Gestiunea memoriei

1) Concepte fundamentale: segmentul de cod, segmentul de date, spatial virtual de adrese la Windows

1.1. Introducere

În conformitate cu arhitectura von Neumann, **memoria primară (internă)** este o componentă principală a calculatoarelor, rolul ei fiind de a păstra date şi programe, atunci când acestea urmează a fi folosite de către un proces executat de către CPU. Memoria primară împreună cu registrele CPU şi memoria „cache" formează **memorie executabilă**, deoarece aceste componente sunt implicate în execuţia unui proces. CPU poate încărca instrucţiunile acestuia numai din memoria primară. Pentru a fi prelucrate, datele sunt încărcate de către unitatea aritmetică şi logică din memoria internă în regiştrii CPU, iar după efectuarea unei instrucţiuni cod maşină, sunt stocate în memoria internă. Unităţile de memorie externă (secundară) sunt utilizate pentru a stoca date pentru o mai lungă perioadă de timp. Fişierele executabile, pentru a deveni procese, precum şi informaţiile prelucrate de acestea, trebuie să fie încărcate în memoria primară.

Există trei cerinţe de bază ale memoriei interne [1]:
• Timpul de acces la memoria internă trebuie să fie cât mai mic posibil; acest lucru se realizează prin proiectarea adecvată atât a componentei hardware cât şi a celei software implicate în gestiunea memoriei. Calculatoarele moderne folosesc m**emoria "cache",** ce reprezintă un tip de memorie cu acces foarte rapid şi care conţine informaţiile cele mai recent utilizate de către CPU.
• Dimensiunea memoriei adresabile trebuie să fie cât mai mare posibil, ceea ce se poate realiza prin conceptul de **memorie virtuală**.
• Memoria internă trebuie să aibă un cost relativ scăzut.

Principalele **obiective** ale gestiunii memoriei sunt:
• calculul de translatare a adresei(relocare);
• protecţia memoriei;
• organizarea şi alocarea memoriei operative;
• gestiunea memoriei secundare;
• politici de schimb între procese, memoria operativă şi memoria secundară.

1.2 Segmentul de cod si de date

1.2.1 Memoria virtuală

Este o tehnică ce permite execuţia proceselor fără a impune necesitatea ca întreg fişierul executabil să fie încărcat în memorie sau capacitatea de a adresa un spaţiu de memorie mai mare decât este cel din memoria internă [1]. Utilizarea memoriei virtuale oferă avantajul că poate fi executat un program de dimensiune oricât de mare (mai mare

decât dimensiunea memoriei fizice); spaţiul de adrese al procesului este divizat în părţi care pot fi încărcate în memoria internă atunci când execuţia procesului necesită acest lucru şi transferate înapoi în memoria secundară, când nu mai este nevoie de ele. Spaţiul de adrese al unui program se împarte în **partea de cod**, cea **de date** şi cea **de stivă**, identificate atât de compilator cât şi de mecanismul hardware utilizat pentru relocare.

Partea (segmentul) de cod are evident un număr de componente mai mare, fiind determinată de fazele de execuţie (logica) ale programului. De exemplu, aproape toate programele conţin o fază necesară iniţializării structurilor de date utilizate în program, alta pentru citirea datelor de intrare, una (sau mai multe) pentru efectuarea unor calcule, altele pentru descoperirea erorilor şi una pentru ieşiri. Analog, există partiţii ale segmentului de date. Această caracteristică a programului se numeşte localizare a **referinţelor în spaţiu [2]** şi este foarte importantă în strategiile utilizate de către sistemele de memorie virtuală.

Când o anumită parte a programului este executată, este utilizată o anumită porţiune din spaţiul său de adrese, adică este realizată o localizare a referinţelor. Când se trece la o altă fază a calculului, corespunzătoare logicii programului, este referenţiată o altă parte a spaţiului de adrese şi, deci se schimbă această localizare a referinţelor. De exemplu, în figura 1, spaţiul de adrese este divizat în 5 părţi.

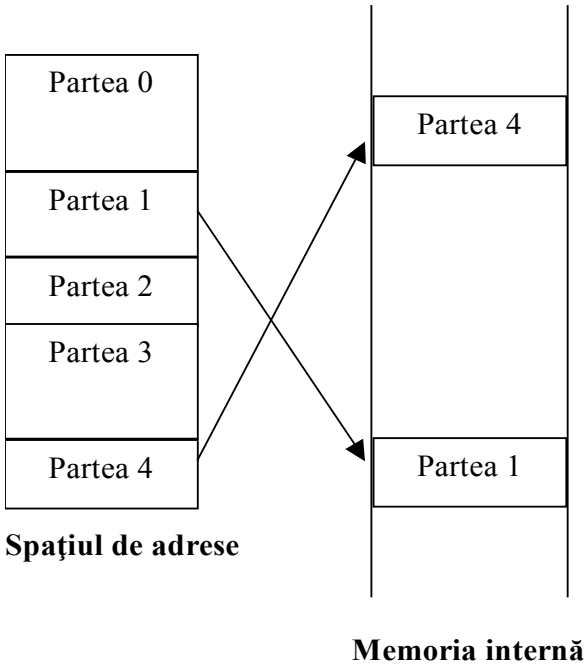

Figura 1 Memoria fizică şi cea virtuală

Numai părţile 1 şi 4 din spaţiul de adrese corespund unor faze ale programului care se execută la momentul respectiv, deci numai acestea vor fi încărcate în memoria internă.

Părţi diferite ale programului vor fi încărcate în memoria primară la momente diferite, în funcţie de oportunitatea de a fi solicitate in desfasurarea procesului. Sarcina administratorului memoriei este de a deduce localizarea programului şi de a urmări încărcarea în memorie a partiţiilor din spaţiul de adrese corespunzătoare, precum şi de a ţine evidenţa acestora în memoria internă, atâta timp cât sunt utilizate de către proces.

Administratorul memoriei virtuale alocă porţiuni din memoria internă care au aceeaşi dimensiune cu cele ale partiţiilor din spaţiul de adrese şi, deci încarcă imaginea executabilă a părţii corespondente din spaţiul de adrese într-o zonă din memoria primară. Acest lucru are ca efect utilizarea de către proces a unei cantităţi de memorie mult mai reduse.

[1] Inside Windows 2000 by David A. Solomon & Mark E. Russinovich

[2] Undocumented Windows 2000 secrets by Sven B. Schreiber

http://www.intellectualheaven.com/Articles/WinMM.pdf

1.2.2. Translatarea adreselor.

Virtualizarea este o tehnica de ascundere a caracteristicilor resurselor unui calculator prin felul in care alte sisteme de operare, aplicatii si utilizatorii interactioneaza cu aceste resurse. Virtualizarea include crearea unei singure resurse fizice (server, sistem de operare, hard disk), pentru a functiona ca multiple resurse logice sau invers (mai multe masini fizice formand una singura, o masina virtuala). Virtualizarea permite multiple masini virtuale, cu sisteme de operare heterogene, sa ruleze separat, una langa alta, pe aceeasi masina fizica. Fiecare masina virtuala are propriul ei set de hardwere (RAM, CPU, NIC, etc) pe care este incarcat un sistem de operare cu tot cu aplicatii. Masinile virtuale sunt incapsulate in fisiere, facand posibila copierea si salvarea rapida a masinii. Intregul sistem poate fi mutat in cateva secunde (sistemul de operare, aplicatiile, Bios-ul virtual, hardwerul virtual). [3]

Funcţia de translatare a adreselor virtuale, Ψ_t este o corespondenţă, variabilă în timp a spaţiului de adrese virtuale ale unui proces, în **spaţiul de adrese fizice**, adică mulţimea tuturor locaţiilor din memoria internă alocate acelui proces [4].

Se cunosc două metode de virtualizare: **alocare paginată şi alocare segmentată.** Deci:

$$\Psi_t : < Spatiul\_de\_adrese\_relative > \rightarrow < Spatiul\_de\_adrese\_fizice > \cup \{\Omega\}$$

în care t este un număr întreg, care reprezintă timpul virtual al procesului iar Ω este un simbol care corespunde adresei nule. Când un element i, al spaţiului de adrese virtuale este încărcat în memoria internă, $\Psi_t(i)$ este adresa fzică unde adresa virtuală i este încărcată. Dacă $\Psi_t(i) = \Omega$, la momentul virtual t şi procesul face referinţă la locaţia i, atunci sistemul întreprinde următoarele acţiuni:

1. Administratorul memoriei cere oprirea execuţiei procesului.

2. Informaţia referenţiată este regăsită în memoria secundară şi încărcată într-o locaţie de memorie k.
3. Administratorul memoriei schimbă valoarea funcţiei Ψ , $\Psi_t(i) = k$.
4. Administratorul memoriei cere reluarea execuţiei programului.

Observăm că locaţia referenţiată din spaţiul de adrese virtuale nu este încărcată în memoria primară, după ce s-a declanşat execuţia instrucţiunii cod maşină respective. Acest lucru va declanşa **reexecutarea** instrucţiunii după ce locaţia respectivă a fost încărcată din memoria virtuală în memoria primară. De asemenea, dimensiunea spaţiului de adrese virtuale ale unui proces, este **mai mare** decât dimensiunea spaţiului de adrese fizice alocat procesului, spre deosebire de metodele când întregul fişier executabil era încărcat în memorie.

[3] http://www.asseut.org/biblioteca/tehnologii-de-virtualizare-virtualizarea-referat.php

[4] http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.mspx

1.3. Alocarea paginată

A apărut la diverse SC ? pentru a evita fragmentarea memoriei interne, care apare la metodele anterioare de alocare. Memoria virtuală este împărţită în zone de lungime fixă numite **pagini virtuale [5].** Paginile virtuale se păstrează în memoria secundară. Memoria operativă este împărţită în zone de lungime fixă, numite **pagini fizice.** Lungimea unei pagini fizice este fixată prin hard. Paginile virtuale şi cele reale(fizice) au aceeaşi lungime, lungime care este o putere a lui 2, şi care este o constantă a sistemului (de exemplu 1Ko, 2Ko, etc). Să presupunem că G reprezintă numărul de locaţii de memorie virtuală ale unui fişier executabil, ale căror adrese sunt cuprinse între 0 şi G-1. De asemenea, să notăm cu H numărul locaţiilor din memoria internă, necesare pentru a încărca conţinutul fişierului executabil, în timpul execuţiei procesului (H<G). De asemenea, cele G-1 locaţii virtuale, corespund la n, cu $n = 2^g$ pagini virtuale, fiecare pagină fiind de dimensiune c, cu $c = 2^h$. Spaţiul de adrese din memoria primară poate fi gândit ca o mulţime de m pagini fizice, $m = 2^j$, fiecare având dimensiunea $c = 2^h$, deci cantitatea de memorie internă alocată procesului va fi $H = 2^{h+j}$. La un anumit moment al execuţiei, conform principiului localizării, numai o parte din paginile virtuale vor fi încărcate în memoria internă, în pagini fizice. Problema care se pune este translatarea fiecărei adrese virtuale într-o adresă fizică.

1.4. Translatarea adreselor virtuale.

Fie $N = \{d_0, d_1, .., d_{n-1}\}$ mulţimea paginilor în spaţiul de adrese virtuale şi $M = \{b_0, b_1, .., b_{m-1}\}$ mulţimea paginilor fizice din memoria primară alocate procesului. O adresă virtuală este un întreg i, unde $0 \le i < G = 2^{g+h}$, deoarece există $n = 2^g$ pagini, fiecare având 2^h cuvinte(locaţii) de memorie [4]. O adresă fizică, k, este o adresă de memorie, de forma $k = U2^h + V (0 \le V < 2^h)$, unde U este numărul pagini fizice. Deci $U2^h$ este adresa din memoria primară a primei locaţii din pagina fizică, iar V este

deplasamentul din cadrul paginii fizice U. Deorece procesului îi sunt alocate 2^j pagini fizice, vom avea $H = 2^{j+h}$ locaţii de memorie fizică care pot fi utilizate de proces. Corespondenţa dintre adresele virtuale şi cele fizice are forma $\Psi_t : [0,...,G-1] \rightarrow < U,V > \cup \{\Omega\}$.

Deoarece fiecare pagină are aceeaşi dimensiune c, adresa virtuală i poate fi convertită într-un număr de pagină şi un deplasament în cadrul paginii, numărul de pagină fiind I div c, iar deplasamentul i mod c. Dacă considerăm reprezentarea binară a adresei, atunci numărul de pagină poate fi obţinut prin deplasare spre dreapta a adresei cu h poziţii, iar deplasamentul prin extragerea primilor g biţi cei mai puţin semnificativi din adresă. Deci fiecare adresă virtuală, repectiv adresă fizică va fi de forma (p,d), respectiv de forma (f,d), unde p este numărul paginii virtuale, f este numărul paginii fizice, iar d este adresa (deplasamentul) în cadrul paginii.

Orice pagină virtuală din spaţiul de adrese al procesului, poate fi încărcată în oricare din paginile fizice din memoria internă alocate acestuia. Acest lucru este realizat printr-o componentă hardware numită MMU(**M**emory **M**anagement **U**nit), care implementează funcţia Ψ_t (figura 2). Dacă este referenţiată o locaţie dintr-o pagină care nu este încărcată în memoria internă, MMU opreşte activitatea CPU, pentru ca SO să poată executa următorii paşi:

1. Procesul care cere o pagină neîncărcată în memoria internă este suspendat.
2. Administratorul memoriei localizează pagina respectivă în memoria secundară.
3. Pagina este încărcată în memoria internă, eventual în locul altei pagini, dacă în memoria internă nu mai există pagini fizice libere alocate procesului respectiv şi în acest caz, tabela de pagini este modificată.
4. Execuţia procesului se reia din locul în care a fost suspendat.

Fiecare proces are propria lui tabelă de pagini, în care este trecută adresa fizică a paginii virtuale, dacă ea este prezentă în memoria operativă. La încărcarea unei noi pagini virtuale, aceasta se depune într-o pagină fizică liberă. Deci, în memoria operativă, paginile fizice sunt distribuite în general necontinuu, între mai multe procese. Spunem că are loc o **proiectare a spatiului virtual peste cel real.**

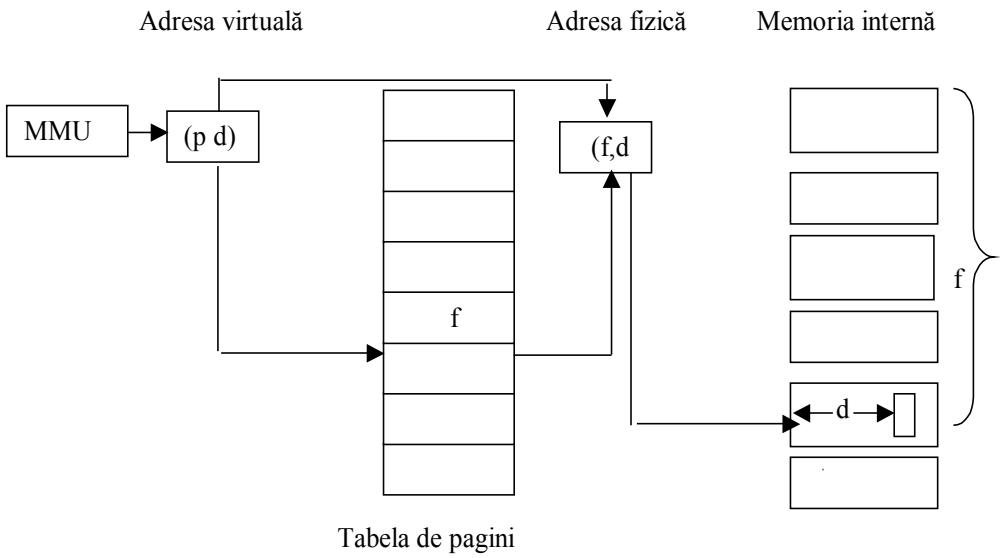

Figura 2 Translatarea unei pagini virtuale într-una fizică

Acest mecanism are avantajul că foloseşte mai eficient memoria operativă, fiecare program ocupând numai memoria strict necesară la un moment dat. Un alt avantaj este posibilitatea folosirii în comun, de către mai multe programe, a instrucţiunilor unor proceduri. O procedură care permite acest lucru se numeste **procedură reentrantă.**

Evidenţa paginilor virtuale încărcate în pagini fizice, este relizată printr-o tabelă [6]. Dacă prin M[0..m] notăm memoria operativă, prin j puterea lui 2 (numărul de biţi) care dă lungimea unei pagini, prin TP adresa de start a tabelei de pagini, atunci pe baza adresei virtuale (p,d) se poate determina locaţia de memorie fizică corespunzătoare, care este $M[TP + p] * 2^j + d$, dacă locaţia virtuală respectivă este încărcată în memorie.

Formula anterioară este valabilă atunci când tabela de pagini ocupă un spaţiu în memoria operativă. Tabela de pagini poate fi privită ca o funcţie, care are ca argument numărul de pagină virtuală şi care determină numărul de pagină fizică. Pe baza deplasamentului se determină locaţia din memoria fizică unde se va încărca locaţia virtuală referenţiată. Această metodă ridică două mari probleme:

1. Tabela de pagini poate avea un număr mare de intrări. Calculatoarele moderne folosesc adrese virtuale pe cel puţin 32 de biţi. Dacă, de exemplu o pagină are dimensiunea de 4K, atunci numărul intrărilor în tabelă este mai mare decât un million. În cazul adreselor pe 64 de biţi numărul intrărilor în tabelă va fi, evident mult mai mare.

2. Corespondenţa dintre locaţia de memorie virtuală şi cea din memoria fizică trebuie să se realizeze cât mai rapid posibil; la un moment dat, o instrucţiune cod maşină poate referenţia una sau chiar două locaţii din memoria virtuală.

Pentru rezolvarea aceastei probleme, s-au încercat mai multe probleme. O primă metodă constă în **folosirea de regiştri ai CPU** pentru memorarea intrărilor în tabelă, soluţie care asigură un acces rapid dar care este foarte costisitoare.

O a doua metodă propune ca **tabela de index** să fie încărcată în memoria primară, folosind un registru pentru a memora adresa de început a tabelei. Această metodă permite ca tabela de index să fie încărcată în zone diferite din memorie, prin modificarea conţinutului registrului.

O metodă mult mai eficientă, care înlocuieşte căutarea secvenţială cu cea arborescentă este **organizarea tabelei pe mai multe niveluri.** Astfel, o adresă virtuală pe 32 de biţi de exemplu, este un triplet (Pt_1, Pt_2, d), primele două câmpuri având o lungime de 10 biţi, iar ultimul de 12 biţi. În figura 3 este ilustrată o astfel de tabelă de pagini.

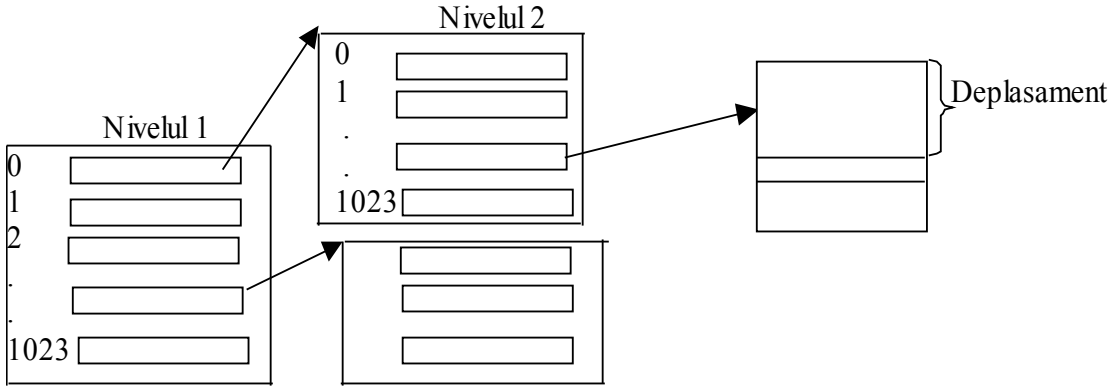

Figura 3. Tabela de pagini pe două niveluri.

Observăm că numărul de intrări în tabela de nivel 1 este de 2^{10}=1024, iar dimensiunea unei pagini este de 2^{12}=4x2^{10}=4K. Fiecare intrare în această tabelă conţine un pointer către o tabelă a nivelului 2. Când o adresă virtuală este prezentată MMU, se extrage valoarea câmpului PT_1, care este folosit ca index în tabela de la nivelul 1. Pe baza acestuia, se găseşte adresa de început a uneia dintre tabelele de la nivelul 2. Câmpul PT_2 va fi un index în tabela de la nivelul 2 selectată, de unde se va lua numărul de pagină fizică corespunzător numărului de adresă virtuală conţinut în adresa referenţiată. Pe baza acestui număr şi a deplasamentului (d), se determină locaţia de memorie fizică în care va fi încărcată respectiva locaţie din memoria virtuală. În condiţiile specificate, o tabelă de la nivelul 2 va gestiona o zonă de memorie de capacitate 1024 x 4 K=4 M.

Pe lângă câmpul care conţine numărul de pagină fizică, intrarea respectivă mai conţine şi alte câmpuri, dintre care cele mai semnificative sunt:

- Bitul **prezent/absent** indică faptul că pagina respectivă este/ sau nu (setat pe 1/setat pe 0) încărcată în memoria fizică.
- Biţii de **protecţie** specifică ce fel de tip de acces este permis. În forma cea mai simplă, acest câmp este de 1 bit, setat pe 0 pentru permisiune de citire şi scriere şi 1 numai pentru citire. Sistemele moderne folosesc 3 biţi de protecţie, unul setat pentru permisiunea de citire, unul pentru scriere şi unul care specifică dreptul de execuţie.
- Bitul **modificat** indică dacă în pagina respectivă s-a scris sau nu.

- Bitul **referit** indică dacă pagina respectivă a fost referenţiată sau nu.
- Bitul "caching disabled/enabled" indică posibilitatea încărcării paginii respective în memoria cache.

Observaţie. Tabela de pagini se poate organiza şi pe 3 sau 4 niveluri, în funcţie de dimensiunea memoriei virtuale şi a celei fizice. Dacă tabela are o dimensiune mare, ea poate fi păstrată(cel puţin parţial) şi pe un disc rapid.

[4] http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.mspx
[5] "Advanced Windows" by Jeffrey Richter, Microsoft Press
[6] http://en.wikipedia.org/wiki/Virtual_address_space

1.5. Alocare segmentată.

Din punctual de vedere al utilizatorului**,** o aplicaţie este formată dintr-un program principal şi o mulţime de subprograme(funcţii sau proceduri). Aceste folosesc diferite structuri de date (tablouri, stive etc), precum şi o mulţime de simboluri (variabile locale sau globale). Toate aceste sunt identificate printr-un nume [4].

Pornind de la această divizare logică a unui program, s-a ajuns la o metoda de alocării **segmentare** a memoriei. Spre deosebire de metodele de alocare a memoriei bazate pe partiţionare, unde fiecărui proces trebuie să i se asigure un spaţiu contiguu de memorie, mecanismul de alocare segmentată, permite ca un proces să fie plasat în zone de program distincte, fiecare dintre ele conţinând o entitate de program, numit **segment**. Segmentele pot fi definite explicit prin directive ale limbajului de programare sau implicit prin semantica programului. De exemplu, un compilator de Pascal poate crea segmente **de cod** pentru fiecare procedură sau funcţie, segmente pentru **variabilele globale,** segmente pentru **variabilele locale,** precum şi **segmente de stivă**. Deosebire esenţială dintre alocarea paginată şi cea segmentată este aceea că segmentele sunt de **lungimi diferite.**

În mod analog cu alocarea paginată, **o adresă virtuală [7]** este o pereche (s,d), unde s este numărul segmentului iar d este adresa din cadrul segmentului. Adresa reală (fizică) este o adresă obisnuită. Transformarea unei adrese virtuale într-o adresă fizică, se face pe baza unei **tabele de segmente**. Fiecare intrare în acestă tabelă este compusă dintr-o **adresă de bază**(adresa fizică unde este localizat segmentul în memorie) şi **lungimea segmentului(limită)**. În figura 4 este ilustrat modul de utilizare a tabelei de segmente.

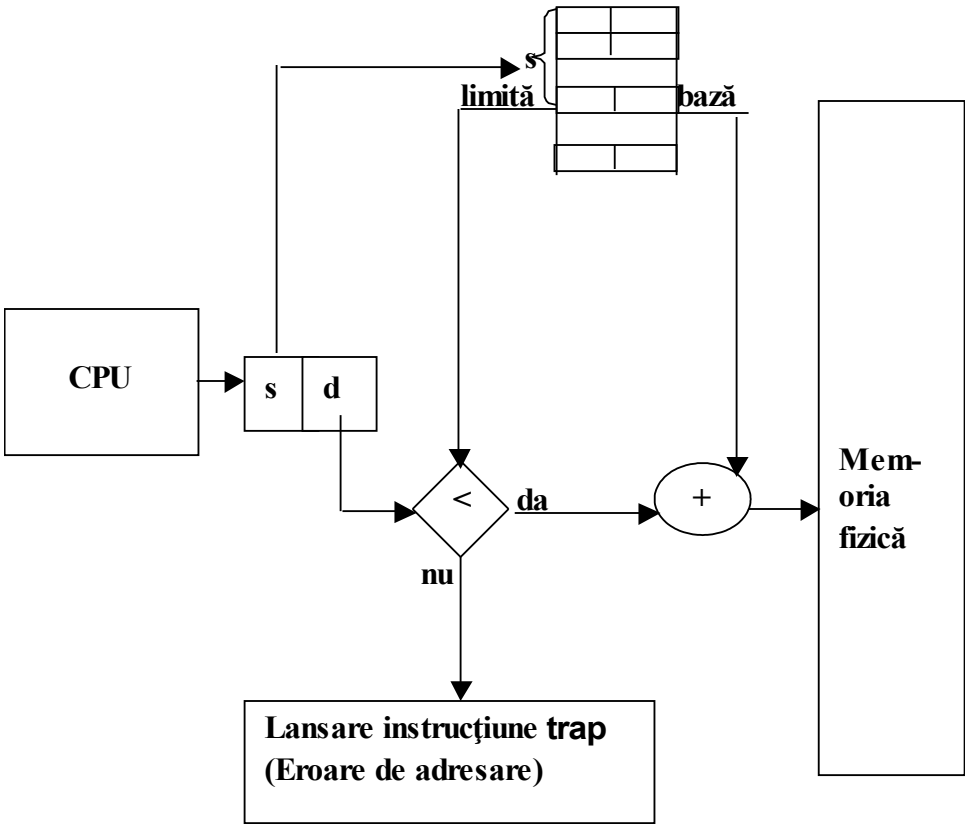

Figura 4. Translatarea adreselor segmentate

Componenta s a adresei virtuale, este un indice în tabela de segmente. Valoarea deplasamentului d trebuie să fie cuprinsă între 0 şi lungimea segmentului respectiv(în caz contrar este lansată o instrucţiune trap, care generează o întrerupere). Dacă valoarea d este validă, ea este adăugată adresei de început a segmentului şi astfel se obţine adresa fizică a locaţiei respective.

[4] http://www.microsoft.com/whdc/system/platform/server/PAE/PAEmem.mspx
[7]http://john0312.wordpress.com/2007/05/27/about-memory-management-and-memory-allocation/

1.6. Alocarea segmentată cu paginare.

Cele două metode de alocare a memoriei au avantajele şi dezavantajele lor. În cazul segmentării poate să apară fenomenul de fragmentare. În cazul paginării, se efectuează o serie de operaţii suplimentare de adresare. De asemenea, unele procesoare folosesc alocarea paginată (Motorola 6800), iar altele folosesc segmentarea (Intel 80x86, Pentium). Ideea segmentării cu paginare, este aceea că alocarea spaţiului pentru fiecare segment să se facă paginat. Această metodă este utilizată de sistemele de operare actuale [5].

Spaţiul de adrese virtuale al fiecărui proces este împărţit în două partiţii; prima partiţie este utilizată numai de către procesul respectiv, pe când cealaltă partiţie, este partajată împreună cu alte procese. Informaţiile despre prima partiţie, respective a doua

partiţie sunt păstrate în **descriptorul tabelei locale (LDT – L**ocal **D**escriptor **T**able**),** respectiv **descriptorul tabelei globale (GDT – G**lobal **D**escriptor **T**able**).** Fiecare intrare în LDT/GDT este reprezentată pe 8 octeţi şi conţine informaţii despre un anumit segment, printre care adresa de început şi lungimea acelui segment.

Adresa virtuală este o pereche (selector, deplasament) [7]. Selectorul este reprezentat pe 16 biţi şi este un triplet (s, g, p) în care s este numărul de segment (13 biţi), g specifică dacă segmental este din LDT sau GDT şi p specifică protecţia. Deplasamentul este reprezentat pe 32 de biţi şi specifică adresa locală a unui octet în cadrul segmentului.

Calculatorul are 6 **regiştri de segmente**, ceea ce permite ca un proces să adreseze 6 segmente în orice moment. De asemenea are 6 regiştri pe 8 octeţi, care păstrează descriptori pentru LDT sau GDT. Acest mod de organizare permite accesarea rapidă a entităţilor respective.

Translatarea unei adrese virtuale în adresă fizică se realizează astfel:

- registrul de segment conţine un pointer către intrarea din LDT sau GDT corespunzătoare acelui segment;
- de aici se iau adresa de început şi deplasamentul;
- dacă adresa respectivă este validă, valoarea deplasamentului este adăugată la adresa de început şi astfel se obţine o adresă liniară, care apoi este translatată într-o adresă fizică.

Aşa cum am specificat mai devreme, fiecare segment este împărţit în pagini, fiecare pagină având o dimensiune de 4 Ko. Datorită dimensiunii posibile a unui segment, tabelele de paginii poate avea un număr mare de intrări (până la 1 000 000). Deoarece fiecare intrare are o lungime de 4 octeţi, rezultă că o astfel de tabelă poate avea o dimensiune de 4 Mo, deci ea nu poate fi păstrată ca o listă liniară în memoria internă. Soluţia este de a folosi o tabelă pe două niveluri. Mecanismul de translatare a adresei este similar cu cel utilizat în cazul alocării paginate a memoriei.

[5] "Advanced Windows" by Jeffrey Richter, Microsoft Press
[7]http://john0312.wordpress.com/2007/05/27/about-memory-management-and-memory-allocation/

1.7. Memoria cu acces rapid(„cache”)

Memoria **cache** conţine copii ale unor blocuri din memoria operativă. Când CPU încearcă citirea unui cuvânt din memorie, se verifică dacă acesta există în memoria **cache**. Dacă există, atunci el este livrat CPU. Dacă nu, atunci el este căutat în memoria operativă, este adus în memoria cache împreună cu blocul din care face parte, după care este livrat CPU. Datorită vitezei mult mai mari de acces la memoria cache, randamentul general al sistemului creşte [8].

Memoria cache este împărţită în mai multe părţi egale, numite **sloturi**. Un slot are dimensiunea unui bloc de memorie, a cărui dimensiune este o putere a lui 2. Fiecare slot conţine o zonă care conţine blocul de memorie operativă depus în slotul respectiv. Problema care se pune, este modul în care se face corespondenţa dintre blocurile din memoria operativă şi sloturile din memoria cache, precum şi politicile de înlocuire a

sloturilor din memoria cache, cu blocuri din memoria fizică. Spunem că are loc o **proiecţie** a spaţiului memoriei operative în cel al memoriei cache.

Proiecţia directă. Dacă c indică numărul total de sloturi din memoria cache, a este o adresă oarecare din memoria operativă, atunci numărul s al slotului în care se proiectează adresa a este s= a mod c. Dezavantajul metodei constă în faptul că fiecare bloc are o poziţie fixă în memoria cache [8]. Dacă, de exemplu se cer accese alternative la două blocuri care ocupă acelaşi slot, atunci trebuie efectuate mai multe operaţii de înlocuire a conţinutului slotului care corespunde celor două blocuri.

Proiecţia asociativă. Fiecare bloc de memorie este plasat în oricare dintre sloturile de memorie cache, care sunt libere. Înlocuirea conţinutului unui slot, cu conţinutul altui bloc din memoria operativă, se face pe baza unuia dintre algoritmii NRU, FIFO sau LRU.

Proiecţia set-asociativă combină cele două metode prezentate anterior. Memoria cache este împărţită în i seturi, un set fiind compus din j sloturi. Avem relaţia c=i x j. Dacă a este o adresă de memorie, numărul k al setului în care va intra blocul, este dat de:

$$K=a \bmod i.$$

Cunoscând numărul setului, blocul va ocupa unul dintre sloturile acestui set, pe baza unuia dintre algoritmii de înlocuire prezentaţi anterior.

[8] http://en.wikipedia.org/wiki/Kernel_(computer_science)

1.8. Concluzii

Pentru a fi executat, orice program trebuie să fie încărcat în memorie, împreună cu toate informaţiile pe care le prelucrează. Astfel, memoria internă este o componentă esenţială a sistemului de calcul.

Modalităţile de gestiune a memoriei interne au evoluat odată cu progresele tehnologice ale sistemelor de calcul, precum şi evoluţia sistemelor de operare. Dacă la început, numai un singur program putea fi încărcat în memoria internă, apariţia unor sisteme de operare mai evoluate, a făcut posibilă apariţia multiprogramării.

Calculatoarele moderne folosesc conceptul de memorie virtuală, ce reprezintă extinderea spaţiului de adrese al memoriei fizice. Ea se sprijină pe ideea asocierii de adrese fizice obiectelor programului în timpul execuţiei lui. Compilatorul şi link-editorul creează un modul absolut, căruia încărcătorul îi asociază adrese fizice înainte ca programul să fie executat. Facilităţile hardware permit administratorului memoriei să încarce automat porţiuni ale spaţiului de adrese virtuale în memoria primară, în timp ce restul spaţiului de adrese este păstrat în memoria secundară.

Sistemele cu paginare transferă blocuri de informaţie de dimensiune fixă între memoria secundară şi cea primară. Datorită dimensiunii fixe a paginii virtuale şi a celei fizice, translatarea unei adrese virtuale într-una fizică devine o problemă reletiv simplă, prin mecanismul tabelei de pagini sau al memoriei virtuale.

Politicile de încărcare şi de înlocuire a paginilor, permit ca paginile să fie încărcate în memorie, numai atunci când se face o referinţă la o locaţie de memorie pe care o conţin.

Politicile de plasare şi de înlocuire a paginilor, definesc reguli prin care se optimizează transferul de informaţii din memoria virtuală în memoria primară şi reciproc.

Segmentarea este o alternativă la paginare. Ea diferă de paginare prin faptul că unităţile de transfer dintre memoria secundară şi cea primară variază în timp. Dimensiunea segmentelor trebuie să fie în mod explicit definită de către programator sau sistem. Translatarea unei adrese virtuale segmentate într-o adresă fizică este mult mai complexă decât translatarea unei adrese virtuale paginată. Segmentarea se sprijină şi pe sistemul de fişiere, deoarece segmentele sunt stocate pe memoriile externe sub formă de fişiere. Segmentarea cu paginare valorifică avantajele oferite de cele două metode, fiind utilizată de sistemele de calcul moderne.

Memoriile cu acces rapid cresc performanţele unui sistem de calcul. Metodele de proiecţie a spaţiului din memoria internă în memoria cache, permite ca orice locaţie fizică să fie accesată mult mai rapid, pentru a fi utilizată de unitatea centrală.

2) Apelurile de sistem pentru gestiunea memoriei:comparatie la BSD(Berkeley Software Distribution), Windows si Linux

Aceasta parte este un studiu al sistemelor de gestionare al memoriei, care se gasesc intr-un sistem de operare. Vom incepe cu o scurta introducere a sistemelor de gestionare a memoriei, si apoi vom compara gestiunea memoriei la 3 dintre sistemele actuale: BSD 4.4, Windows 2000 si Linux 2.4.

2.1. Introducere

Vom compara sub-sistemele de gestionare al memoriei ale urmatoarelor sisteme: BSD 4.4, Windows 2000 si Linux 2.4.

BSD 4.4 a fost ales deoarece este o versiune UNIX reprezentativa ce include principii importante de design ale unui sistem de operare, pe care se bazeaza unele dintre sistemele de operare actuale: FreeBSD [3], NetBSD [5] si OpenBSD [6]. Mai mult despre acestea, este documentat in lucrarea [12].

Windows 2000 a fost ales deoarece este un sistem de operare folosit ca desktop, in special de incepatori, si care a evoluat intr-un sistem de operare foarte avansat.

Linux 2.4 [4], a fost ales deoarece devine din ce in ce mai popular pe zi ce trece, si pare ca va juca un rol important in viitor. Nu vom fi foarte interesati de caracteristicile de performanta ale acestor sisteme in lucrarea aceasta, ci ne vom indrepta atentia catre designul si arhitectura lor.

[3] The FreeBSD Project. Mai multe informatii pot fi gasite la http://www.freebsd.org.
[4] The Linux Project..Mai multe informatii pot fi gasite la http://www.linux.org.
[5] The NetBSD Project.Mai multe informatii pot fi gasite la http://www.netbsd.org.
[6] The OpenBSD Project.Mai multe informatii pot fi gasite la http://www.openbsd.org.
[12] M.K. McKusick et al. The Design and Implementation of 4.4BSD Operating System. Addison-Wesley, 1996.

2.2. Sisteme de gestionare al memoriei

Se repeta din alte subcap de mai sus.

Sistemul de gestionare al memoriei este una dintre cele mai importante parti ce alcatuiesc un sistem de operare. Functia de baza a acestui sistem, este de a administra ierarhia memoriilor RAM si a hard-diskurilor aflate intr-o masina. Una dintre cele mai importante functii este de a aloca si dealoca memoria proceselor care se ocupa de logistica, si implementarea memoriei virtuale, prin folosirea hard-diskului ca RAM aditional. Sistemul de memorie trebuie optimizat la maxim, deoarece performanta sa afecteaza performanta totala, cat si viteza de functionare a intregului sistem.

2.2.1. Memoria virtuala

Un concept important in cadrul sistemelor MM, il constituie memoria virtuala. In zilele de inceput ale informaticii, oamenii de stiinta au observat o crestere a necesitatii memoriei utilizata de fiecare program in parte, astfel nascandu-se conceptul de Memorie Virtuala. Ideea care sta la baza conceptului, este de a da unui program iluzia de prezenta a unei mari cantitati de memorie, pe care acesta il poate folosi. Kernelul va realiza acest lucru, prin folosirea unei unitati aditionale(hard-diskul), pentru a umple golul de memorie.

Pentru ca sistemul de memorie virtuala sa functioneze, vom avea nevoie de o functie de mapare, care va face translatarea adreselor, convertind adresa virtuala intr-una fizica. Adresa virtuala este adresa pe care o aplicatie o foloseste pentru a se referi la o locatie a memoriei, iar adresa fizica este locatia actuala a memoriei. Aceasta functie este in general una de Paginare, sau de Segmentare, sau ambele, in functie de Kernel, arhitectura procesorului si starea sa.

2.2.2. Paginarea

In Paginare, spatiul de adresa(atat real cat si virtual), este impartit in pagini de marime fixa(desi pot fi si de marimi diferite [14]). Aceste pagini pot fi folosite individual, si puse in diferite locuri in cadrul memoriei fizice si hard-diskului. Translatarea adreselor este de fapt facuta de Unitatea de Gestionare a Memoriei (MMU) a procesorului, prin folosirea unei tabele de paginare, cum este aratat in figura 1.

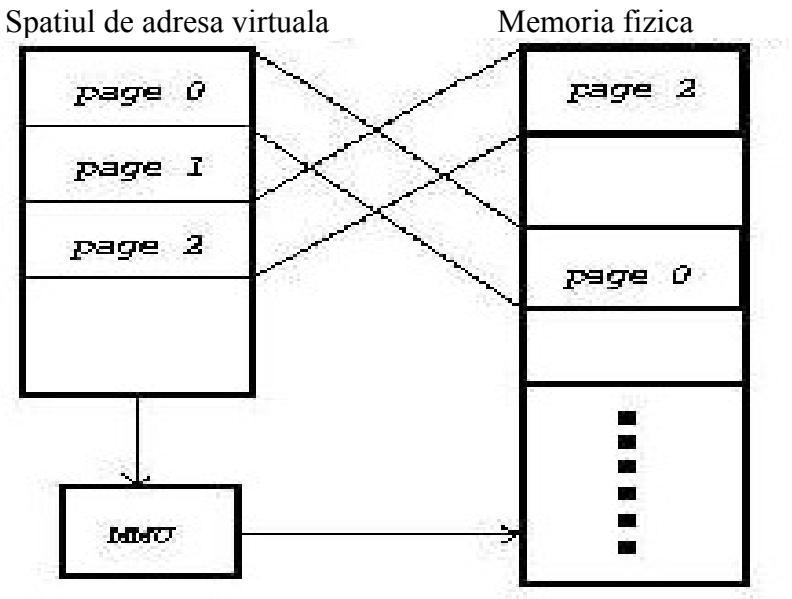

Figura 1 Tabela de paginare

Tabelele de paginare specifica maparea intre paginile virtuale si cele fizice, pe care pagina de memorie virtuala o ocupa cu cea fizica. MMU converteste adresa de

memorie virtuala cu una fizica, ce consta intr-un numar al cadrului paginii, si un punct de plecare in acea pagina. Poate fi aplicata protectie pe fiecare pagina, folosindu-se sistemul pagina cu pagina. De vreme ce adresa virtuala este imensa comparativ cu memoria fizica, va trebuie sa folosim hard-diskul pentru a stoca pagini ce nu pot fi stocate in memoria fizica. Asocierea cu fiecare pagina in tabela de paginare, este prea putin pentru a ne da seama daca pagina este cu adevarat prezenta in memoria fizica, sau nu. Daca pagina nu este prezenta in memoria fizica, device-ul va genera o eroare de exceptie de pagina. Aceasta exceptie este tratata de software, care aduce pagina ceruta de pe hard-disk, inapoi in memoria fizica, sau, daca este invalida, va genera o eroare.

Coffman si Denning [2], caracterizeaza sistemele de paginare, dupa 3 reguli importante:

1. Cand sistemul incarca pagini in memorie – regula de aducere.
2. Cand sistemul plaseaza pagini in memorie – regula de plasare.
3. Cum sistemul alege paginile ce sunt inlaturate din memoria principala, cand ele nu sunt prezente la o cerere de plasare – regula de inlocuire a paginii.

Regula de plasare este importanta numai pentru a optimiza unele comportamente [16]. Deci practic, comportamentul unui sistem depinde doar de regulile de aducere si de plasare. La majoritatea sistemelor moderne, pentru regula de aducere este folosit un sistem de paginare la cerere, in cadrul caruia sistemul aduce o pagina in memorie numai cand ii este cerut, desi cateodata, prepaginarea unor pagini se asteapta a fi facuta. Tinand cont de regula de inlocuire a paginii, de-a lungul anilor au fost dezvoltati multi algoritmi. Comparatiile intre performantele diferitilor algoritmi de inlocuire, pot fi gasite in multe lucrari, cum ar fi [15].

[2] E. G. Coffman and P. J. Denning. Operating Systems Theory. Prentice Hall, 1973.
[14] Yousef A. Khalidi, Madhusudhan Talluri, Michael N. Nelson, and Dock Williams. Virtual memory support for multiple page sizes. InWorkshop on Workstation Operating Systems,pages 104–109, 1993.
[15] W.F. King. Analysis of demand paging algorithms. In In International Federation for Information Processing Conference Proceedings,pages 485–490, 1972.
[16] T. Romer. Using virtual memory to improve cache and TLB performance. Technical Report TR-98-05-03, 1998.

2.3. Comparatie a sub-sistemelor de gestionare a memoriei ale sistemelor Windows 2000, Linux 2.4 si BSD 4.4.

Sistemul VM al BSD 4.4, este bazat pe codul VM al Mach 2.0, 2.5 si 3.0. Windows 2000 a fost dezvoltat pe baza unei serii lungi de sisteme de operare, incepand cu MSDOS. Linux 2.4 a fost realizat de hackeri, infiintat original de Linux Torvalds.

In afara de susrele citate in aceasta lucrare, mai mult detalii despre subiect pot fi gasite in lucrarile [17], [18] si [21].

In loc de a descrie fiecare sistem MM, al fiecarui sistem de operare in parte, ceea ce ar fi un lucru foarte greu de facut, vom compara aici cateva dintre partile lor principale.

Toate cele 3 sisteme au sisteme MM moderne, si au surprinzator de multe in comun. Structura de date este similara la toate 3, si caracteristicile fiecaruia sunt aproximativ similare. Cateva din asemanarile acestor sisteme sunt enumerate mai jos:

- Stratul de Abstractie Hardware – Toate sistemele de operare au un strat, denumit Strat de Abstractie Hardware (HAL), care face munca ce tine de sistem, inlesnind astfel codarea restului kernelui in platforme independente. Asta inlesneste portarea pe alte platforme.
- Copierea la scriere – Cand o pagina este sharuita, sistemul foloseste o singura pagina, in timp ce ambele procese impart aceeasi copie a paginii. Cu toate acestea, cand un proces face o actiune de scriere pe acea pagina, este facuta o copie personala a acelei pagini, pe care ulterior procesul o poate folosi individual. Aceasta da o mai buna eficienta procesului.
- Paginarea in umbra – Un obiect umbra este creat pentru un obiect original, in asa fel incat obiectul umbra are cateva din paginile sale modificate fata de original, insa imparte restul cu originalul. Acesta este rezultatul copierii la scriere.
- Fisiere mapate in memorie – Un fisier poate fi mapat in memorie, care poate fi ulterior folosit cu instructiuni simple de citire/scriere.
- Comunicarea intre procese – Fisierelelor mapate in memorie li se da permisiunea de a fi sheruite intre procese, formanduse astfel o metoda de comunicare intre procese.

In urmatoarele capitole, vom compara aceste sisteme dupa cateva aspecte.

2.3.1. Structuri de date pentru descrierea spatiului unui proces

Acum vom studia structurile de date folosite de sisteme pentru a intretine si a urmari memoria virtuala.

2,3.1.1. BSD 4.4

Structurile de date pentru BSD 4.4 sunt aratate in figura 2.

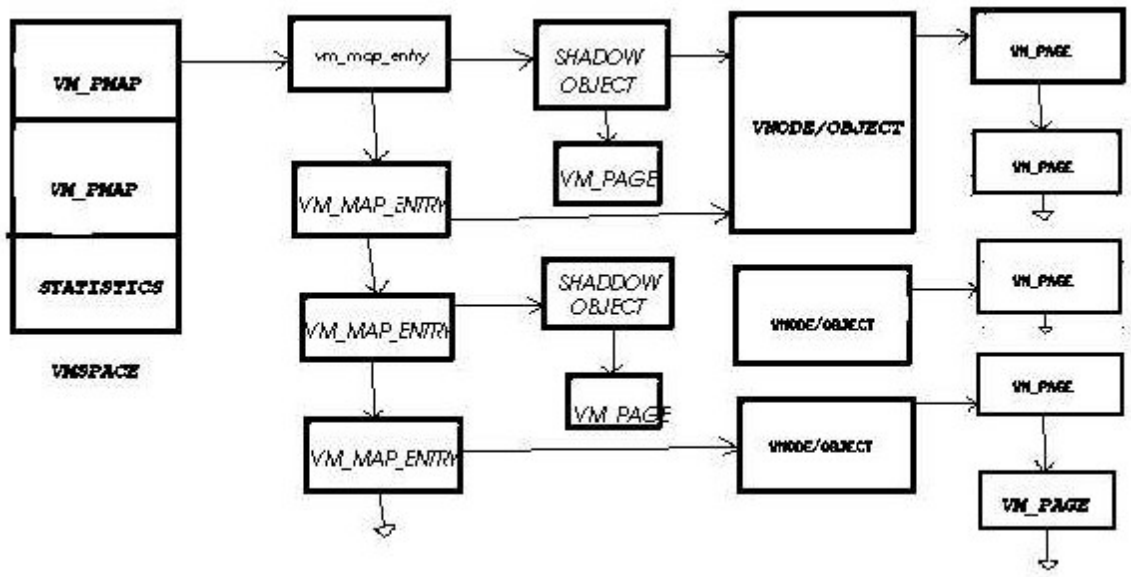

Figura 2 Structurile de date pentru BSD 4.4 pentru gestiunea procesului memoriei virtuale

Aceasta structura este repetata pentru fiecare proces, deoarece fiecare proces are propriul spatiu de adrese virtuale.

Structurile principale sunt:

- vmspace
- vm_map
- vm_map_entry
- object
- shadow object
- vm_page

Structura vm_map este un strat dependent hardware. Are grija de administrarea memoriei la cel mai mic nivel, in general avand grija de diferitele metode pe care procesoarele le au pentru programarea memoriei virtuale, etc. Punand codul dependent hardware intr-un singur modul, face restul de cod VM independent hardware, ceea ce duce la un design modular, si face relativ usoara portarea codului pe diferite arhitecturi. Structura vm_map contine un pointer catre vm_pmap, si un pointer catre lantul vm_map_entry. O singura vm_map_entry este folosita pentru fiecare regiune contigua a memoriei virtuale, care are aceleasi drepturi de protectie si mostenire. Aceasta arata apoi catre un lant de obiecte vm_object. Ultimul in lista este obiectul in sine, restul fiind obiecte umbra. Despre obiectele umbra vom vorbi in alt capitol de comparatie. Obiectul are un pointer catre o lista de obiecte vm_page, care reprezinta de fapt paginile de memorie in sine. Aceste pagini in memoria principala sunt considerate ca un cache al hard-diskului, un concept de memorie virtuala. Obiectul vm_object contine si pointeri catre functiile ce vor face operatii pe el.

2.**3.1.2. Windows**

Structurile de date folosite la Windows NT sunt prezentate in figura 3.

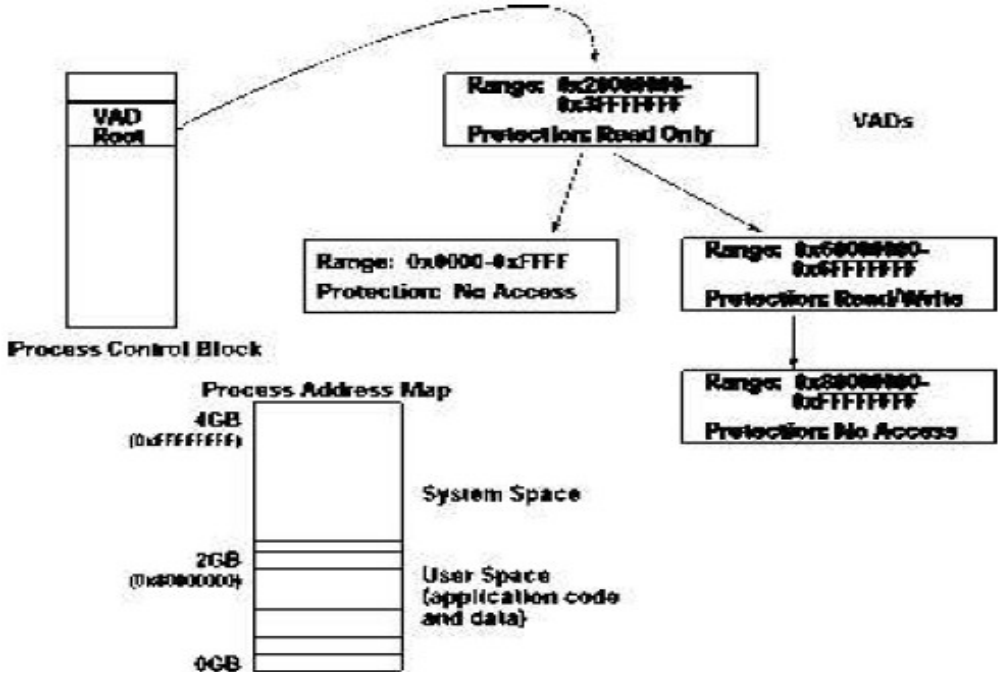

Figura 3 Structurile de date folosite la Windows NT pentru gestionarea memoriei virtuale

In locul unei liste legate, sistemul Windows NT pastreaza lista intr-o forma de arbore. Fiecare nod al arborelui este numit Descriptor de Adresa Virtuala (VAD). Fiecare VAD denota o raza de adrese, care au aceiasi parametrii de protectie, si informatie de stare finalizata..Arborele este de asemenea balansat, ceea ce inseamna ca profunzimea lui este mentinuta la un minim. Aceasta implica evident, ca timpul de cautare, va fi relativ scazut. VAD-ul marcheaza fiecare nod ca fiind, rezervat, liber sau ocupat. Ocupate sunt cele care au fost deja folosite (datele au fost deja mapate pe ele). Nodurile marcate ca libere nu au fost folosite, iar cele rezervate sunt nodurile care inca nu sunt disponibile pentru mapare, pana ce rezervarea nu va fi ridicata total. Rezervarile sunt folosite in cazuri speciale.

2.**3.1.3. Linux**

Linuxul implementeaza structura de date a memoriei virtuale, intr-o maniera similara cu a UNIX-ului. Pastreaza o lista legata de obiecte vm_area_structs. Acestea sunt structuri ce reprezinta zone de memorie continua care eu aceiasi parametrii de protectie. Aceasta lista este cautata de fiecare data cand o pagina este cautata, deoarece contine o locatie specifica. Structura de asemenea tine contul razei de adrese catre care mapeaza, modul de protectie, daca este inradacinata in memorie, si directia in care se indreapta. De asemenea inregistreaza shi daca zona este publica sau privata. Daca numarul de intrari depaseste, de obicei 32, atunci lista este transformata intr-un tree. Aceasta este o abordare destul de buna, folosind cea mai buna structura, in cele mai bune situatii. Cam putin

2.3.2. Distribuirea spatiului de adrese a proceselor

Toate sistemele organizate in tip ,arbore distribuie spatiul de adrese virtual a procesului in mod similar. O mare parte din el e folosit de kernel, in timp ce procesul foloseste o parte mai mica. Deci practic in timp ce comutam un proces, trebuie sa comutam intrarile in pagina ale partii mici, pentru ca partea mare sa ramana la fel. La Linux si BSD, de obicei 3 GB sunt folositi pentru proces, si 1 GB ii este acordat kernelului. La Windows, ambele folosesc 2 GB.

2.3.3. Inlocuirea paginilor

Inlocuirea paginilor este o parte importanta in orice sistem MM. In sine, inlocuirea paginilor alege ce pagina sa fie data afara din memorie, oricand se pune problema necesitatii de mai multa memorie libera.

Algoritmul ideal de inlocuire a paginilor, consta in inlaturarea paginii a carei folosire este cea mai indepartata. Aceasta va cauza cele mai putine erori de pagina, astfel economisindu-se timp la schimbarea paginilor, astfel imbunatatind performanta sistemului. Dar nu este posibil a se sti ce pagini vor fi acesate in viitor, deci acest algoritm este imposibil de implementat.

Vom vedea in urmatoarele randuri cum este gandit fiecare sistem, referitor la inlocuirea paginilor.

2.3.3.1. BSD 4.4

Sistemul foloseste o metoda de cerere a paginarii, pentru regula de aducere, si un algoritm de aproximare, bazat pe ideea celei mai putin recent folosite pagini.

Cererea de paginare inseamna ca paginile vor fi aduse in memorie, numai atunci cand va fi nevoie de ele. In circumstante practice insa, paginile care se asteapta sa fie folosite, sunt si aduse in memorie initial.

Pentru scoaterea acestor pagini, sistemul foloseste un sistem global de inlocuire. Global inseamna ca sistemul alege pagina ce trebuie inlaturata, indiferent de procesul ce foloseste acea pagina, ceea ce inseamna ca toate paginile tuturor proceselor sunt considerate egale, deci, trebuie folosit un alt parametru pentru a face diferenta.

Sistemul imparte memoria principala, in 4 liste:

- Legate – aceste pagini sunt inchise. Aceste pagini nu pot fi mutate, si sunt de obicei folosite de kernel.
- Active – in aceasta lista sunt puse paginile ce sunt folosite.
- Inactive – paginile inactive, ce au continut cunoscut, insa nu au mai fost folosite de ceva timp
- Libere – pagini cu continut necunoscut, deci pot fi folosite imediat

Un daemon(Disk And Executive MONitor) pentru pagina este folosit pentru a mentine o cantitate de memorie libera in sistem. Acest proces este pornit la inceput in kernel, si ramane activ pana la inchiderea computerului. Telul acestui proces este de a mentine un numar minim de pagini libere, specific stocate in free_min(de obicei 5% din capacitatea memoriei). Mai exista o variabila, free_target, ce reprezinta de obicei

7% din capacitatea memoriei – daemonul isi inceteaza functionarea cand a obtinut free_targetul maxim de 7%. Numarul de pagini inactive trebuie si el mentinut de obicei la un 33% din capacitatea memoriei. Cu toate acestea, valoarea free_target-ului este ajustata de sistem in timp. Deci, de fiecare data cand memoria libera scade sub free_min, daemonul intra in functiune.

Daemonul incepe scanarea de la cele mai vechi pana la cele mai noi intrari in lista de inactive, si face urmatoarele pentru fiecare pagina:

- daca pagina este libera shi nu se face referire la ea, o muta in lista de Libere.
- daca la pagina face referire un proces activ, o muta din lista de inactive in lista de active.
- daca pagina este in proces de scriere, pentru moment o ignora.
- daca pagina nu este in proces de scriere, si nici nu este folosita de un proces, o rescriere pe disk.

Dupa scanare, daemonul verifica daca lista de inactive este mai mica decat inactive_target, si reincepe scanarea activelor, pentru a le aduce inapoi in lista de inactive.

Dea semenea exista si un concept de swap in BSD. Cand nu poate tine pasul cu erorile in timp ce isi exercita rolul, sau cand un proces devine inactiv pentru mai mult de 20 de secunde, daemonul intra in mod swap. In mod swap, daemonul ia procesul care ruleaza de cel mai mult timp, si il aduce complet inapoi pe hard-disk.

2.3.3.2. Windows

Sistemul folosit de Windows in acest caz este foarte sofisticat si complex,acest lucru fiind demonstrat in continuare..

Windowsul foloseste o cerere de paginare pe baza de clustere pentru aducerea paginilor, si algoritmul ceas pentru inlocuirea lor.

In cererea de paginare pe baza de clustere, paginile sunt aduse in memorie numai cand este nevoie de ele. Deasemenea, in loc sa aduca una, Windowsul de obicei aduce intre 1 si 8 pagini, numarul lor depinzand de starea sistemului.

Kernelul primeste 5 tipuri de erori de pagina:

- referinta la pagina nu a fost comisa (savarsita)
- s-a produs o incalcare a protectiei
- o pagina pusa in sistemul share (pusa la comun) a fost scrisa
- stiva trebuie sa creasca
- pagina la care se face referire exista, insa nu este mapata(tehnica de gestionare a memoriei care da impresia unei aplicatii ca are destula memorie ca sa ruleze)

Primele 2 erori sunt irecuperabile. A treia indica o incercare de scriere peste o pagina read-only. Trebuie copiata pagina aceasta in alta parte, iar originalul trebuie facut read/write. La a patra eroare trebuie cautata o pagina in plus.

Cel mai important lucru la Windows, este ca foloseste din plin conceptul workset(set de lucru). Acest concept este definit ca, cantitatea de memorie acordata in

acel moment unui proces, deci forta lui de munca, consta in paginile lui aflate in memoria principala.

Algoritmul ceas folosit de Windows este local. Cand se produce o eroare de pagina, acea pagina este adaugata la forta de munca. Insa, sistemul face si optimizari globale. De exemplu, mareste forta de munca pentru procesele care produce un set mare de erori, si reduce forta de munca la procesele in stare buna de functionare.

In loc sa isi continue functionarea in timp ce se produce o eroare, ca UNIX-ul, Windowsul are si el un daemon, numit Manager de Balanta. Acesta este invocat o data la o secunda, si verifica daca este indeajunsa memorie libera. Daca nu este, atunci invoca forta de munca(working set).

2.3.3.3. Linux

Riel [20] a muncit mult la VM-ul Linuxului in ultimii ani, si a faut imbunatatiri majore pentru Linux 2.4.

Linux foloseste un sistem de cerere de paginare, fara prepaginare. [19]

Pana la versiunea de kernel 2.2, Linuxul folosea versiunea NRU pentru inlocuirea paginilor, insa datorita neajunsurilor, au implementat in 2.4 metoda celei mai putin recent folosite pagini(LRU).

Linux 2.4 imparte paginile virtuale in 4 liste:

- Lista de Active
- Lista de Inactive-Murdare
- Lista de inactive-Curate
- Lista de Libere

Pentru a separa paginile care erau sortite evacuarii, s-a creat Lista de Inactive-Murdare. Normal, paginile active sunt in prima lista. Dar, cu trecerea timpului, daca unele pagini nu sunt active, dar varsta lor descreste si ajunge la 0, acestea devin candidati pentru evacuare.

Lista de inactive in BSD 4.4 are o tinta de 33%, pe care daemonul trebuie sa o mentina la acest nivel.

Pentru Linux 2.4, marimea listei de inactive a fost facuta mai dinamic. Acum, sistemul singur decide cate pagini inactive sa tina in memorie, in functie de situatie.

O alta optimizare prezenta in Kernelul Linuxului, este ca acum recunoaste continuu I/O, scade prioritatea paginii “din spate”, astfel transformand’o intr-un candidat pentru evacuare mai rapid.

Avand in vedere cele prezentate mai sus, Windows-ul este mai bun in ceea ce priveste apelurile de sistem pentru gestionarea memoriei.De asemenea Windows-ul are o interfata mai usoara. La Linux s-a renuntat la functionalitate pentru performanta, interfata este mai dificila, este greoi de utilizat in schimb merge bine.

[2] E. G. Coffman and P. J. Denning. OperatingSystems Theory. Prentice Hall, 1973.
[7] C. Cranor. Design and implementation of the uvm virtual memory system, 1998.

[8] Charles D. Cranor and Gurudatta M. Parulkar.The UVM virtual memory system. In Proceedings of the Usenix 1999 Annual Technical Conference,pages 117–130, 1999.
[9] Brazil Dept of Computer Science, Univesity of Sao Paulo. Linux 2.4 vm overview. http://linuxcompressed.sourceforge.net/vm24/.
[11] Rohit Dube. A comparison of the memory management sub-systems in freeBSD and linux.Technical Report CS-TR-3929, 1998.
[12] M.K. McKusick et al. The Design and Implementation of 4.4BSD Operating System. Addison-Wesley, 1996.
[13] T. Kilburn et al. One level storage system. IRE Transactions, EC-11(2):223–235, 1962.
[17] Mark Russinovich. Inside memory mangagement.Windows and .NET magazine, June June1998.
[18] David A. Solomon and Mark E. Russinovich.Inside Windows 2000. Microsoft Press, third edition, 2000.
[19] Andrew Tanenbaum. Modern Operating Systems.Prentice Hall, 2001.
[20] Rik van Riel. Page replacement in linux 2.4memory management. In Proceedings of theUSENIX Annual Technical Conference, 2001.
[21] Paul R. Wilson. The GNU/Linux 2.2 virtualmemory system, January 1999

2.4. FreeBSD, OpenBSD, NetBSD si UVM

Discutia de mai sus se refera la BSD4.4, care nu se mai prea foloseste. Insa, sunt multi succesori ai BSD4.4, ca FreeBSD, NetBSD, ce se bazeaza pe codul lui. Desi au multe in comun, ultimele versiuni au trecut printr-un proces de avansare semnificativ.

In privinta VM-ului, VM-ul FreeBSD-ului a fost dezvoltat si optimizat mult prin munca lui John Dyson, David Greenman, Matthew Dillon. [10]

NetBSD si OpenBSD au evoluat prin a folosi UVM. Autorii UVM-ului sustin ca designul este mai bun decat BSD4.4 si FreeBSD.

Designurile VM-ului FreeBSD si Linux au fost comparate mai sus. Insa comparatia de performanta intre VM-urile BSD-urilor si Linuxului, ramane inca discutabila.

[10] Matther Dillon. Design elements of the FreeBSD VM system. DaemonNews, January 2000.

2.5. Comentarii si concluzii

Cele 3 sisteme, isi au originile in locuri diferite.

Toate 3 sunt destul de moderne, si au la baza concepte teoretice solide, fiind potrivite pentru medii de productie.

Au multe in comun, si putine diferente, tehnic vorbind :fiecare are aceleasi principii de gestionare a memoriei insa le folosesc un pic diferit)Windows are visuals,Linux e mai performant dar mai greu de utilizat -acest lucru a fost prezentat mai sus.

Windows, fiind dezvoltat cu o motivatie materiala solida, a avut parte de un mare efort pe parte de design in dezvoltarea sa. In cazul celorlalte 2 sisteme, de

cele mai multe ori decizia a fost luata in favoarea simplitatii, defavorizand performanta, de cele mai multe ori.

In acest caz, Windowsul a evoluat, ajungand un cod complex si sofisticat, pe cand UNIX este simplu si elegant, insa in acelasi timp modern.

Insa, pentru userul de rand, Windowsul tinde sa devina alegerea mai usoara, in pofida crashurilor(prabusirilor) ocazionale.

Inca este nevoie de multa munca la sistemele cu sursa la vedere, si pe buna dreptate.

Este clar ca documentatia referitoare la sistemele cu sursa la vedere lipseste, in special una inteligibila si updatata. Se pare ca de fiecare data cand o documentatie este gata, programul evolueaza, shi ea devine veche.

Rata de dezvoltare a acestor sisteme cu sursa la vedere, intretinuta de hackeri din lumea intreaga, este ametitoare. Este de asteptat ca in viitor, aceste sisteme sa fie la egalitate, daca nu chiar mai bune, decat rivalii lor acuali.

Cuprins