bucati numite “overlays”. Se incepea cu overlay 0 si cand acesta se termina se contiuna cu altul si altul. Anumite sisteme mai complexe permiteau aflarea in memorie a mai multe overlay-uri in acelasi timp. Ele se aflau pe disk si cand era nevoie de ele erau incarcate sau descarcate din memoria fizica.

Desi incarcarea si descarcarea din memorie se facea automat de catre calculator, impartirea programelor trebuia facuta de catre programator. Munca de impartire era obositoare si plictisitoare si astfel s-a conceput un mod in care calculatorul sa faca toata munca de impartire si incarcare cand este nevoie.

Aceasta metoda s-a numit Virtual Memory. Principiul de baza al acestei metode era ca totalitatea memoriei ocupate de program putea depasi memoria fizica. Memoria fizica impreuna cu ceva spatiu pe disk erau ca o memorie mare, dar intr-un anumit moment in memoria fizica se afla doar o parte din program, restul asteptand pe disc sa fie incarcat in memoria fizica. Avantajul memoriei virtuale era ca toate aceste operatii erau facute de sistemul de operare.

Memoria virtuala se poate folosi si in sisteme “multitasking”, in care in memoria fizica sunt parti din diferite programe si pe disc sunt restul partilor care asteapta sa fie incarcate. De exemplu, in timpul in care un program asteapta o intrare de la tastatura alt program se executa fara ai fi simtita prezenta.

(*1)Pais Mihail Nicanor

Ref: “Modern Operating Systems 2nd”, Andrews S. Tanenbaum

http://www.linux-tutorial.info/modules.php?name=MContent&pageid=322

1.2 Paginarea

Majoritatea sistemelor cu memorie viruala folosesc un sistem numit “Paginare”. In fiecare calculator exista un set de adrese de memorie pe care programele pot sa le genereze. Cand un program da o comanda de punere a unei valori in memorie se genereaza o adresa numita “adresa virtuala” care face parte din spatiul adreselor virtuale. Daca memoria virtuala nu este disponibila atunci adresa este pusa direct pe bus-ul 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 bus-ul de memorie ci este trimisa la MMU (Unitatea de gestiune a memoriei) care transforma adresa virtuala in adresa fizica cum ne este aratat in figura urmatoare.

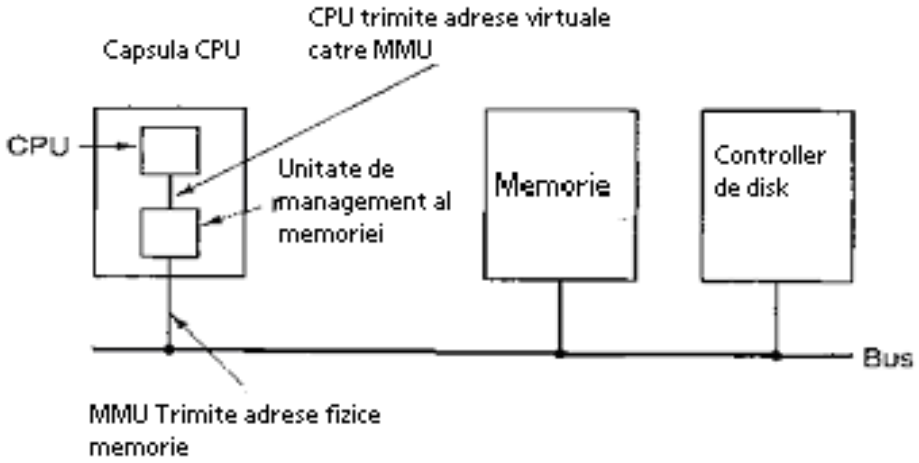


Fig.1

Ref: “Modern Operating Systems 2nd”, Andrews S. Tanenbaum

Pentru a exemplifica mai bine vom lua urmatorul caz: avem un calculator ce poate genera 16 biti de adrese virtuale (de la 0 la 64 Kb). Memoria fizica are numai 32 Kb asa ca nu pot fi translatate toate adresele virtuale in adrese fizice in acelasi timp. O copie a programului de 64 KB trebuie sa se afle mereu pe disc pentru ca bucati din el sa se aduca in memoria fizica atunci cand este nevoie de ele.

Spatiul adreselor virtuale este impartit in unitati numite “pagini”. Unitatile corespunzatoare in memoria fizica se numesc “page frames”. Paginile si “page frame-urile” sunt mereu de aceeasi dimensiune. Cand un program cere o valoare din memorie de fapt cere o adresa virtuala, care este trimisa la MMU, care la randul ei este transformata in adresa fizica si acea data de la adresa fizica este pusa pe bus-ul de memorie. Memoria fizica nu stie nimic despre MMU, ea doar primeste comanda de a pune o valoare dintr-o celula pe bus-ul de memorie.

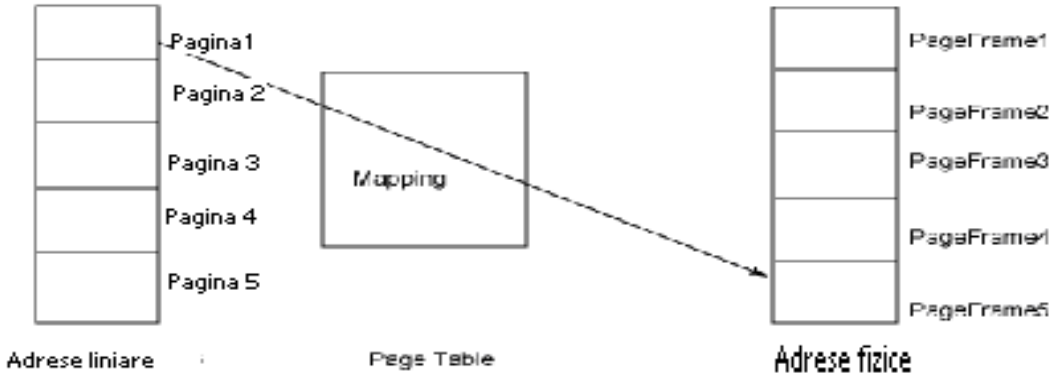
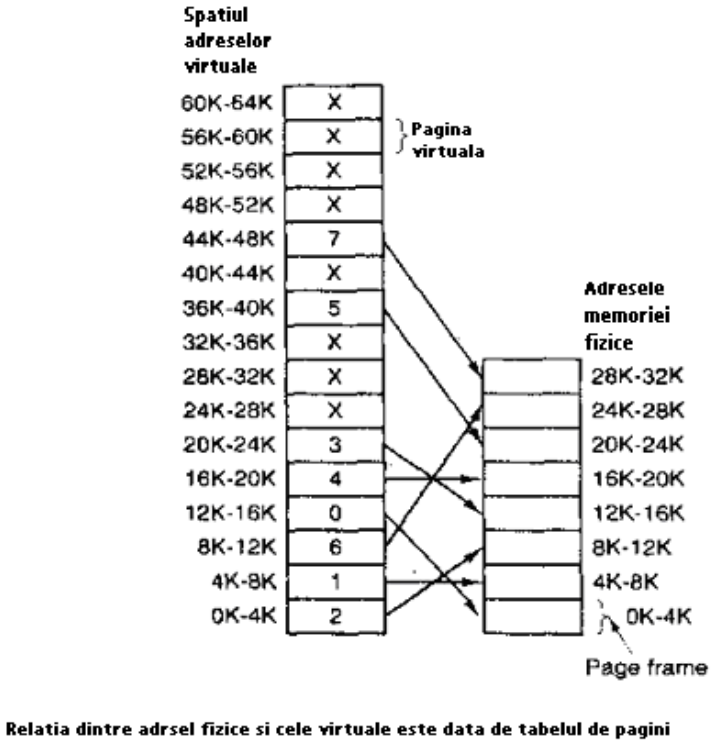


Fig.2

http://www.ibm.com/developerworks/linux/library/l-memmod/



Relatia dintre adrsel fizice si cele virtuale este data de tabelul de pagini

Fig.3

Ref: "Modern Operating Systems 2nd", Andrews S. Tanenbaum

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.

In orice moment avem doar 8 adrese virtuale legate de memoria fizica, restul fiind marcate ca neavand o adresa fizica. Daca un program acceseaza o astfel de locatie de memorie care nu are

corespondenta in memoria fizica atunci se da un mesaj sistemului de operare (numit “page fault”) care ia continutul unei locatii de memorie fizica nefolosita, o copiaza pe disc si apoi aloca noul spatiu creat adresei virtuale cerute de program. Daca se acceseaza o valoare ce nu are corespondenta in memoria fizica si nici nu exista pe disc, atunci inseamna ca acea pagina nu mai exista sau a aparut o eroare si atunci acea intrare va fi eliminata din memorie.

(*1)Pais Mihail Nicanor

Ref: “Modern Operating Systems 2nd”, Andrews S. Tanenbaum

http://www.linux-tutorial.info/modules.php?name=MContent&pageid=89

http://en.wikipedia.org/wiki/Paging

Paginarea mai detaliata :

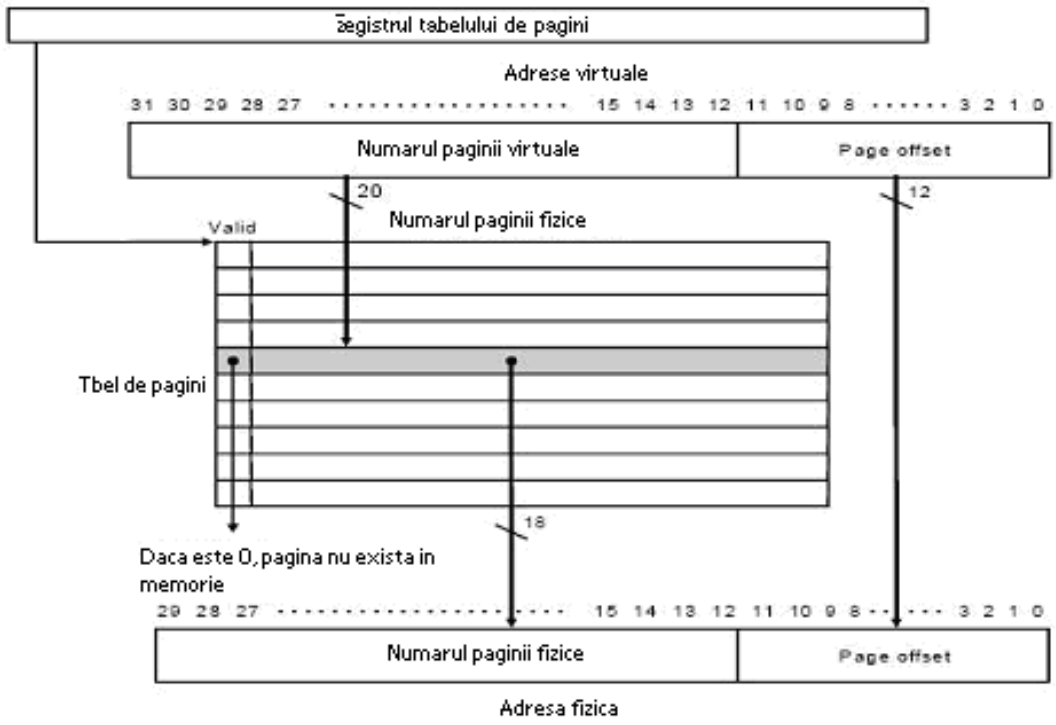


Fig.4

http://webdizen.new21.net/blog/1857?TSSESSION=524f8d2e136dad187840a1765e872e3a

2 Swaping si algoritmi de inlocuire a paginilor (Windows si Linux):

Daca un process are nevoie sa incarce o pagina virtuala in memoria fizica si nu exista pagini fizice libere, sistemul de operare trebuie sa elibereze o pagina din memorie, renuntand la una activa.

Daca, spre exemplu, pagina la care se va renunta provine dintr-o imagine sau un fisier asupra caruia nu s-au facut modificari pagina nu trebuie sa fie salvata ci pur si simplu se elibereaza, data viitoare cand este nevoie de ea fiind incarcata din nou din fisierul original.

Data insa asupra paginii s-au adus modificari, atunci sistemul de operare trebuie sa salveze intr-un fel aceasta pagina modificata pentru a o putea accesa mai tarziu. Acest fel de pagina se numeste “dirty page”. Atunci cand aceasta este stearsa din memorie, ea este salvata intr-un fisier special numit swap file.

Daca algoritmul de swap, care se foloseste pentru a decide ce pagina sa fie stearsa si ce pagina sa fie salvata, nu este eficient atunci are loc un eveniment numit “thrashing”. In acest caz paginile sunt scrise si citite in continuu din memorie lucru ce face sistemul de operare sa devina greu de apelat.

Setul de pagini pe care le foloseste un proces se numeste “working set”, iar un swap eficient asigura faptul ca toate procesele au “working set”-ul complet in memoria fizica.

Pentru a determina paginile care 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 atunci cand 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 din memoria locala a procesului cu o alta pagina din aceeasi memorie, algoritmul este de tip local. In cazul in care pagina aleasa din orice parte a memoriei, algoritmul este de tip global.

Algoritmii locali partitioneaza memoria in asa fel incat sa se poata determina cate pagini sa fie oferite unui proces sau unui grup de procese. Cele mai cunoscute forme de partitionare sunt “fixed partitioning” si “balanced set”, algoritmi bazati pe modelul working set. Marele avantaj al acestor

algoritmi il reprezinta stabilitatea: fiecare proces isi poate face un management al paginilor independent 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.

(*3)Pandia Cristian

Andrew S. Tanenbaum : Modern Operating Systems(Second Edition)

Abraham Silberschatz, Peter Baer Galvin, Greg Gagne : Operating Systres Concepts

Algoritmul optim de inlocuire a paginilor (teoretic)

Acest algoritm este cunoscut si ca OPT si functioneaza in felul urmator: cand o pagina din memorie are nevoie sa fie folosita, sistemul de operare o pune in locul paginii de care va fi nevoie cel mai tarziu in viitor. Deci in va fi in locuita sa zicem o pagina de care va mai fi nevoie abia peste 10 secunde si nu una care va fi ceruta din nou in numai 4 secunde.

Din pacate acest algoritm nu poate fi implementat intrucat este imposibil sa se determine exact cand va fi nevoie de o pagina din nou.

(*3)

Algoritmul Not Recently Used (NRU)

El pastreaza in memorie paginile care au fost folosite cel mai 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.

(*3)

First-in, First-out (FIFO)

Este unu algoritm care necesita un “effort” minim din partea sistemului de operare in ceea ce priveste notarea paginilor. 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.

(*3)

Second Chance

Este o forma imbunatatita a lui FIFO. Acesta verifica primele pagini din coada dar stabileste daca si bitul de adresare a fost atribuit. Daca el nu a fost atribuit, elimina pagina, dar in caz contrar bitul este sters si pagina este mutata la capatul celalalt al cozii ca si cum ar fi o pagina noua. Procedura se repeta pana cand se gaseste o pagina ce poate fi eliminata.

(*3)

Clock

Este o varianta mai eficienta a FIFO decat Second Chance intrucat paginile nu trebuie sa fie mutate constant in celalalt capat al cozii, dar functia pe care o indeplineste este aceeasi. Acest algoritm tine o lista circulara a paginilor din memoria virtuala cu un bit de adresare in dreptul fiecarei pagini. Algoritmul are mereu un pointer catre cea mai veche pagina din lista. 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.

(*3)

Least Recently Used (LRU)

Principiul acestuia este ca 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 mia 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.

(*3)

Not Frequently Used

NFU necesita un numarator si fiecarei pagini ii este asociat un contor, in starea initiala zero. La fiecare 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.

(*3)

Aging

Este urmasul algoritmului NFU 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 metodaface 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 alege 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.

(*3)Pandia Cristian

Andrew S. Tanenbaum : Modern Operating Systems(Second Edition)

Abraham Silberschatz, Peter Baer Galvin, Greg Gagne : Operating Systres Concepts

www.cs.utexas.edu

www.informit.com

http://www.vishalgupta.co.uk/oracle/ixora/www.ixora.com.au/notes/paging.htm

3 Algoritmi de gestiune a memoriei

In continuare vom vedea cativa algoritmi ca sa furnizeze un nou page frame atunci cand este nevoie. Se foloseste simbolul “N” pentru a reprezenta numarul de pagini de memorie.

3.1 Bitmap

Bitmap-ul este, de fapt, o matrice de N/8 bytes este folosita ca o harta mare pentru indicarea folosirii memoriei.

Fiecare intrare in Bitmap contine o valoare pe unul sau mai multi biti ce reprezinta starea de alocare a blocului de memorie corespunzator acelei intrari.

In cea mai simpla forma intrarea este pe un bit avand valoarea 0, daca blocul de memorie este liber si 1 daca blocul de memorie este folosit de un proces. De exemplu, la un sistem u memorie de 256 MB avem nevoie de un Bitmap de dimensiune 8 192 Bytes pentru a gestiona toate cele 65 536 de pagini.

Intr-o varianta mai avansata intrarea este o valoare pe doi biti, reprezentand una din cele 3 stari posibile:

- „liber" – starea corespunzatoare blocului de memorie nealocat (nefolosit);
- „sub-alocat" – starea corespunzatoare blocului de memorie care este el insusi un set alocat de mai multe blocuri;
- „alocat" – starea corespunzand blocului de memorie folosit de un proces.

(*1)Pais Mihail Nicanor

http://www.patentstorm.us/patents/6175900-claims.html

Avantajul folosirii Bitmap-ului este reprezentat de implementarea sa simpla (alocarea spatiului pentru o matrice „O", parcurgerea si modificarea usoara a valorii starii blocului de memorie, de ex. : O(1,2) = 1).

Dezavantajele sunt urmatoarele:

- trebuie parcurs intreg Bitmap-ul la fiecare alocare a unui bloc de memorie

- pentru a aduce spre folosinta un numar K de blocuri alaturate de memorie, manager-ul de memorie, trebuie sa gaseasca o sectiune continua de K intrari in Bitmap. Toate acestea inseamna mult timp.

Insa, folosirea unui pointer care sa indice ultima intrare a unui bloc alocat, ar putea imbunatati performatele cautarii (astfel se detine informatia ca intrarile anterioare au fost cautate si sunt deja alocate).

(*1)Pais Mihail Nicanor

http://www.osdev.org/osfaq2/index.php/Algorithms%20and%20Tips%20for%20Memory%20Management

3.2 Stack/List of pages

O alternativa la Bitmap (care este folosita si de Linux si de Windows) este folosirea unei structuri dinamice de tip *stiva de pagini* (sau *„stack of pages")*. Adresele (fizice) ale paginilor libere sunt puse in stiva, iar cand o pagina trebuie alocata adresa urmatoare este scoasa din stiva si folosita.

Cand o pagina este eliberata, adresa sa este pusa inapoi in stiva, si tot asa. Astfel, o alocare (sau de-alocare) devine o problema de incrementare (sau decrementare) a unui pointer.

Desi multe alocari nu necesita ca adresele fizice sa fie alaturate, exista si astfel de cazuri cum ar fi DMA. Daca este nevoie ca adresele sa fie alaturate, atunci managerul de memorie trebuie sa scoata adresele din mijlocul stivei, operatie ceva mai complicata.
Avantajul acestei metode este ca alocarea si de-alocarea sunt rapide, in schimb verificarea starii unei pagini nu este deloc practica. In plus, daca toata memoria ar fi libera ar fi nevoie de N **x** 4 bytes pentru a stoca toate starile.

(*1)Pais Mihail Nicanor

Ref: Tim Robinson - „Memory Management 1"

3.3 Scheme hibride

O alta alternativa ar fi folosirea unor scheme hibride dupa cum urmeaza:

1. Alocatorii (cei care aloca memoria) pot fi inlaintuiti astfel incat in stiva sa fie retinute doar ultimele operatii, iar sfarsitul ei sa fie legat de Bitmap, pentru o buna compacitate. Daca stivei ii lipsesc pagini, se poate scana Bitmap-ul pentru a le gasi.

2. A doua varianta ar fi ca in loc sa memoram biti reprezentand pagina, sau numere intr-o stiva, sa folosim o matrice de structuri pentru a reprezenta memoria. Aceste structuri de page frame-uri stocheaza o singura legatura catre o pagina urmatoare (de marimea unui pointer) si un block de informatie de 8-16 biti, care sa indice starea paginii. Algoritmul include, de asemenea, un pointer catre o pagina virtuala si TCB-ul fiecarui numar de pagina. Se pastreaza pointeri pentru fiecare tip de pagina, care sa indice atat inceputul cat si sfarsitul listelor. In acest fel se poate afisa cu usurinta informatia despre continutul lor, numarul de pagini pentru fiecare tip, tipuri mixe. De asemenea, se permite o curatare dinamica, se permite un „copy-on write" destul de usor si se pastreaza clar si concis overview-urile paginilor. Este la fel ca la un tabel de mapare invers, care listeaza si tipurile de pagina.

(*1)Pais Mihail Nicanor

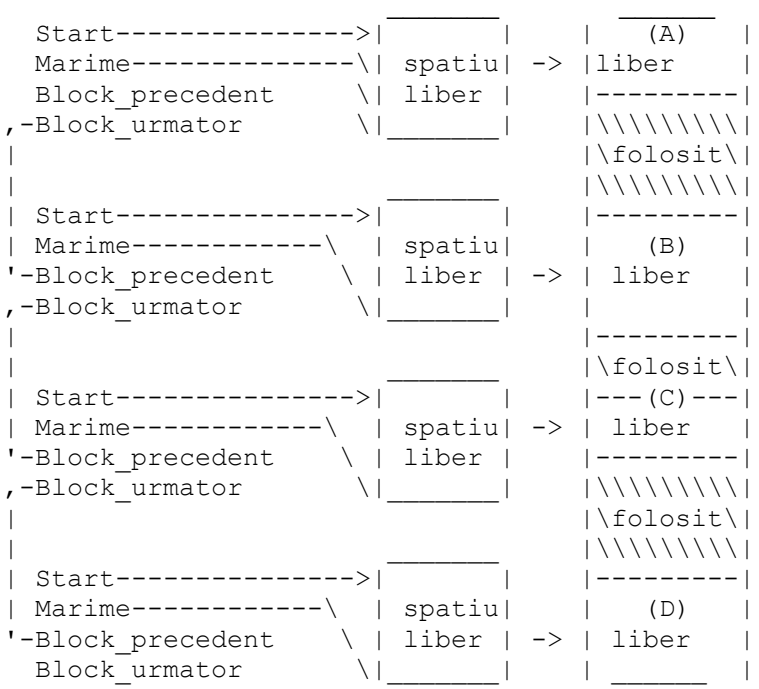
3.4 Flat List

O buna maniera de a gestiona zone mari ale spatiului adreselor este folosirea unei *liste inlantuite* – *„linked-list".* Fiecare regiune „libera" (nealocata) este asociata cu un descriptor ce indica marimea si adresa ei de baza (inceput). Cand un anumit spatiu trebuie alocat, lista este scanata dupa un algortim *„first fit"*sau *„best fit"* sau un alt algoritm corespunzator.

Cand memoria este alocata, intrarea respectiva este ori inlaturata (daca intreaga regiune a fost alocata), ori redimensionata (numai o poritune din regiune a fost alocata).
Un dezavantaj ar fi ca in cazul *„linked-lists"*, pentru a raspunde la intrebari de tipul : „este memoria la adresa XXX libera ?" sau „ unde pot gasi un bloc de dimensiunea YYY ?", trebuie parcursa completa lista.
Iar daca memoria este fragmentata si lista se mareste, aceasta ar putea deveni o problema.

Intrebarea „este memoria la adresa XXX libera ?" este folosita, mai ales, pentru a uni doua zone libere intr-una mai mare si este si mai usor de utilizat daca lista este ordonata crescator dupa adrese.

Exemplu de lista inlantuita :

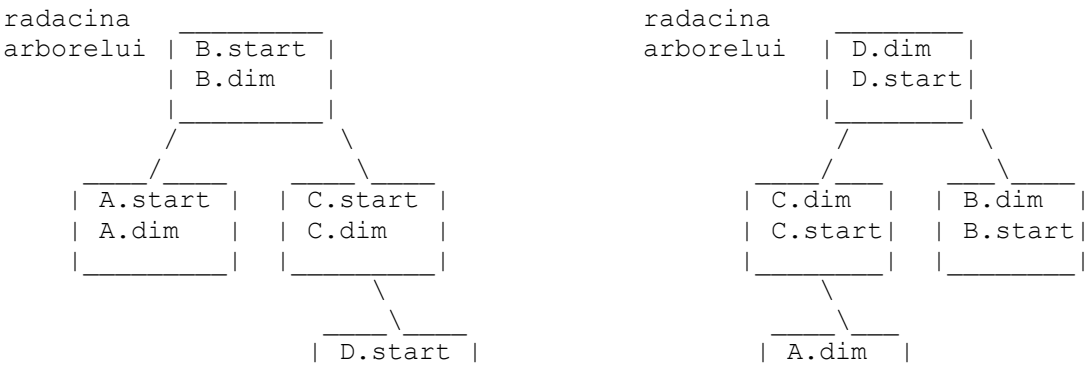


(*2)Herea Florentina

3.5 Arborele AVL

Din moment ce a devenit frecventa cautarea in lista a unei anumite adrese sau a unui spatiu de memorie de o anumita dimensiune, o solutie ar fi utilizarea unor structuri de date mai eficiente. Una din structuri care pastreaza inca parcurgerea intregii liste este *„AVL Tree" („Arborele AVL")*. Fiecare nod din acest arbore va descrie o regiune de memorie si are un pointer catre *subarborii (subtrees)* nodurilor inferioare si catre *subarborii (subtrees)* nodurilor superioare.

Exemplu de Arbore AVL



Desi inserarea sau stergerea intr-un astfel de arbore necesita operatii mai complexe decat o simpla manipulare a listei, parcurgerea arborelui se face cu $\log_2(N)$ iteratii, in loc de N iteratii la liste inlantuite – asa ca daca avem 100 de intrari este nevoie doar de 10 iteratii, in loc de 100 pentru a gasi un interval cautat.

Observatie importanta : algoritmul se folosete pentru *regiuni* (ca acelea pe care le gsim in /proc/xxx/maps) si nu pentru interfata de tip *malloc*.

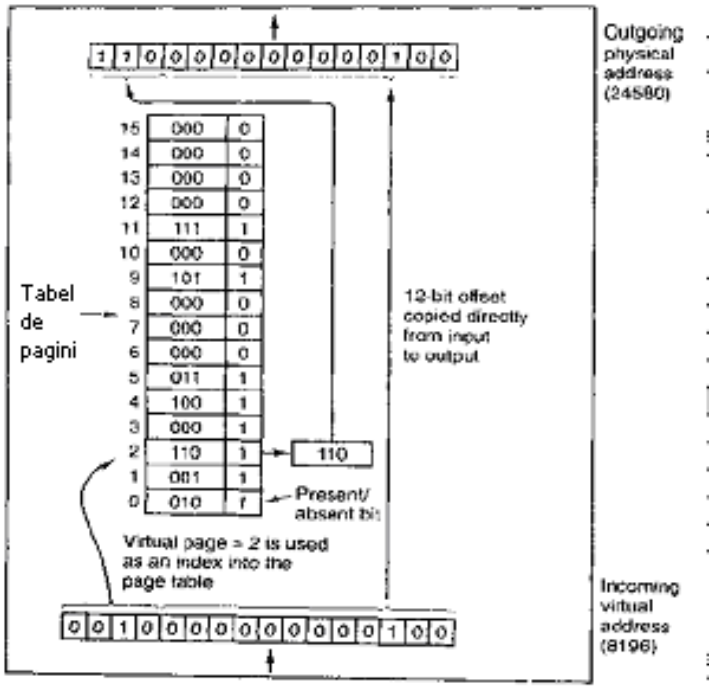
(*2)Herea Florentina

http://www.osdev.org/osfaq2/index.php/Algorithms%20and%20Tips%20for%20Memory%20Management

4 Implementarea gestiunii memoriei la Windows

4.1 Tabela de pagini (page table)

In cel mai simplu caz, maparea adreselor virtuale in adrese fizice este exact cum am descris-o mai sus. Adresa virtuala este impartita in doua : bitii high-order, care contin numarul paginii virtuale si bitii low-order, in care este retinut offset-ul. Pentru o adresa de 16 biti si o pagina de 4 KB, primii 4 biti pot specifica una din cele 16 pagini si urmatorii 12 biti ar specifica offset-ul byte-ului (de la 0 la 4095), din pagina respectiva. Totusi, o impartire cu 3 sau 5 biti pentru numarul paginii este de asemenea posibila. Impartiri diferite implica diferite marimi de pagini.



Operatiile interne ale MMU cu 16 pagini a 4KB

Fig.5

Numarul paginii virtuale este folosit ca index in tabela de pagini pentru a gasi intrarea pentru acea pagina virtuala. De acolo este gasit si numarul frame-ului paginii (page frame), daca exista. Numarul frame-ului paginii este adaugat la capatul high-order al offset-ului, inlocuind numarul paginii virtuale, pentru a forma adresa fizica, ce poate fi trimisa catre memorie.

Ref: "Modern Operating Systems 2nd", Andrews S. Tanenbaum

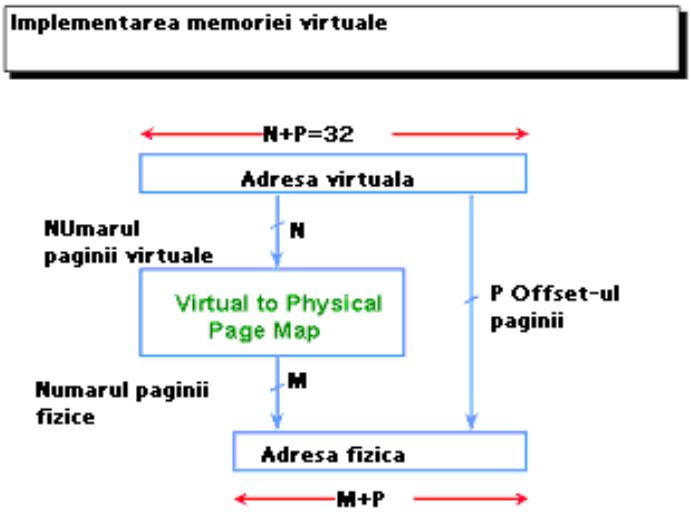


Fig.6

Ref : http://www.microway.com/papers/appnotes/Microway_Working_Note_5.html

Scopul tabelei paginii este de a mapa paginile virtuale spre frame-uri de pagina. Din punct de vedere matematic, tabela de pagina este o functie cu numarul paginii virtuale ca argument si numarul frame-ului fizic ca rezultat. Folosind rezultatul acestei functii, campul paginii virtuale in adresa virtuala poate fi inlocuit cu un camp al frame-ului paginii, astfel formand o adresa fizica de memorie.

In ciuda acestei descrieri, trebuie luate in calcul doua mari probleme:

1. Tabela de pagina poate fi extrem de extinsa.
2. Maparea trebuie sa se realizeze rapid.

Prima problema rezulta din faptul ca PC-urile moderne folosesc adrese virtuale de cel putin 32 de biti. Cu pagina de 4 KB, spatiul adreselor de 32 de biti are 1 milion de pagini, iar spatiul adreselor de 64 de biti ocupa mult mai mult. Cu 1 milion de pagini in spatiul adreselor virtuale, tabela de pagina trebuie sa aiba 1 milion de intrari. Trebuie sa tinem cont ca fiecare proces are nevoie de o tabela de pagina proprie (pentru ca are propriul spatiu de adrese virtuale).

A doua problema este o consecinta a faptului ca translatarea virtuala-fizica trebuie facuta la fiecare referinta la memorie. O instructiune tipica include un cuvant instructiune si un operand de asemenea. In consecinta, este necesar sa se faca referire la tabela de pagina de o data, de doua sau de mai multe ori pe instructiune. Daca o instructiune se face in 4 nsec, o privire in tabela de pagina trebuie sa se faca in mai putin de 1 nsec, pentru a se evita un blocaj major.

Necesitatea pentru o mapare extinsa si rapida este o constrangere importanta pentru modul in care sunt facute computerele. Desi problema este foarte serioasa pentru computerele de top, este, de asemenea, serioasa pentru cele mai slabe, unde costurile si raportul pret-performanta sunt critice. In aceasta sectiune si in urmatoarele, urmarim designul tabelei de pagina in detaliu si aratam cateva solutii hardware, care au fost observate in computerele actuale.

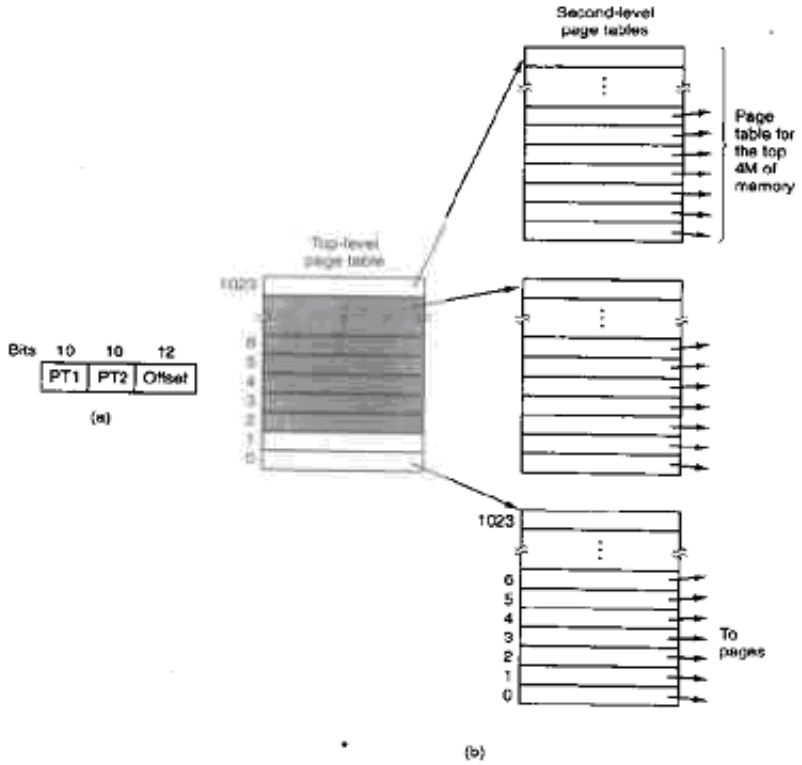
Cel mai simplu design (cel putin la nivel conceptual) ar fi sa existe o singura tabela de pagina constand intr-o matrice de registre rapide hard, cu o intrare pentru fiecare pagina virtuala, indexata dupa numarul acesteia (asa cum se vede in fig 4-11). Cand un proces este pornit, sistemul de operare

incarca registrii cu tabela de pagina a procesului, luata dintr-o copie tinuta in memoria principala. Pe timpul executiei procesului nu este necesara nici o alta accesare a memoriei pentru tabela de pagina. Avantajele acestei metode sunt faptul ca este simpla si nu necesita accesarea memoriei in timpul maparii. Un dezavantaj ar fi ca metoda este costisitoare daca tabela este mare, iar incarcarea intregii tabele in fiecare context afecteaza performanta.

La cealalta extrema, tabela de pagina ar fi putea fi in intregime retinuta in memoria principala. Atunci ar fi nevoie de un singur registru care sa indice inceputul tabelei de pagina. Acest mod permite ca maparea memoriei sa fie schimbata intr-un anumit context prin incarcarea unui registru. De sigur, aceasta metoda are dezavantajul de a necesita una sau mai multe accesari ale memoriei pentru citirea intrarilor tabelei de pagina, in timpul executiei fiecarei instructiuni. Din acest motiv aceasta abordare este folosita rareori in forma ei cea mai pura.

4.2 Tabele de pagini multilevel

Pentru a trece peste problema stocarii in memorie a unor tabele de pagina imense, multe computere folosesc o tabela de pagina pe mai multe nivele (multilevel). Un exemplu simplu este aratat in fig. 4-12. In 4-12 (a) este prezentata o adresa virtuala de 32 de biti impartita in mai multe campuri : 10 biti pentru campul **PT1,** 10 biti pentru PT2 si 12 biti pentru offset. Din moment ce offset-ul are 12 biti vom avea pagini de 4 KB.



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

Fig.7

Secretul metodei tabelei de pagina multilevel este de a evita pastrarea tuturor tabelelor de pagina in memorie tot timpul. Presupunand ca un proces are nevoie de 12 MB, primii 4 MB de jos din memorie – pentru textul programului, urmatorii 4 MB – pentru date, si ultimii 4 MB pentru stiva. Intre partea de sus a datelor si partea de jos a stivei este un mare gol nefolosit.

(*2)Herea Florentina

Ref: "Modern Operating Systems 2nd", Andrews S. Tanenbaum

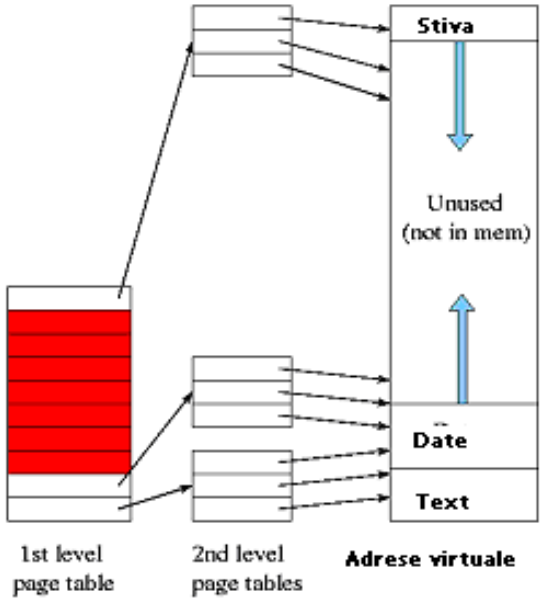


Fig.8

Zona rosie corespunde portiunii nefolosite din spatiul adreselor virtuale. PTE-urile rosii nu au corespondenta catre frame-uri, cauzand un page fault in cazul in care sunt accesate.

Ref: **http://cs.nyu.edu/~gottlieb/courses/1999-00-spring/os/class-notes.html**

In fig. 4-12 (b) vedem cum functioneaza o tabela de pagina pe 2 nivele. In stanga avem tabela de pagina "top-level" cu 1024 de intrari, corespunzatoare campului PT1 de 10 biti. Cand o adresa virtuala este prezentata MMU, se extrage mai intai campul PT1 si se foloseste aceasta valoare ca index in tabela de pagina top-level. Fiecare dintre aceste 1024 de intrari reprezinta 4M, deoarece intregul spatiu al adreselor virtuale de 4 GB a fost impartit in bucati de 1024 Bytes.

Intrarea localizata prin indexarea in tabela de pagina top-level da adresa sau numarului frame-ului de pagina a tabelei de pagina de nivel 2. Intrarea 0 din tabela de pagina top-level indica tabela de pagina pentru textul programului, intrarea 1 indica tabela de pagina pentru date, iar intrarea 1023 indica tabela de pagina pentru stiva. Celelalte intrari nu sunt folosite. Campul PT2 este folosit acum drept index in tabela de pagina de nivel 2 selectata, pentru a gasi numarul frame-ului de pagina pentru pagina in sine.

De exemplu, consideram adresa virtuala de 32 de biti : 0x00403004 (4 206 596 decimal), care inseamna 12 292 bytes in date. Aceasta adresa virtuala are campurile corespunzatoare : PT1=1, PT2=1 si Offset=4. MMU mai intai foloseste PT1 pentru a cauta in tabela top-level, pentru a obtine intrarea 1, corespunzatoare adreselor dela 4M la 8M. Apoi foloseste PT2 pentru a cauta in tabela de nivel 2, tocmai gasita, pentru a extrage intrarea 3, corespunzatoare adreselor 12 288 – 16 386 (care fac parte din bucata de adrese 4M: 4 206 592 – 4 210 687). Aceasta intrare contine numarul frame-ului de pagina a paginii care contine adresa virtuala 0x00403004. Daca acea pagina nu se afla in memorie, *bitul de Present/Absent* din intrarea in tabela de pagina va fi 0, cauzand un “page fault”. Daca pagina se afla in memorie, numarul frame-ului paginii luat din tabela de nivel 2 este combinat cu offset-ul pentru a construi adresa fizica. Aceasta adresa este trimisa pe Bus catre memorie.

Partea interesanta este ca, desi spatiul adreselor contine peste 1 milion de pagini, doar 4 tabele de pagina sunt realmente necesare: tabela top-level si tabelele de nivel 2 de la 0 la 4M, de la 4M la 8M si top 4M. Bitii pentru Prezent/Absent din 1021 de intrari in tabela de pagina top-level sunt setati 0, fortand page fault-ul daca sunt accesati. Daca aceasta se intampla sistemul de operare va observa ca procesul incearca sa acceseze o parte de memorie pe care nu ar trebui, si va lua masurile corespunzatoare, cum ar fi trimiterea unui semnal sau sa termine procesul. In acest moment am ales numere rotunde pentru diferitele marimi si am ales PT1 egal cu PT2, dar in practica sunt posibile si alte valori.

Tabela de pagina de nivel 2 a sistemului din fig. 4-12 poate fi extinsa la 3,4 sau mai multe nivele. Nivelele aditionale dau mai multa flexibilitate, dar este indoielnic ca ar merita o complexitate mai mare de nivel 3.

(*2)Herea Florentina

Ref: “Modern Operating Systems 2nd”, Andrews S. Tanenbaum

4.3 Structura unei intrari in tabela de pagina

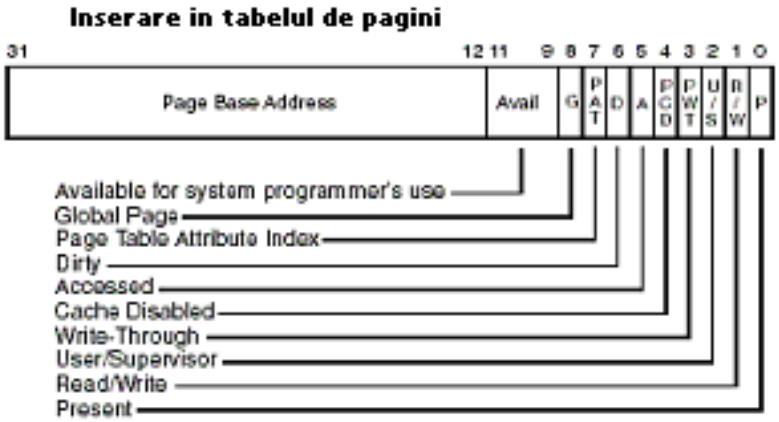
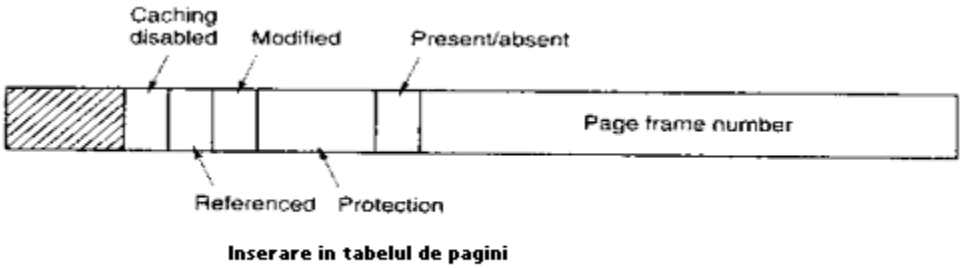


Fig.9

Ref: http://www.viralpatel.net/taj/tutorial/paging.php

Forma exacta a unei intrari in tabela de pagina este puternic dependenta de masina, asa ca difera de la masina la masina. In fig. 4-13 avem un exemplu de o astfel de intrare. In general marimea unei intrari este de 32 biti, dar ea variaza de la calculator la calculator.



Inserare in tabelul de pagini

Fig.10

Dupa cum vedem in figura cel mai mare camp este *numarul de Page Frame*. Acesta este si cel mai important, intrucat scopul maparii paginilor este de a afla acest numar.

Urmeaza bitul de *Prezent/Absent*, care este setat 1 pentru o intrare valida si 0 daca pagina virtuala, corespunzatoare intrarii, nu se afla in memoria curenta. Accesarea unei intrari de tabela de pagina care are acest bit 0 genereaza un **“**page fault**”**.

Bitii de *Protectie* indica tipul de acces permis. Pentru cea mai simpla forma acest camp ocupa 1 bit, care este 0 pentru citire/scriere si 1 doar pentru citire. Campul poate avea si mai mult de 1 bit, o varianta folosita ar fi de 3 biti - cate un bit pentru fiecare : citirea, scrierea si executia paginii.

Bitii de *Modificare* si *Referentiere* tin evidenta folosirii/accesarii paginii.

Bitul de *Modificare* este setat automat hardware, atunci cand se scrie in pagina si este valorificat cand sistemul revendica un page frame. Daca pagina a fost modificata, este considerata “murdara” si trebuie rescrisa pe disk, altfel, daca nu a fost modificata, pagina este considerata “curata” si este lasata in pace, intrucat copia sa de pe disk este valabila. Pentru ca acest bit reflecta starea paginii (“murdara”/“curata”) bitul mai poarta numele de *“dirty bit”*.

Bitul de *Referentiere* este setat ori de cate ori pagina este accesata sau referentiata, fie pentru scriere, fie pentru citire. Bitul are un rol foarte important in momentul in care sistemul de operare trebuie sa aleaga o pagina si se afla in situatia de page fault. Paginile care nu sunt folosite sunt considerate candidate mai bune fata de paginile care sunt folosite, deci bitul de referentiere este important in mai multi algoritmi de inlocuire a paginilor.

Ultimul bit, bitul *Caching* poate scoate optiunea de “caching” pentru pagina. Aceasta optiune conteaza mai mult pentru paginile care mapeaza catre registrele diferitelor device-uri, decat pentru memorie. Daca sistemul de operare se afla intr-o bucla asteptand un dispozitiv de I/O sa raspunda la o comanda tocmai data, este necesar ca hardware-ul sa continue sa ceara raspunsul de la dispozitiv si sa nu foloseasca o copie veche retinuta in cache. Masinile care au spatiu separat pentru I/O si nu folosesc maparea memorie pentru I/O nu au nevoie de acest bit.

Trebuie retinut ca adresa de disk folosita sa retina pagina cand nu este in memorie nu face parte din tabela de pagina. Aceasta din urma retine numai acea informatie necesara 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 sistemul de operare.

(*2)Herea Florentina

Ref: “Modern Operating Systems 2nd”, Andrews S. Tanenbaum

4.4 Windows XP Page File

Performanta este intotdeauna o problema in ceea ce priveste computerele. De aceea suntem foarte atenti la viteza procesorului, cat RAM avem, la viteza placii video si nu numai. Este adevarat ca aceste componente au un rol important in ceea ce priveste performanta, dar mai este un lucru cu un impact substantial in performanta si anume : “Paging File” – Fisierul de Paginare. Fisierul de paginare este strans legat de memoria RAM fizica instalata in calculator, si care impreuna cu memoria fizica se ocupa de memoria virtuala. Are ca scop extinderea RAM-ului fizic pentru datele din RAM ce nu au fost folosite recent. Atat serviciile, cat si aplicatiile instalate, pot beneficia de pe urma acestui “RAM” .

S-a incercat prin diferite metode extinderea RAM-ului instalat, dar la baza toate sunt fisiere de paginare. Atunci cand serviciile si aplicatiile folosesc aproape tot RAM-ul , sistemul de operare “striga” dupa si mai mult RAM si aici intra in functiune Fisierul de Paginare.

Fisierul de paginare este creat in timpul instalarii Windows-ului XP, este memorat pe hard si se masoara in megabytes. Marimea fisierului de paginare se bazeaza pe cantitatea de RAM instalata. In general, XP-ul isi creaza un astfel de Fisier de 1.5 ori mai mare decat RAM-ul instalat, marime minima

recomandata, marime ce poate fi setata si de utilizator pana la un maxim de 3 ori mai mare decat RAM-ul.

Implicit Windows-ul memoreaza fisierul de paginare pe partitia de bootare (partitia ce contine sistemul de operare si fisierele sale de suport). Pentru a imbunatati performanta calculatorului se poate pune fisierul de paginare pe o partitie diferita, sau chiar pe un hard fizic diferit. In acest mod, Windows-ul poate manevra mai multe cereri I/O mai repede. Cand fisierul de paginare se afla pe partitia de bootare, Windows-ul trebuie sa execute cereri de scriere si citire si in folderul de sistem si in fisierul de paginare. Cand fisierul se afla pe o partitie diferita, exista o competitie mai mica intre cererile de scriere si citire.

Principiul de functionare al paginarii la Windows XP sau 2000 este cel clasic in care memoria swap (pagefile) este accesata in momentul cand sistemul nu are suficienta memorie fizica disponibila.

Acest pagefile poate fi de mai 2 tipuri : static sau dinamic.

Cand pagefile-ul este alocat static sitemul de operare rezerva un spatiu anume pe hard-disk, spatiu ce ramane blocat indiferent daca este folosit sau nu. Posesorilor de calculatoare care au putin loc pe hard nu le este la indemana aceasta varianta.

Cand pagefile-ul este alocat dinamic utilizatorul poate alege o dimensiune minima si maxima pe care o poate lua acesta. Dimensiunea minima putand fi chiar 0, acest lucru fiind benefic utilizatorilor cu hard-disk mic sau utilizatorilor cu multa memorie fizica (RAM).

In ultima instanta utilizatorii care beneficiaza de multa memorie ram (mai mult de 1 GB) pot sa apeleze la varianta in care pagefile-ul este oprit si astfel nu se mai ocupa spatiu degeaba pe hard-disk si nici nu mai lasa sistemul sa stocheze date pe disk cand exista suficienta memorie RAM disponibila, ceea ce duce la sporirea performantelor calculatorului.

(*1)Pais Mihail Nicanor

http://en.wikipedia.org/wiki/Virtual_memory

http://en.wikipedia.org/wiki/Paging

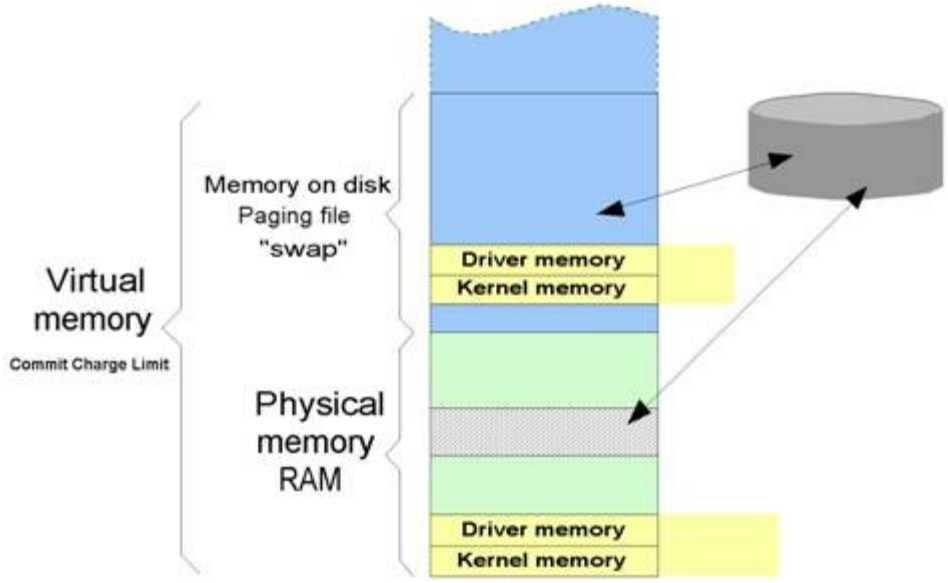


Fig.11

http://www.loriotpro.com/ServiceAndSupport/How_to/How_to_control_Memory_usage.php

4.5 Subsistemul de gestionare a memoriei fizice

Subsistemul de gestiune a memoriei este unul dintre cele mai importante parti ale sistemului de operare. Inca de la aparitia calculatoarelor a existat necesitatea de a avea mai multa memorie decat cea existenta in sistem. Astfel, s-au dezvoltat diverse strategii pentru a combate aceasta limitare si cea mai importanta dintre ele este memoria virtuala. Memoria virtuala face posibila extinderea memoriei sistemului impartind aceasta memorie proceselor necesare.

Subsistemul de gestiune al memoriei garanteaza:

Spatii mari de adresa

Sistemul de operare face ca intreg sistemul sa para ca are mai multa memorie decat este disponibila in realitate. Memoria virtuala poate fi mult mai mare decat memoria fizica aflata in sistem.

Protectie

Fiecare proces din sistem are propriul spatiu de adresa virtuala. Aceste spatii de adrese virtuale sunt complet separate una de alta, astfel ca un proces ce se desfasoara pentru o anumita aplicatie sa nu interfereze alta aplicatie.

Maparea memoriei

Maparea memoriei este folosita pentru a creea o legatura intre imagini sau fisiere de date si spatiul adresei unui proces.

In maparea memoriei, continutul unui fisier este inlantuit direct de spatiul adresei virtuale al unui proces.

Alocarea corecta a memoriei fizice

Subsistemul de gestionare a memoriei permite fiecarui proces activ din sistem o impartire corecta a memoriei fizice a sistemului.

Memoria virtuala comuna

Desi memoria virtuala permite proceselor sa aiba spatii de adrese (virtuale) separate, sunt si cazuri in care mai multe procese pot imparti aceeasi zona de memorie. De exemplu pot exista diferite

procese in sistem care sa utilizeze comanda shell “bash”. Decat sa folosim mai multe copii de bash, unul pentru fiecare spatiu al adresei virtuale al proceselor, este mult mai util sa avem o singura copie in memoria fizica si sa fie comun pentru toate procesele ce utilizeaza bash. Librariile dinamice sunt un alt exemplu tipic de cod de executie utilizat la comun de diferite procese.

Memoria virtuala comuna poate fi folosita, de asemenea, precum un mecanism de comunicare intre procese (Inter Process Communication - IPC), cu doua sau mai multe procese interschimband informatii intre ele prin intermediul memoriei comune de care dispun.

(*4)Nastase Denisa

Referinta :Modern Operating Systems – 2-nd edtion by Andrew Tanenbaum

4.6 Memoria virtuala si memoria fizica

Cantitatea de memorie virtuala si fizica de care dispune fiecare calculator ce foloseste un sistem de operate Microsoft Windows este determinata de configuratia hardware si de editia platformei de Windows folosita. Pe un hardware de 32 de biti spatiul adresei virtuale este de 4 GB si catitatea maxima de memorie fizica este cuprinsa intre 2 si 128 GB. Pe un hardware de 64 de biti spatiul adresei virtuale este de 16 TB si catintatea maxima de memorie fizica este cuprinsa intre 64 GB si 1 TB.

Urmatorul tabel prezinta cantitatea de memorie virtuala si cantitatea maxima de memorie fizica pentru fiecare editie de Windows.

Operating system version	Edition	Virtual memory	Maximum physical memory
Microsoft Windows Server™ 2003 SP 1	Standard	4 GB	4 GB
	Web	4 GB	2 GB
	Enterprise	4 GB	64 GB, if hardware supports Physical Address Extension (PAE)
	Enterprise (64-bit)	16 terabytes	1 terabyte
	Datacenter	4 GB	128 GB, if hardware supports PAE
	Datacenter (64-bit)	16 terabytes	1 terabyte

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

Mapara memoriei virtuale in memorie fizica

Fig.12

Referinta : http://www.microsoft.com/whdc/driver/kernel/mem-mgmt.mspx - Memory Management : What every driver writer needs to know

4.7 Spatiul adresei fizice

Spatiul adrsei fizice contine set de adrese pe care memorie fizica il poate ocupa. Spatiul adresei fizice are o dimensiune maxima determinata de numarul de biti al adresei fizice pe care CPU si chipsetul il pot decoda. Aceasta dimensiune stabileste de asemenea cantitatea maxima teoretica de memorie fizica (RAM) din sistem.

Spatiul adresei fizice nu este numai folosit pentru a accesa memoria fizica(RAM). Acesta mai este de asemenea folosit pentru a accesa intreaga memorie disponibila si o parte din registrii utilizati de alte dispozitive. In consecinta, daca un sistem de calcul este configurat cu cantitatea maxima de memorie fizica , o parte din aceea memorie nu va putea fi folosita deoarece o parte din spatiul adresei fizice este mapat pentru alte intrebuintari.

Memoria utilizata de diferite dispozitive precum si de catre registrii este mapata in

spatiul adresei virtuale intr-o gama de memorie controlata de catre chipset. Locatiile de memorie sunt citite si scrise cu instructiunile : LOAD si STORE.Unii registrii ai dispozitivelor sunt mapati in spatiul adresei fizice, altii in spatiul Input/Output , depinzand de configuratia dispozitivului hardware si de chipsetul acestuia.

Spatiul adreselor fizice (adresa fizica) este descris de :

- Adrese fizice relative procesorului
- Device-bus-uri.

(*4)Nastase Denisa

Referinta : http://www.microsoft.com/whdc/driver/kernel/mem-mgmt.mspx - Memory Management : What every driver writer needs to know

Urmatoare figura prezinta tipurile de adrese folosite de diferitele componente din sistem.

Adresele fizice

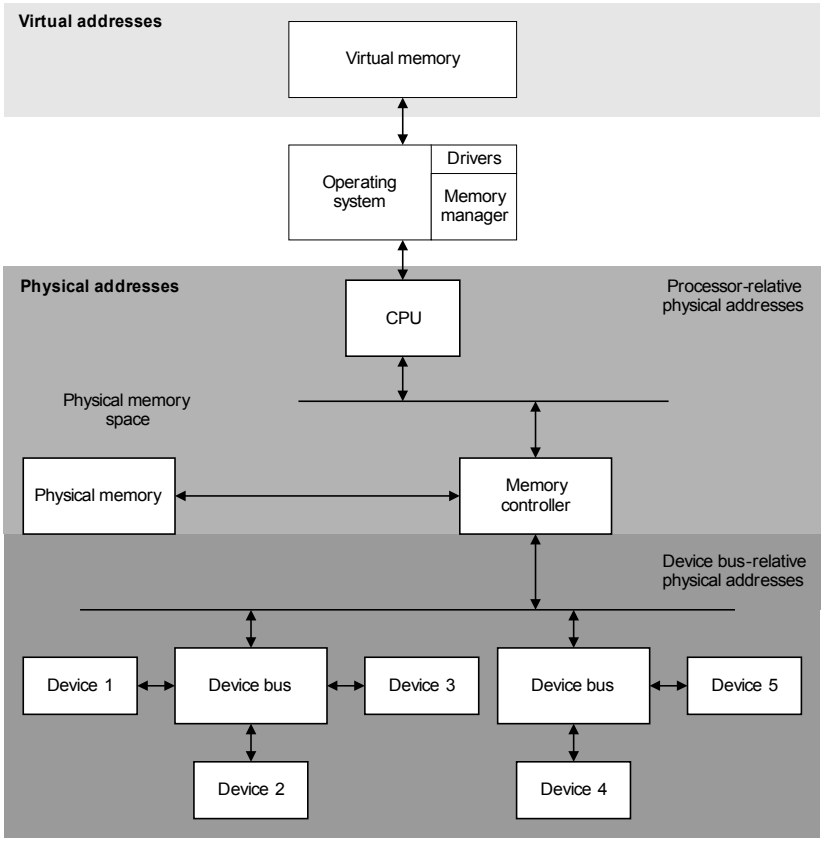


Fig.13

Referinta : http://www.microsoft.com/whdc/driver/kernel/mem-mgmt.mspx - Memory Management : What every driver writer needs to know

(*4)Nastase Denisa

5. Implementarea gestiunii memoriei la Linux

5.1 Linux Page Table Management

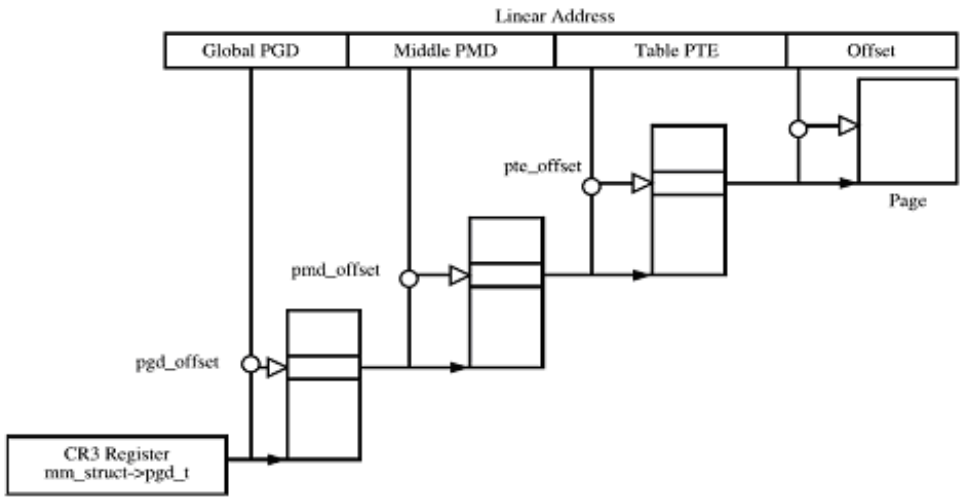
In Linux se prefera mentinerea conceptului de pagina de tabela pe 3 nivele, ceea ce inseamna ca diferentierea intre diferite tipuri de pagini este greu de facut, iar paginile sunt identificate prin flag-urile lor sau prin ceea ce listeaza mai mult decat dupa obiectele caror apartin.

Pentru arhitecturile care isi gestioneaza MMU-ul diferit se presupune ca emuleaza tabelele de pagina pe 3 nivele. Insa, pentru arhitecturile care nu isi gestioneaza automat cache-ul sau Translation Lookaside Buffer-ul (TLB), **hooks** care sunt dependente de masina trebuie sa fie scrise explicit in cod. Pentru aceasta functiile respective si cum ar trebui folosite, sunt foarte bine explicate in fisierul “cachetlb.txt” din documentatia kernel-ului.

5.2 Linux Page Directory

Fiecare proces are propriul *Page Global Directory (PGD)*, care este un page frame fizic continand matricea **pgd_t**, un tip specific de arhitectura definit in <asm/page.h>. Tabelele de pagina sunt incarcate diferit in functie de arhitectura. La x86 tabela de pagina a procesului este incarcata prin copierea pointerului in PGD, in registrul CR3, care are efectul secundar de a curata TLB-ul.

Fiecare intrare activa in tabela PGD indica un page frame continand o matrice de intrari a *Page Middle Directory (PMD)* de tipul **pmd_t**, ce indica in schimb page frame-uri cu *Page Table Entries (PTE)* de tipul **pte_t**, care la randul lor indica in final page frame-urile care contin data ceruta de utilizator.



Aspectul tabelului de pagini

Fig.14

In cazul in care pagina a fost swapped out to backing storage, intrarea incuita este retinuta in PTE si folosita de functia **do_swap_page()** in timpul page fault-ului, pentru a gasi intrarea inlocuita continand pagina.

Orice adresa liniara data poate fi imparita in mai multe parti, offset-uri in cadrul celor 3 nivele ale tabelei de pagina si un offset in cadrul pagine actuale. Pentru a sparge mai usor adresa liniara, sunt furnizate un numar de Macros in trei exemplare pentru fiecare nivel al tabelei de pagina, numite **SHIFT**, **SIZE** si **MASK.**

- **Shift** macros specifica lungimea in biti, care sunt mapati de fiecare nivel al tabelei de pagina, ca in figura urmatoare:
-

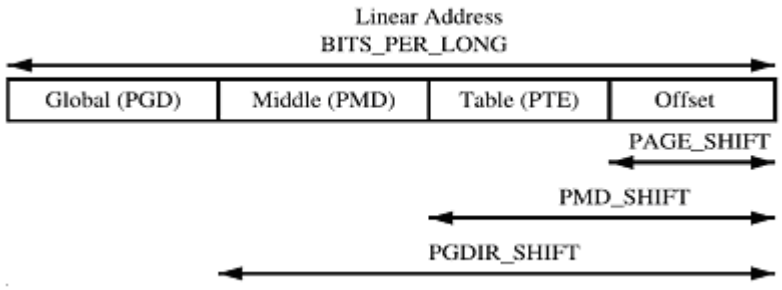


Fig.15

- Valorile **Mask** facute SI logic cu adresa liniara pot masca toti bitii superiori, si sunt folosite frecvent pentru a determina daca o adresa liniara este aliniata cu un anume nivel din tabela de pagina.
- **Size** macros arata cati bytes sunt adresati de fiecare intrare la fiecare nivel.

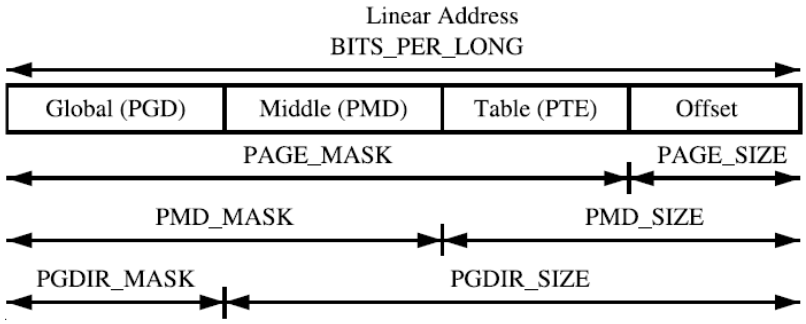


Fig.16

PAGE_SHIFT reprezinta lungimea in biti a partii de offset din spatiul adresei liniare , care este de 12 biti la x86. Marimea paginii se poate calcula foarte usor ca fiind 2^{PAGE_SHIFT}. Masca se calculeaza negand bitii din PAGE_SIZE – 1. Daca o pagina trebuie aliniata la limita se foloseste PAGE_ALIGN() , acest macro adaugand PAGE_SIZE – 1 este adaugat la adresa inainte de a se face operatia SI logic cu PAGE_MASK.

PMD_SHIFT este numarul de biti din adresa liniara , mapati de al doilea nivel al tabelei. PMD_SIZE si PMD_MASK sunt calculate similar cu macro-urile pentru tabela de pagina.

PGDIR_SHIFT reprezinta numarul de biti mapati de primul nivel al tabelei de pagina.

Ultimele 3 macrouri importante reprezinta PTRS_PER_x, care determina numarul de intrari din fiecare nivel al tabelei de pagina. PTRS_PER_PGD este numarul de pointeri din PGD, 1024 la x86 fara PAE. PTRS_PER_PMD este pentru PMD, 1 la x86 fara PAE si PTRS_PER_PTE este pentru cel mai jos nivel, 1024 la x86.

(*2)Herea Florentina

Ref: "Understanding The Linux Virtual Memory Manager", Mel Gorman

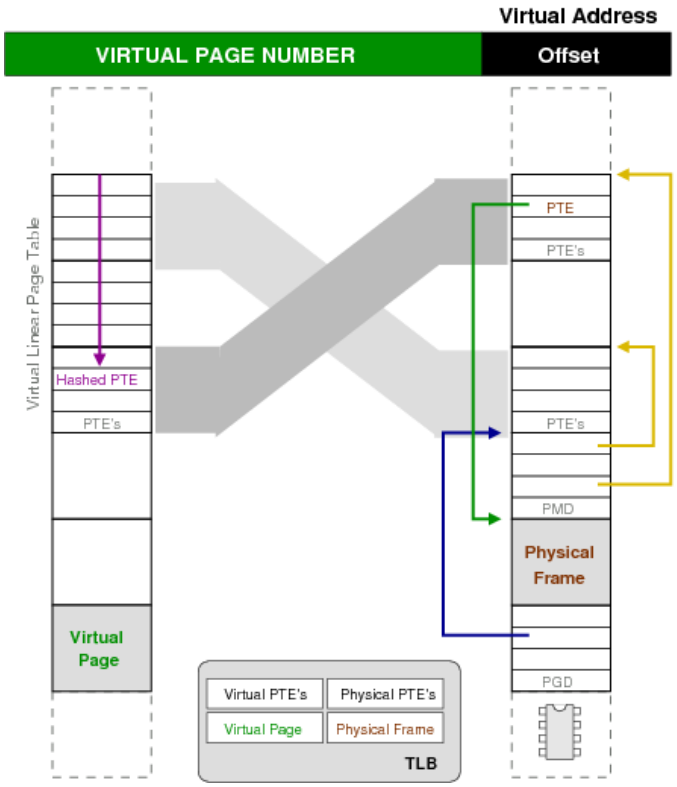


Fig.17

Ref: **www.technovelty.org/code/linux/**

5.3 Memoria Virtuala la linux:

Un principiu de baza al implementarii memoriei virtuale in Linux este conceptul de "pagina". O pagina este o zona de memorie de 4 Kb si este unitatea de baza a memoriei cu care kernel-ul si CPU-ul opereaza. Chiar daca amandoua acceseaza biti sau bytes individuali, cantitatea de memorie este impartita totusi in pagini.

Linux-ul foloseste principiul de a incarca in memoria fizica doar ce este necesar si cu care se lucreaza in momentul acela, pentru a lasa loc in memorie pentru orice alt program, care are de executat ceva anume. Ca de exemplu atunci cand erati studenti si invatati la o masa cu toate cartile pe masa deschise la anumite capitole dar nu foloseati decat una odata si nimeni altcineva nu putea sa invete la acea masa din cauza cartilor nefolosite in acel moment. Daca le tineati pe cele la care nu va uitati in acel moment in sertar sau intr-un colt al mesei inchise, alt coleg putea invata si el la aceeasi masa si astfel se putea folosi mai eficient masa de lucru.

Responsabilitatea pentru gestiunea memoriei in Linux este impartita intre Kernel si CPU. Acestea au grija sa faca schimbul intre memoria de pe disc si memoria fizica, astfel ca niciodata sa nu se incarce ceva gresit cand programul cere ceva ce se afla pe disc si memoria fizica e plina, adica un proces sa nu aiba acces la datele altui proces.

CPU-ul ajuta Kernel-ul prin faptul ca ii comunica dorinta unui program de a accesa o data ce nu se afla in memorie. Kernel-ul atunci identifica procesul ce a cerut acea pagina si o incarca. Tot responsabilitatea Kernel-ului este de a nu lasa un proces sa ocupe toata memoria fizica, daca se intampla sa fie singurul proces ce ruleaza, in ideea de a facilita accesul la memorie a noilor procese ce vor aparea. Doar daca memoria fizica este mult mai mare decat programul, atunci el va fi incarcat complet in memorie.

Kernel-ul este mai mereu ocupat cu incarcarea paginilor corecte fiecarui proces. De cele mai multe ori sunt mai multe procese ce ocupa mai mult decat memoria fizica si astfel se foloseste o portiune de disc pentru stocari de date, ce nu sunt folosite, numita “swap”. In functie de necesitati swap-ul poate fi mai mare sau mai mic.

Paginile ce vor fi incarcate la loc in memorie din swap sunt doar cele ce sunt alocate unor procese care sunt in starea “activa de lucru”. Acest lucru era benefic deoarece incarcarile ce nu erau necesare pe moment, nu duceau decat la o ingreunare a calculatorului, deoarece memoria “hard-disk” este mult mai lenta decat memoria RAM.

Pana la kernel-ul 2.3.24 memoria swap maxim folosibila era de 127,5 MB. Se putea mari dimensiunea fisierului, dar doar 127,5 MB din el urmau sa fie folositi. Pentru a avea mai mult swap se creau, totusi, mai multe fisiere de 127,5 MB, dar insumate nu trebuiau sa depaseasca 2 GB.

In momentul de fata Linux-ul poate suporta pana la 64 GB de memorie fizica si pana la cativa TB de memorie swap.

(*1)Pais Mihail Nicanor

http://en.wikipedia.org/wiki/Paging

Ref: “Modern Operating Systems 2nd”, Andrews S. Tanenbaum

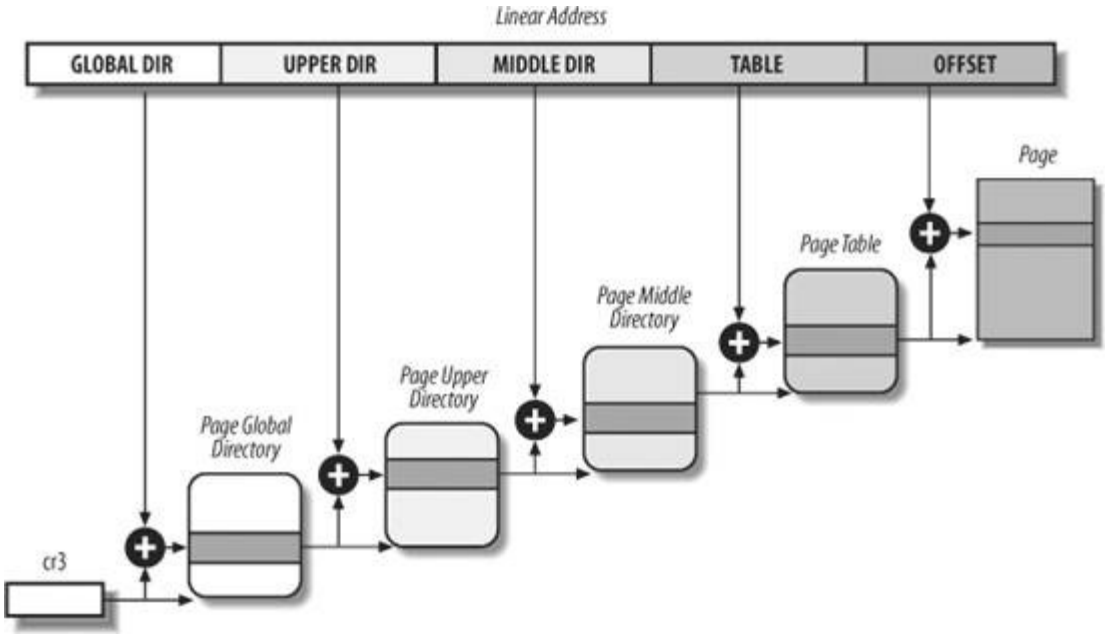


Fig.18

http://www.linux-security.cn/ebooks/ulk3-html/0596005652/understandlk-CHP-2-SECT-5.html

5.4 Algoritmul prietenului la Linux

Sistemul prietenului pentru Linux este un algoritm simplu de alocare a memoriei. Principalul sau scop este de a reduce pe cat posibil numarul intreruperilor externe si in acelasi timp sa permita alocarea si re-alocarea rapida a paginilor in memorie. Pentru a reduce numarul intreruperilor externe paginile adiacente din memorie sunt grupate in liste de diferite dimensiuni. Acest lucru permite ca toate blocurile formate din doua pagini sa se afle intr-o lista, blocurile formate din patru pagini in alta lista s.a.m.d. Daca va exista o cerere de 4 pagini adiacente, cererea poate fi usor satisfacuta prin verificarea blocurilor de 4 pagini libere. Deci daca un bloc de 8 pagini va fi liber acesta se va imparti in 2 blocuri de cate 4 pagini si una dintre ele va fi trimisa catre cerere, iar cealalta va fi alocata listei cu blocuri de 4 pagini. Acest lucru evita impartirea blocurilor libere cu un numar mare de pagini cand o cerere poate fi satisfacuta de un bloc mult mai mic, astfel reducandu-se fragmentarea externa.

De asemenea adresa fizica a primei pagini trebuie sa fie un multiplu al dimensiunii blocului. Exemplu: un bloc de dimesiune 2^n trebuie sa fie aliniat cu $4k*2^n$.

In schimb, cand o pagina de un anumit rang este eliberata din memorie, se realizeaza o incercare de a o ingloba in blocul ei corespunzator (blocul "prieten") de acealasi rang, daca acesta este liber, permitand astfel blocurilor de un rang mai mare sa ramana libere. Aceast lucru are loc recursiv pana cand nu mai este posibila o asociere mai mare. Blocul ramas liber este apoi alocat listei libere careia ii corespunde. Acest procedeu se mai numeste si *fuziune.*

Linux foloseste liste de 1,2,4,8,16,32,64,128,256 si 512 blocuri de pagini. Pentru a gestiona aceste liste si pentru a implementa sistemul buddy se foloseste structura free_area_struct (numita si free_area_t)

```
typedef struct free_area_struct{
        struct list_head free_list;
        unsigned long *map;
```

}free_area_t

Campurile structurii de mai sus sunt folosite astfel :

free_list - este o lista dublu inlantuita de blocuri libere de o anumita dimensiune. Aceasta are la capete primul si respectiv ultimul bloc de pagini, in timp ce structura *list* este folosita pentru a inlantui paginile intre ele.

map - numit si prietenul “bitmap”, contine informatii privind disponibilitatea unui *prieten.* Dimensiunea lui este data de formula: ((numarul de pagini) - 1 >> (rang + 4)) + 1 byte.

Fiecare bit reprezinta doua blocuri adiacente de aceeasi dimensiune. Are ca valoare 0 daca ambele blocuri sunt partial sau total ocupate sau daca ambele sunt libere. Are ca valoare 1 daca unul din cele doua blocuri este liber si unul este (partial sau total) ocupat.

Fiecare zona are un vector format din aceste structuri, unul pentru fiecare dimensiune.

Exemplu :

Sa presupunem ca avem un sistem cu doar 16 pagini de RAM (figura 2.1). Din moment ce sunt doar 16 pagini de RAM vom avea *prieteni bitmap* doar pentru patru ranguri. Acestea vor fi distribuite astfel :

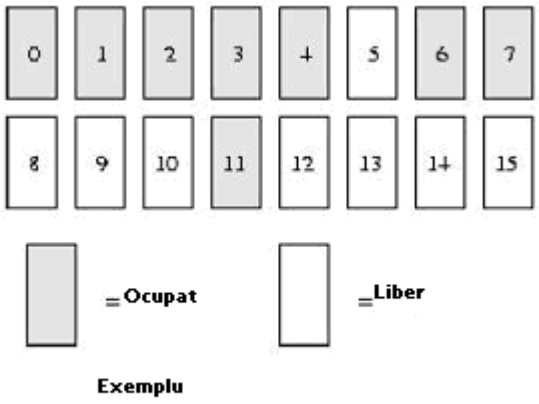


Fig.19

Referinta : http://freesoftware.fsf.org/lkdp- Memory management in Linux

pagini :	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
rang (0) :	0		0		1		0		0		1		0		0	
rang (1) :		0			0				1				0			
rang (2) :			0							1						
rang (3) :								0								

In rang (0), primul bit reprezinta primele doua pagini, al doilea bit reprezinta urmatoarele doua pagini s.a.m.d. Al treilea bit este 1 deoarece pagina 4 este ocupata in timp ce pagina 5 este libera. De asemenea in rang (1) bitul 3 este 1 deoarece un *prieten* este complet liber (paginile 8 si 9) si celalalt *prieten* este ocupat (paginile 10 si 11) asa ca exista posibilitatea unei asocieri.

Alocarea

Mai jos sunt pasii ce trebuie executati daca vrem ca un bloc de pagini de rang (1) sa fie liber :

1. Initial lista libera va fi :
Rang (0) : 5, 10
Rang (1) : 8 [8,9]
Rang (2) : 12 [12,13,14,15]
Rang (3) :

2. Din moment ce lista de rang (1) contine un bloc de pagini liber acesta este returnat catre utilizator si inlaturat din lista.

3. Daca vom avea nevoie de alt bloc de rang (1) vom scana din noua listele libere incepand cu lista de rang (1).

4. Din moment ce nu exista nici un bloc de rang (1) disponibil, se trece la lista urmatoare de rang (2).

5. Aici exista un bloc de pagini liber incepand cu pagina 12. Acest bloc va fi impartit in doua blocuri mai mici de rang (1) : [12,23] si [14,15]. Blocul [14,15] va fi adaugat la lista de rang (1) iar celalalt bloc [12,13] va fi trimis catre utilizator.

6. In final lista libera va fi:

Rang (0) : 5, 10
Rang (1) : 14 [14,15]
Rang (2) :
Rang (3) :

De-alocarea

Luand in calcul acelasi exemplu de mai sus, in continuare vom prezenta pasii parcursi pentru eliberarea pagina 11 (de rang (0)).

1. Se gaseste bitul care reprezinta pagina 11 in *prietenul bitmap (lista de prieteni)* de rang (0) folosind urmatoarea formula :

index = page_idx >> (order + 1) = 11 >> (0 + 1) = 5

2. Apoi se verifica valoarea acelui bit. Daca este 1 inseamna ca exista un prieten liber adiacent. Bitul 5 este 1 pentru ca prietenul aferent (pagina 10) este liber.

3. Se reseteaza acest bit la valoarea 0, ambii prieteni fiind acum liberi.

4. Se extrage pagina 10 din lista de rang (0).

5. Se reia procedeul avand de data aceasta 2 pagini libere (10 si 11, de rang (1)).

6. Acest nou bloc de pagini va incepe cu pagina 10. Indexul sau se va gasi in lista de prieteni de rang (1). Folosind aceeasi formula scrisa mai sus vom deduce ca acesta este dat de al treilea bit de valoare 2.

6. Concluzii :

Memoria Virtuala a fost un pas inainte pentru sistemele de operare la momentul aparitiei deoarece memoriile RAM erau de abea la inceput si astfel se puteau rula mai multe programe odata sau un program ce necesita mai multa memorie fara a cheltui o avere pe o memorie mai mare !

Odata cu trecerea timpului memoriile au devenit din ce in ce mai mari si din ce in ce mai ieftine si astfel in momentul de fata “swap file”-ul poate fi considerat optional la sistemele cu mai mult de 1 GB de RAM. Din aceasta cauza sistemele moderne de operare Windows au pe langa optiunea de marire sau micsorare a swap file-ul si optiunea de dezactivare a acesteia. Astfel se lucreaza doar cu memoria RAM care este mult mai rapida decat Hard Disk-ul.

Din punct de vedere al paginarii linuxul foloseste un principiu care in momentul de fata nu mai este unul ideal si anume sa tina mereu ceva din program pe Hard-Disk astfel incat sa ramana memorie libera in permanenta pentru un program ce o sa ruleze la un moment viitor.

(*1) Pais Mihail Nicanor

Referitor la modul de paginare, se foloseste tabela de pagina este principala unealta.
Folosirea tabelei de pagina este un mod usor de a translata o adresa virtuala intr-o adresa fizica. Desi ce mai simpla solutie ar fi o singura tabela de pagina cu o intrare pentru fiecare pagina virtuala. Chiar daca pagina ar consta intr-o matrice de registre rapide hard, incarcarea acesteia tot ar dura foarte mult si performantele ar scadea, deci solutia nu mai este fiabila. De aceea pentru amble sisteme se operare s-a ajuns la o tabela de pagina care sa contina pointeri catre alte tabele. La Windows se numeste tabela de pagina multilevel(tabela ce indica o alta tabela … si tot asa), iar la Linux se foloseste PGD (Page Global Directory), care indica PMD (Page Middle Directory), care la randul lui indica PTE(Page Table Entries). De obicei nu este necesara o ierarhizare mai mare de trei nivele, asa ca metoda este eficienta si rapida totodata.

(*2) Herea Florentina

Una din cele mai importante si dificile provocari este aceea de a gestiona memoria in mod eficient.

O analiza a memoriei virtuale demonstreaza faptul ca acest concept este nu numai fezabil, dar si extrem de folositor , precum si o necesitate de baza continua crestere a sistemelor computationale, unde memoria principala de mare viteza este limitata.

Zilnic programatorii se confrunta cu problema gasirii unei metode eficiente, printre care se numara folosirea memoriei virtuale,memoriei heap, maparea memoriei , pentru implementarea diferitelor tipuri de date . O alta problema consta in a stii cand sa pastram datele continute in memorie si cand sa renuntam la ele in vederea refolosirii spatiului de memorie existent. Din pacate exista multe moduri in care o gestiune ineficienta a memoriei poate afecta robustetea si viteza de calcul a sistemului.

(*4) Nastase Denisa

Bibliografie generala:

1) Tanenbaum - Modern Operating Systems 2nd Ed
2) http://www.linux-tutorial.info/index.php
3) http://en.wikipedia.org/wiki/Page_replacement_algorithm#The_theoretically_optimal_page_replacement_algorithm
4) Abhishek Nayani, Mel Gorman & Rodrigo S. de Castro – Memory Management in Linux
5) “Understanding The Linux Virtual Memory Manager”, Mel Gorman

Studenti :

1) Pais Mihail Nicanor (441A) (Memoria Virtuala, Paginarea, Memoria Virtuala si paginarea la Linux si Windows, Bitmap, Stack/List of pages, Scheme hibride)
2) Herea Florentina (441A) (Tabela de pagini, Tabela de pagini Multinivel, Structura unei intrari in tabela de pagini,Linux Page-Table Management, Flat List, Arborele AVL)
3) Pandia Cristian (441A) (Swaping si algoritmi de inlocuire a paginilor)
4) Nastase Denisa (441A) (Subsistemul de gestiune a memoriei fizice)
5) Apostol Catalin (441A) (Algoritmul “prieten”)