FACULTATEA DE ELECTRONICA, TELECOMUNICATII SI TEHNOLOGIA INFORMATIEI

GESTIUNEA MEMORIEI

-proiect sisteme de operare-

Cooordonator stiintific
Prof. dr. ing. ŞTEFAN STĂNCESU

PARTEA I:	PANDRĂ I. Iulian	431A
PARTEA a II-a	PĂTRU P. Mircea	431A

2010

Cuprins

Prefaţă

PARTEA I: CONCEPTE FUNDAMENTALE - PANDRĂ I. IULIAN 431A

Contine o scurta introducere conceptele fundamentale precum: segmentul de cod, segmentul de date, spatial virtual de adrese la windows

PARTEA A II-A : APELURILE DE SISTEM PENTRU GESTIUNEA MEMORIEI. COMPARATIE WINDOWS LINUX - PĂTRU P. MIRCEA 431A

In sectiunea **Gestiunea Memoriei** vom pune accent pe importanta unei organizari cat mai bune a memoriei pentru o utilizare cat mai eficienta a sistemului de calcul. In continuare vom introduce conceptele de *multitasking* si *memorie virtuala* utile in intelegerea modului de lucru a sistemelor de operare cu procesele. Memoria virtuala urmareste cresterea resurselor de memorie disponibile pentru programele utilizator in timp ce multitaskingul se refera la executia in paralel a mai multor procese. Cu ajutorul memoriei virtuale se mapeaza spatii de adrese ale proceselor in resursele distribuite ale sistemului.

In cadrul **Managementul Memoriei la Linux** vom discuta despre*: stiva*, segmentul cel mai de sus in spatiul de adresa al procesului gestionat automat de compilator dupa principiul LIFO, *segmental de mapare a memoriei* pentru maparea continutului fisierelor direct in memorie, *heap-zona de memorie* dedicata alocarii dinamice a memoriei, *zona de cod* si *zona de date***.**

In **Probleme de lucru cu memoria** vom face referire la 2 situatii ce apar frecvent in lucrul cu memoria: *accesul invalid la memorie* si *leaku-rile de memorie***.** In Linux alocarea memoriei pentru procesele utilizator se realizeaza prin intermediul functiilor de biblioteca malloc, calloc si realloc iar dezalocarea ei prin intermediul functiei free.

In cadrul **Managementului Memoriei la Windows** vom lamuri conceptele de *director de pagina* si *tabela de pagina* specifice fiecarui process. Pentru **Protectia memoriei Windowsul** blocheaza atacuri ale anumitor procese asupra altora. Ca si la Linux **Alocarea memoriei in Windows** se refera la managementul spatiului virtual de adrese pentru care exista 3 functii speciale: *VirtualAlloc*, *VirtualAllocEx* si *VirtualFree.*

PARTEA I: Concepte fundamentale - PANDRĂ I. Iulian 431A

[Abstract: Concepte fundamentale utilizate segmentul de cod, segmentul de date, spatial virtual de adrese la windows.]

Notiuni introductive

1. Memoria constituie o resursa fizica fundamentala in orice sistem de calcul.
2. Memoria se caracterizeaza prin:

-capacitate;
-mod de acces la informatie;
-timp de acces;
-durata ciclului;
-viteza de transfer a informatiei, cost.

3. Se poate vorbi despre o divizare a echipamentelor, prevazute cu posibilitatea de a inregistra, pastra si reda informatia, in doua categorii:

-memoria interna (operativa, principala, centrala);
-memoria externa.

4. Memoria principala este adesea insuficienta pentru programele aflate in executie
5. Memoria interna este o componenta principala a calculatoarelor, rolul ei fiind de a pastra date si programe,atunci cand acestea urmeaza a fi folosite de catre un proces executat de catre CPU(von Neumann).
6. Memoria primara impreuna cu registrii CPU si memoria ‘cache ‘ formeaza memoria executabila, deoarece aceste componente sunt implicate in executia unui proces.
7. Unitatile de memorie externa (secundara)- utilizate pentru stocarea datelor pe o durata mai lunga de timp.
8. Fisierele executabile pentru a deveni procese trebuie sa fie incarcate in memoria primara.
9. Administratorul memoriei interne are ca obiective partajarea memoriei interne de catre mai multe procese si minimizarea timpului de acces.

Cerinte ale memoriei interne

1. Timpul de acces la memoria interna trebuie sa fie cat mai mic posibil ; acest lucru se realizeaza prin proiectarea adecvata atat a componentei hardware cat si a celei software implicate in gestiunea memoriei. Calculatoarele moderne folosesc memoria ‘cache’, ce reprezinta un tip de memorie cu acces foarte rapid si care contine informatiile cele mai recent utilizate de catre CPU.

2. Dimensiunea memoriei adresabile trebuie sa fie cat mai mare posibil, ceea ce se poate realiza prin conceptul de memorie virtuala.

3. Memoria interna trebuie sa aiba un cost relativ scazut.

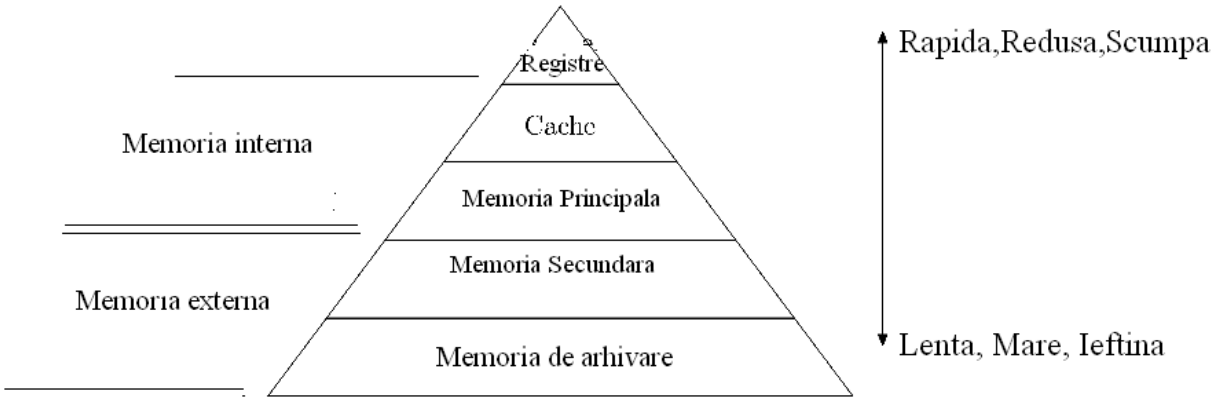
Obiect de activitate al gestiunii de memorie

- contabilizeaza zonele ocupate / libere;
- aloca memoria pentru procese;
- dealoca memoria de care procesele nu au nevoie;
- face transferul proceselor intre memoria principala si hard disk;
- gestiunea memoriei secundare;
- politici de schimb intre procese, memoria operativa si memoria secundara;
- calculul de translatare a adresei(realocare);
- protectia memoriei;

Structura memoriei

In structura actuala, memoria unui sistem de calcul apare ca in figura:

Organizarea ierarhica a memoriei



Memoria cache contine informatiile cele mai recent utilizate de catre UCP. Are o capacitate mica,dar este foarte rapida. La fiecare acces, UCP verifica daca pachetul de date invocat se afla in memoria cache.Daca da,atunci are loc schimbul intre UCP si el.Daca nu, atunci este cautat in nivelele superioare.Pachetul respectiv este adus din nivelul in care se afla inmemoria cache,dar odata cu ea se aduce un numar de locatii vecine ei astfel incat impreuna sa umple memoria cache.

Memoria operativa(memoria principala) contine instructiunile si datele pentru toate procesele existente in sistem.In momentul in care un proces este terminat si distrus, spatiul de memorie operativa pe care l-a ocupat este eliberat si va fi ocupat de alte procese.Viteza de acces este mare.

Memoria secundara apare la sisteme de operare care detin mecanisme de memorie virtuala.Aceasta memorie este privita ca o extensie a memoriei operative. Suportul ei principal este discul magnetic.Accesul la memoria aceasta se face mult mai lent decat la memoria operativa.

Memoria de arhivare este gestionata de utilizator si consta din fisiere,baze de date rezidente pe diferite suporturi magnetice(discuri, benzi, etc).

Memoria cache si memoria operativa formeaza **memoria interna**. Accesul UCP-ului la acestea se face in mod direct. Pentru ca UCP sa aiba acces la datele din memoria secundara si de arhivare, acestea trebuie mai intai mutate in memoria interna.

SISTEME DE GESTIUNE ALE MEMORIEI:

-cele care folosesc mecanismele de swapping si memorie virtuala.
-cele care nu le folosesc pe acestea.

Segmentarea memoriei

Un segment este o zona continua de memorie, utilizat pentru a retine instructiuni sau date. Adresa unui octet dintr-un segment se compune din doua parti independente:

- adresa de inceput(de baza) a segmentului;
- deplasamentul octetului in interiorul segmentului;

In concluzie, orice adresa poate fi scrisa sub forma unei perechi de forma:

(adresa_baza_segment,deplasament)

Fiecarui segment existent in memorie i se asociaza o structura de date numita descriptor de segment.Informatiile continute de acesta sunt urmatoarele:

- adresa de inceput a segmentului;
- dimensiunea segmentului;
- drepturi de acces la segment;

In momentul in care un proces incearca sa acceseze o adresa de memorie,au loc urmatoarele actiuni:

- se verifica in descriptorul segmentului drepturile de acces, pentru a se decide daca procesul are dreptul de a accesa adresa dorita

- se verifica daca deplasamentul nu depaseste dimensiunea segmentului

- daca se produce o eroare la unul din pasii anteriori, se genereaza o intrerupere, a carei rutina de tratare va anunta procesul asupra erorii sau il va termina in mod fortat

- daca nu s-a produc nici-o eroare, se calculeaza adresa fizica si se realizeaza accesul prupriu-zis

Este necesar sa existe un algoritm care sa decida care din zonele libere este cea mai potrivita pentru a fi ocupata de noul segment.

Algoritmi in acest scop:

-FIRST FIT: se parcurge memoria in ordine crescatoare a adreselor si se plaseaza segmentul in prima zona libera suficient de mare.

A20	20	108					
A15	20	15	93				
A10	20	15	10	83			
A25	20	15	10	25	58		
D20	20	15	10	25	58		
D10	20	15	10	25	58		
A8	8	12	15	10	25	58	
A30	8	12	15	10	25	30	28
D15	8	37			25	30	28
A15	8	15	22		25	30	28

-NEXT FIT: se parcurge memoria in ordine crescatoare a adreselor si se plaseaza segmentul in a II-a zona libera suficient de mare

A8	20	15	10	25	8	50		
A30	20	15	10	25	8	30	20	
D15	45			25	8	30	20	
A15	45			25	8	30	15	5

-BEST FIT: se parcurge intreaga memorie si se plaseaza segmentul in cea mai mica zona libera suficient de mare

A8	20	15	8	2	25	58		
A30	20	15	8	2	25	30	28	
D15	35		8	2	25	30	28	
A15	35		8	2	25	30	15	13

-WORST FIT: se parcurge intreaga memorie si se plaseaza segmentul in cea mai mare zona libera existenta

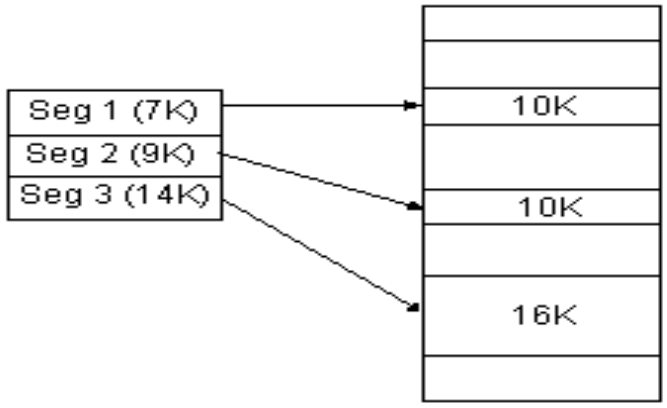
A8	20	15	10	25	8	50	
A30	20	15	10	25	8	30	20
D15	45			25	8	30	20
A15	15	30		25	8	30	20

Alocarea segmentata

Organizarea la nivel de segment imparte spatiul de memorie in blocuri de dimensiune variabila, numite *segmente*. Fiecare segment este o entitate logică de care programatorul este constient si poate contine o procedura sau un tablou sau o stiva sau o colectie de variabile scalare, dar deobicei nu contine combinatii ale acestora. Astfel, un program poate fi plasat in zone de memorie distincte. Adresa virtuala consta din numarul segmentului, adresa in cadrul segmentului si drepturi de acces (pentru protectie), formand descriptorul de segment. Fiecare proces are o tabela cu descriptorii de segment asociati. Calculul de adresa se face analog celui de la alocarea paginate.

Avantaje :

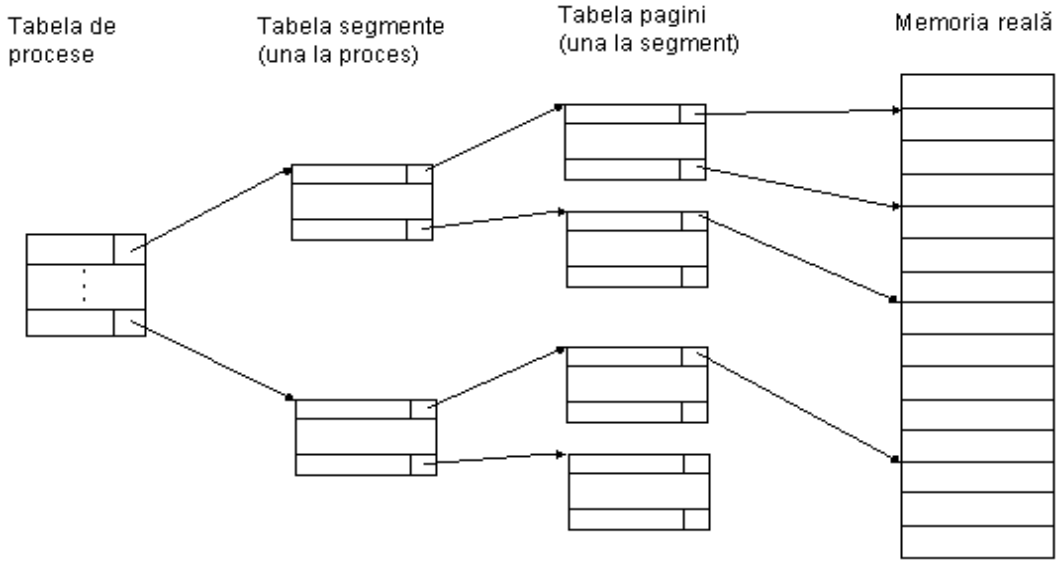
- daca procedura din segmentul n este modificata, recompilarea ei nu va afecta celelalte proceduri, pentru că adresa de inceput nu a fost modificata.
- segmentarea faciliteaza partajarea procedurilor sau datelor intre procese. Pentru aceasta, este suficient ca toate procesele sa aiba in tabelele lor aceeasi adresa pentru segmentul partajat.
- fiecare segment are codul propriu de protectie. Astfel, un segment de procedura (cod) poate fi specificat doar ca executabil, o structura de date ca read/write. Aceasta ajuta la gasirea unor erori de programare.
- legare si incarcare dinamica : o procedura se va incarca in memorie doar daca si numai atunci cand este apelata.



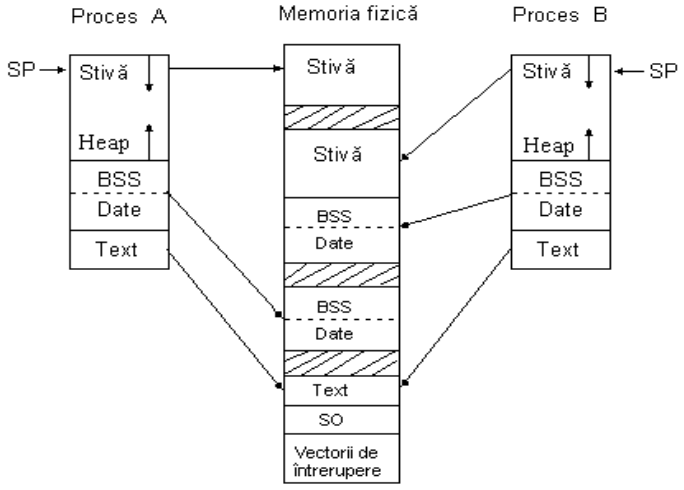
Dezavantaje :

Este posibil sa apara fenomenul de fragmentare. Compactarea presupune strategii mai complexe. Un alt dezavantaj este acela ca un obiect logic (de ex., o matrice) poate fi spart in mai multe segmente (trebuie folosite mai multe adrese de baza de segment pentru translatare).

Pentru evitarea fragmentarii, se foloseste **alocarea paginata şi segmentata**, in sensul că alocarea spatiului pentru fiecare segment sa se faca paginat.



UNIX foloseste alocarea segmentata (ofera paginare la cerere). Fiecare proces are un spatiu de adrese format din trei segmente : text (cod), date si stiva.



Segmentul de text contine codul executabil al programului (instructiuni masina). Este de tip read-only. Este partajat de catre toate procesele care executa acelasi cod (procesele fii).

Segmentul de text nu-si modifica dimensiunea in timpul executiei (nu creste, nu se micsoreaza).

Segmentul de date contine variabilele programului, siruri de caractere, tablouri si alte date. Este de tip read-write. Are doua parti : date initializate si date neinitializate (BSS). Datele initializate sunt datele ce primesc valori la compilare. Programul executabil contine datele initializate, textul (codul) si un header care ii spune SO-ului numarul de octeti ce trebuie alocata pentru datele neinitializate (astfel se evita pastrarea efectiva în programul executabil a spatiului necesar datelor neinitializate).

Segmentul de date isi modifica, in timpul executiei, nu numai continutul (modificarea valorilor variabilelor) ci si dimensiunea (alocare si dezalocare pentru variabile dinamice).

Segmentul de stiva incepe, de obicei, de la adresa cea mai mare a spatiului de adrese virtuale a procesului si creste spre adresele inferioare. Contine stiva nucleului in spatiul procesului, argumentele programului, pointer spre ambianta, stiva procesului si zona de heap.

Programele nu isi gestioneaza explicit segmentul de stiva. La lansarea in executie a unui program, stiva nu este goala. Ea contine toate variabilele de mediu precum si linia de comanda prin care s-a lansat in executie. Astfel, un program poate obtine argumentele apelului.

Deoarece fiecare segment reprezinta un spatiu de adrese separat, segmente diferite vor creste si vor scadea independent, fara sa se influienteze reciproc. Daca o stiva dintr-un anumit segment are nevoie de spatiu suplimentar pentru a creste, il poate avea, neexistand nimic peste care sa se suprapuna. Evident, un segment se poate umple, dar deoarece segmentele sunt de obicei mari, aceasta situatie este deosebit de rara. Pentru a preciza o adresa in aceasta memorie segmentata sau bidimensionala, programul trebuie sa furnizeze o adresa cu doua componente: un numar de segment si o adresa in cadrul segmentului.

Subliniem ca un segment este o entitate logica, de care programatorul este constient si pe care il foloseste ca pe o unica entitate logica. Un segment poate contine o procedura, un vector, o stiva sau o colectie de variabile scalare, dar in general nu contine un amestec de tipuri diferite. O memorie segmentata are si alte avantaje in afara de simplificarea prelucrarii structurilor de date care se extind sau se restrang. Segmentarea mai permite si partajarea de date sau de cod intre mai multe programe. Un exemplu comun il reprezinta biblioteca partajata. Statiile de lucru moderne ruleaza sisteme de ferestre avansate care au deobicei biblioteci compilate extrem de voluminoase aproape in fiecare program. Intr-un sistem segmentat, biblioteca grafica poate fi salvata intr-un segment si partajata de mai multe procese, eliminand nevoia ca fiecare proces sa aiba in spatiul sau de adrese.

Pe scurt, putem spune ca cele mai simple sisteme nu transfera pagini pe disc sau nu pagineaza deloc. Odata incarcat in memorie, un program ramane acolo pana termina. Anumite sisteme de operare permit existenta in memorie a unui singur proces la un moment dat, in timp ce altele suporta multiprogramarea.Urmatorul pas este transferul paginilor catre disc , in cazul acesta sistemul poate trata mai multe procese decat numarul pentru care are loc in memorie.Procesele

pentru care nu mai este loc sunt transferate pe disc.Sistemele de paginare pot fi modelate prin abstractizarea sirului de referinte al paginilor din program si utilizarea aceluiasi sir de referinte cu algoritmi diferiti. Aceste modele pot fi folosite pentru a face anumite previziuni legate de comportamentul paginarii.

Segmentarea ajuta la tratarea structurilor de date care isi modifica dimensiunea in timpul executiei si simplifica editarea de legaturi si partajarea. Faciliteaza de asemenea asigurarea unor gade de protectie diferite pentru segmente diferite. Segmentarea si paginarea sunt uneori combinate pentru a asigura o memorie virtuala bidimensionala.

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 sa fie in mod explicit definita de catre programator sau sistem. Translatarea unei adrese virtuale segmentate intr-o adresa fizica este mult mai complexa decat translatarea unei adrese virtuale paginata. Segmentarea se sprijina şi pe sistemul de fisiere, deoarece segmentele sunt stocate pe memoriile externe sub forma de fisiere. Segmentarea cu paginare valorifica avantajele oferite de cele două metode, fiind utilizata de sistemele de calcul moderne.

Referinte:

[1] Sisteme de operare moderne' Editia a doua, Andrew S.Tanenbaum
[2] http://en.wikipedia.org/
[3] Curs Sisteme de operare

PARTEA A II-A : Apelurile de sistem pentru gestiunea memoriei. Comparatie Windows Linux - PĂTRU P. Mircea 431A

Managementul memoriei este o parte importanta a unui sistem de operare; este crucial atat pentru partea de programare cat si pentru partea de administrare de sistem. In exemplele de mai jos ne vom referi doar la sisteme de operare Linux si Windows pe 32 de biti x86.

Gestiunea memoriei

Arhitecturile actuale de calculator ierahizeaza spatiul de memorie incepand de la cele mai rapide registre (registrele procesorului), memoria cache a CPU, memoria RAM, unitatile de memorie pe suport magnetic si terminand cu memoria de arhivare, o memorie nevolatila de dimensiuni mari, dar si cu timpi de acces foarte mari.

Un sistem de management al memoriei al unui sistem de operare decide ce tip de memorie sa utilizeze in functie de gradul de incarcare, ce memorie urmeaza sa fie alocata sau dezalocata, viteza de lucru necesara in cadrul unei aplicatii si modul in care se face transferul de date.

Sistemul de gestiune a memoriei din cadrul unui sistem de operare este folosit de toate celelalte sisteme: scheduling, I/O, filesystem, gestiunea proceselor, networking. Memoria este o resursa importanta a sistemului si sunt necesari algoritmi eficienti de utilizare si gestiune a acesteia.

Neintelegerea interfetelor si a actiunilor ce se petrec in spatele sistemelor de operare conduc la o serie deprobleme foarte des intalnite in aplicatiile software: memory leak-uri, accese invalide, suprascrieri, buffer overflow, corupere de zone de memorie. Este, in consecinta, fundamentala cunoasterea contextului in care actioneaza sistemul de gestiune a memoriei si inelegerea interfetei pusa la dispozitie de sistemul de operare programatorului.

Referinte:
[1] http: //operatingsystem. wordpress. com/2007/05/30/memory-management/
[2] http: //marconi. unitbv. ro/
[3] Radu Radescu - Arhitectura Sistemelor de calcul, Editura Politehnica Press, Bucuresti, 2009

Limitari ale componentelor fizice

Cele mai multe limitari ale unor procese sunt impuse de componenta hard a sistemului si nu de sistemul de operare. Orice calculator are un procesor si o memorie RAM, cunoscuta si sub numele de memorie fizica.

Un procesor interpreteaza un sir de date ca niste instructiuni pe care urmeaza sa le execute; are una sau mai multe unitati de procesare care realizeaza operatii cu numere intregi sau operatii in virgula mobila si alte operatii logice si aritmetice complexe. Registrele interne-direct adresabile – ale unui procesor reprezinta o memorie de foarte mare viteza care este folosita la stocarea operanzilor si a rezultatelor operatiilor. Dimensiunea registrului determina numarul cel mai mare cu care poate lucra un procesor intr-un singur ciclu de ceas.

Procesorul este conectat la memoria fizica prin magistrala de memorie. Dimensiunea adresei fizice (adresa utilizata de procesor pentru a indexa memoria RAM) limiteaza spatiul de memorie care poate fi adresat. De exemplu, o adresa fizica de 16 biti poate sa adreseze de la 0x0000 pana la 0xFFFF, ceea ce inseamna 2^16=65536 locatii unice de memorie. Daca fiecare adresa indica un byte de date, o adresa fizica de 16biti ar permite procesorului sa adreseze 64 KB de memorie.

Numarul de biti din descrierera unui procesor se refera, in general, la dimensiunea registrelor, dar mai exista si exceptii precum arhitectura 390 31-bit -in care se refera la dimensiunea fizica a adreselor. Pentru calculatoare personale, servere acest numar este 31, 32 sau 64; pentru dispozitive embedded si microprocesoare, dimensiunea registrelor ajunge si pana la 4. Dimensiunea fizica a adreseri poate sa fie ca cea a registrului, dar poate sa fie si mai mare sau mai mica. Cele mai multe procesoare pe 64 -biti pot sa ruleze programe de 32 de biti atunci cand pe sistemul de calcul este instalat un sistem de operare adecvat.

Tabel 1. Lista cu cele mai importante arhitecturi pe care ruleaza sistemele Windows si Linux cu dimensiunea registrelor si dimensiunea fizica a adreselor

Table 1. Register and physical address size of some popular processor architectures

Architecture	Register width (bits)	Physical address size (bits)
(Modern) Intel® x86	32	32 36 with Physical Address Extension (Pentium Pro and above)
x86 64	64	Currently 48-bit (scope to increase later)
PPC64	64	50-bit at POWER 5
390 31-bit	32	31
390 64-bit	64	64

Referinte:

[1] Andrew Hall, Software Engineer, IBM - Thanks for the memory

Sistemele de operare si memoria virtuala

Pentru scrierea aplicatiilor care ruleaza direct pe un procesor, fara un sistem de operare, se poate utiliza intreg spatiul fizic de memorie pe care procesorul poate sa-l adreseze(presupunem ca exista suficient spatiu fizic RAM). Pentru a utiliza facilitati precum multitaskingul sau alte abstractizari ale hardwerului, este obligatoriu sa se utilizeze un sistem de operare pentru a rula programele.

Fiecare aplicatie sau program care ruleaza pe sistemul de operare este un proces alcatuit din unul sau mai multe fire de executie (thread-uri). Un thread este un set de instructiuni sau o parte anume a aplicatiei, care se executa independent in cadrul programului sau sistemului. Threadurile sunt responsabile cu multitasking-ul.

Multitaskingul presupune executia in paralel a mai multor procese, pentru a imbunatati eficienta sistemului.

In multitasking, in SO precum Windows si Linux mai multe programe partajeaza resursele de sistem, inclusiv memoria. Pentru fiecare program trebuie sa se aloce regiuni din spatiul fizic de memorie pentru a le rula. Se pot realiza SO in care pentru fiecare program se aloca spatiu de memorie fizic si in care sa se limiteze accesul lor decat in spatiul de memorie dedicat. Unele SO proiectate pentru dispozitive embedded lucreaza astfel, dar nu este practic ca intr-un mediu constituit din mai multe programe care nu sunt testate impreuna deoarece orice program poate sa corupa memoria unui alt program sau chiar SO insusi.

Memoria virtuala este un concept arhitectural prin care memoria operativa este extinsa peste spatiul de adresare al memoriei externe (hard disk, banda magnetica). Prin tehnicile de implementare, numite paginare si segmentare, se urmaresc doua aspecte esentiale:

- cresterea resurselor de memoriei disponibile pentru programele utilizator
- protejarea zonelor de memorie alocate modulelor de program

Transferul de informatii intre diferitele nivele de memorie se realizeaza in mod automat, fie de catre componente hardware specializate (unitatea de gestiune a memoriei, MMU), fie de catre sistemul de operare.

Memoria virtuala este utila in cazul sistemelor de operare multitasking si multiuser (Windows, Linux)

Memoria virtuala permite mai multor procese sa imparta memoria fizica fara sa suprapuna datele unele peste altele.

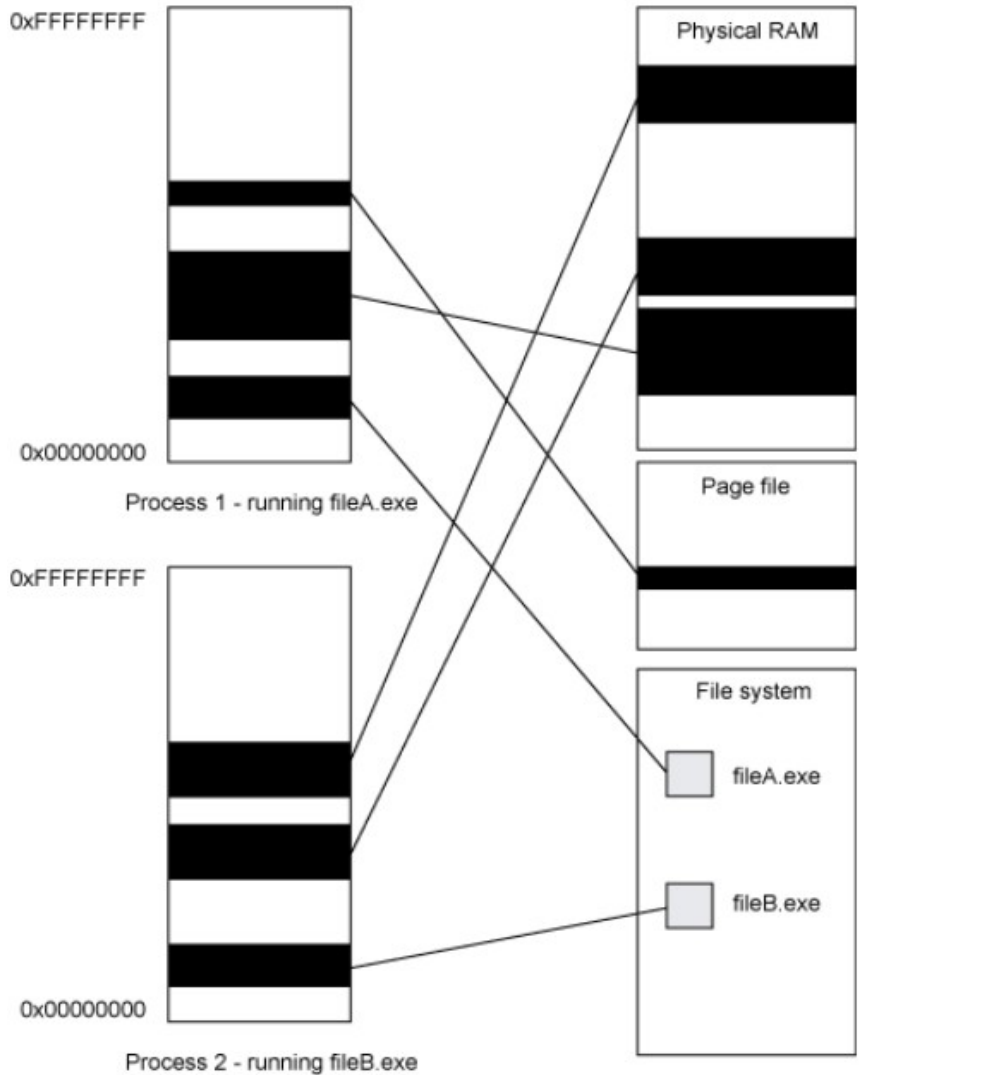
Intr-un sistem de operare cu memorie virtuala (Windows si Linux), fiecare program dispune de propriul spatiu virtual de adrese -un spatiu logic de adrese a caror dimensiune este dictata de dimensiunea adresei din system (31, 32 sau 64 de biti pentru unitati desktop si servere).

Regiunile spatiului virtual de adrese a unui proces poate fi **mapat** (translatat) la o memorie fizica, la un fisier sau catre orice alt spatiu de stocare adresabil.

Sistemul de operare poate sa mute date din memoria fizica de la /catre o **zona de swap** (pagina in cazul Windows sau o partitie swap in cazul Linux) cand nu sunt folosite, pentru a utiliza eficient spatiul fizic de memorie. Cand un program incearca sa acceseze memoria utilizand o adresa virtuala, SO alaturi de partea hardware mapeaza adresele virtuale spre o locatie fizica. Locatia fizica poate sa fie o zona din memoria RAM, un fisier, sau o pagina/partitie swap. Daca o anumita regiune din memorie a fost mutata catre o zona swap, atunci ea este incarcata din nou in memoria fizica inainte de a fi utilizata.

In figura de mai jos este ilustrat modul in care MV lucreaza (memoria virtuala) prin maparea spatiilor de adrese ale proceselor in resursele distribuite ale sistemului.

Figure 1. Virtual memory mapping process address spaces to physical resources



Dimensiunea spatiului virtual de adrese poate sa fie mai mica decat dimesiunea adresei fizice a procesorului. Intel x86 32-bit a avut initial o adresa fizica de 32 de biti care i-a permis procesorului sa adresesze o zona de 4GB. Mai tarziu, a fost conceputa o extensie numita PAE(*Physical Address Extension*) ce a marit dimensiunea adresei fizice pana la 36 de biti -ceea ce a permis sa se instaleze si sa se adreseze un spatiu de pana la 64GB de RAM. PAE a permis sistemelor de operare sa mapeze 4GB de spatiu virtual de adrese de 32 biti pe un interval mare de adrese fizice, dar nu permite fiecarui proces sa aibe un spatiu virtual de adrese de 64GB.

Astfel daca se instaleaza mai mult de 4GB de memorie pe un server Intel pe 32-biti, nu se poate mapa toata direct intr-un singur proces.

Linux foloseste tehnologii bazate pe maparea regiunilor in spatiul virtual de adrese. Desi nu putem sa referentiem direct mai mult de 4 GB de memorie, se poate lucra cu regiuni largi de memorie

Referinte:
[1] Andrew Hall, Software Engineer, IBM - Thanks for the memory
[2] Radu Radescu -Arhitectura Sistemelor de calcul, Editura Politehnica Press, Bucuresti, 2009
[3] S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004
[4] http: //www. microsoft. com/whdc/system/kernel/wmm. msp

Spatiul virtual de adrese. Spatiul kernel si spatiul utilizator

O adresa virtuala identifica o locatie in memoria virtuala. Spatiul virtual de adresa este divizat in 2 zone: spatiul utilizator si spatiul kernel(sistem)

Spatiul utilizator este portiunea din spatiul virtual de adrese in care Windows-ul mapeaza procesele-utilizator, libariile-utilizator dinamice(DLL- *Dynamic-link library*). Fiecare proces are un context propriu descris de codul si datele pe care acesta le utilizeaza. Cat timp procesul ruleaza, o portiune din context, numita setul de lucru, este rezident in memorie.

Spatiul kernel sau **spatiul sistem**, este portiunea spatiului de adrese in care driverele sistemului de operare si ale kernelului sunt rezidente. Acest spatiu este accesibil numai codului kernel.

Kernelul este programul principal al SO si contine logica de interfata cu componentele hard ale sistemului, managementul programelor, si furnizarea de servicii cum ar fi cele de retea si memorie virtuala.

Ca o parte a secventei de boot, kernel-ul SO porneste si initializeaza hardwer-ul. Atunci cand kernelul termina de configurat componenta hard si propria stare interna, primul proces utilizator este pornit. Daca un program utilizator are nevoie de un serviciu de la SO, poate sa realizeze o operatie numita -apel de sistem-care face un salt in programul kernel, care mai apoi proceseaza cererea. Apelurile de sistem sunt, in genereal, necesare la operatii precum scriere si citire din fisiere, in aplicatii de retea si la pornirea unor procese noi.

Distinctia dintre spatiul-utilizator si spatiul-sistem este importanta in mentinerea securitatii sistemului. Un thread utilizator poate sa acceseze date numai in contextul propriilor procese. Nu poate sa acceseze date din contextele altor procese sau din spatiul de adrese ale sistemului. Aceste restrictii impiedica threadurile-utilizator sa citeasca si sa scrie date care apartin altor procese sau sistemului de operare si a drivere-lor, ce pot genera vulnerabilitati in securitatea sistemului sau chiar o cadere a sistemului (*system crash*).

Driverele kernel sunt autorizate de sistemul de operare si astfel pot accesa atat spatiul utilizator cat si spatiul kernel (sistem). Cand este apelata o rutina driver in contextul unui thread user; toate datele threadului ramin in spatiul de adrese utilizator. Driverul poate sa acceseze spatiul utilizator pentru un thread pe langa faptul ca poate sa acceseze spatiul sistem. Threadul-utilizator nu are in schimb acces la datele spatiului sistem ale driverului

Dimensiunile dintre spatiu kernel si *spatiu utilizator* variaza de la un SO la altul si chiar pentru instante ale aceluiasi SO care ruleaza pe arhitecturi hardware diferite. Aceste spatii sunt de obiecei configurabile astfel incat sa se ofere mai mult spatiu pentru aplicatiile utilizator sau kernellului in functie de cerinte.

Reducerea spatiului de kernel poate sa cauzeze probleme precum restrangerea numarului de utilizatori care pot sa se logheze simultan sau limitarea numarului de procese care por rula, un spatiu utilizator mai mic inseamna ca aplicatiile vor avea un spatiu mai mic in care sa lucreze.

Din constructie, SO Windows pe 32-biti au 2 GB spatiu utilizator si 2 GB spatiu kernel. Aceste dimensiuni pot fi schimbate astfel: se pot aloca 3GB spatiu utilizator si 1Gb spatiu kernel pe unele versiuni de Windows prin adaugarea /3GB switch pe configuratia de boot si prin relinkarea aplicatiilor cu /LARGEADDRESSAWARE switch.

Prin folosirea switch-ului /3GB pe Windows se reduce spatiul kernel la jumatate la cat era in mod original. Este posibil sa se ocupe spatiul de 1GB de kernel si sa apara o scadere a vitezei de lucru cu dispozitivele I/O(intrare /iesire) sau probleme in crearea unor sesiuni noi pentru alti utilizatori.

Pe SO Linux pe 32 de biti sunt alocati 3GB pentru spatiul utilizator si 1GB spatiu de kernel. Unele distributii Linux ofera un kernel "*hugemem*" care suporta 4GB pentru spatiu utilizator. Pentru a se obtine o astfel de modificare kernelului ii este data o adresa de spatiu a sa pentru a fi folosita cand este realizat un apel de sistem.

Cresterea spatiului utilizator determina in schimb apeluri de sistem mai lente deoarece SO trebuie sa copieze date intre diferite adrese de spatii si sa reseteze procesele de mapare a adreselor de spatii de fiecare data cand este realizat un apel de sistem.

In figurile de mai jos este figurat aranjamentul spatiilor de adrese pentru un SO Windows si Linux pe 32 biti.

Figure 2. Address-space layout for 32-bit Windows

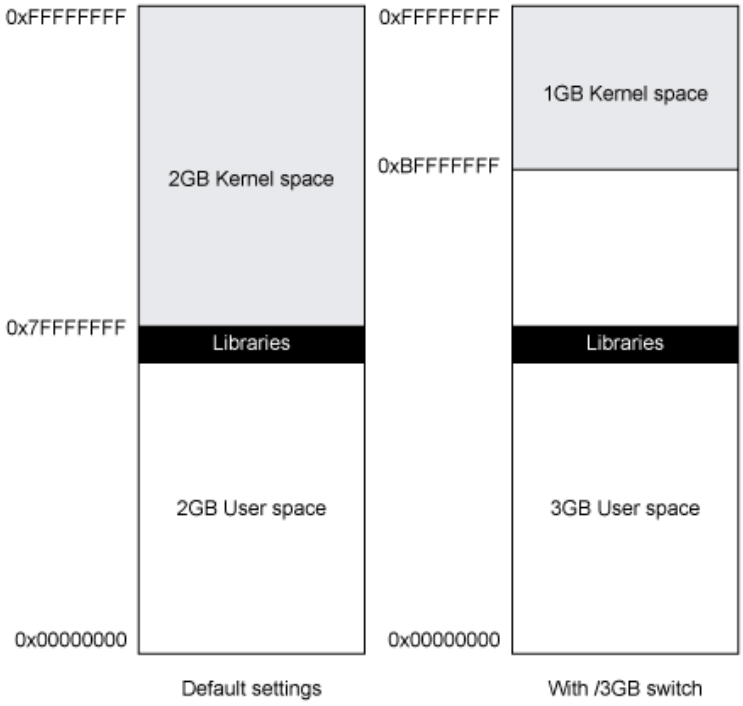
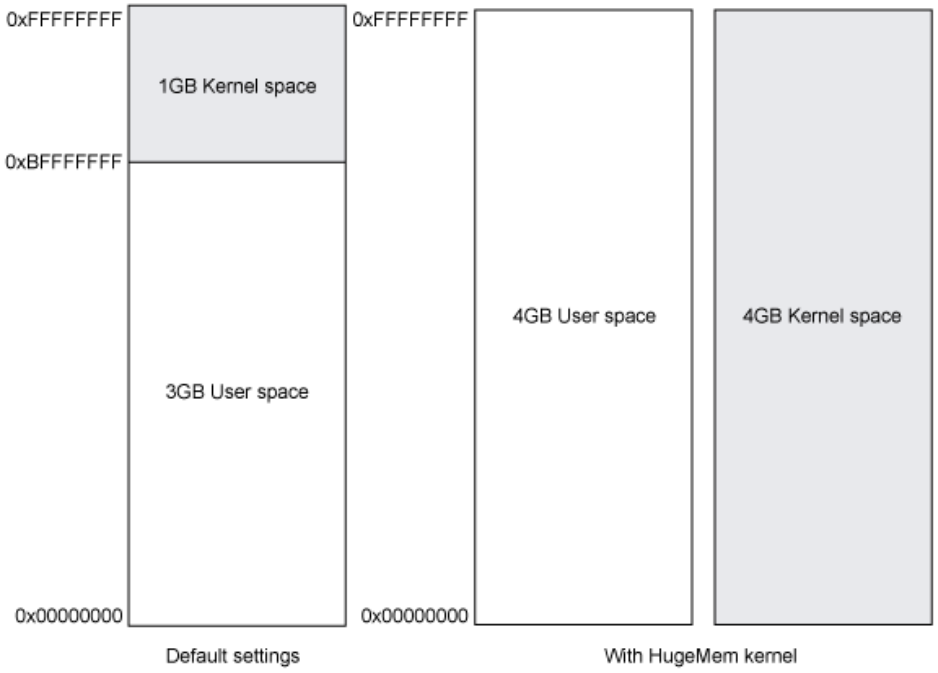


Figure 3. Address-space layout for 32-bit Linux



Spatiul de adrese al unui proces trebuie sa contina tot ce are nevoie un program-inclusiv programul insusi si librariile (DLL la Windows, fisiere. so la Linux) pe care le foloseste.

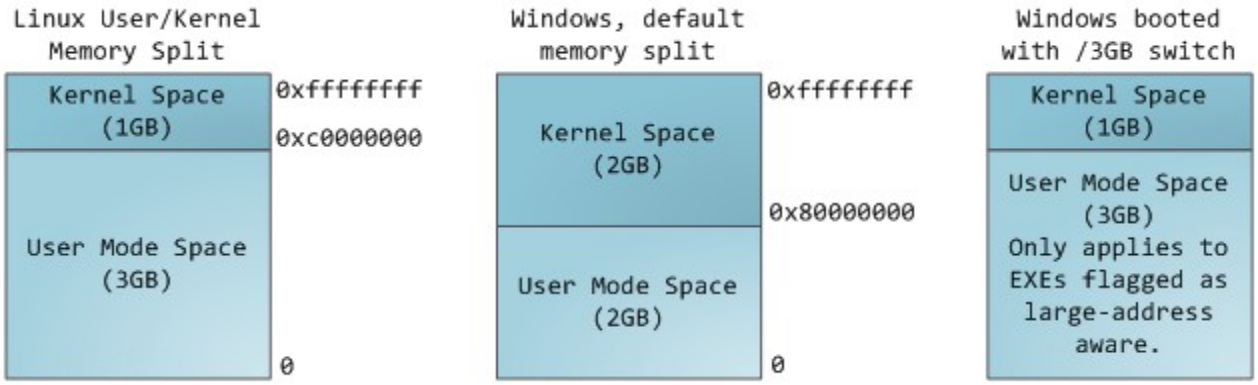
Librariile pe langa faptul ca pot sa consume spatiul pe care programul poate sa-l utilizeze pentru a stoca date pot sa fragmenteze si spatiul de adrese si sa reduca spatiul de memorie care poate fi alocat ca o portiune continua.

Acest lucru devine de neglijat la un program care ruleaza pe Windows x86 care are un spatiu utilizator de 3GB. DLL -urile sunt construite de la o adresa stabilita: cand un DLL este incarcat; este mapat in spatiul de adresa de la o locatie anume. iar daca aceasta este ocupata este mutat si incarcat in alta parte.

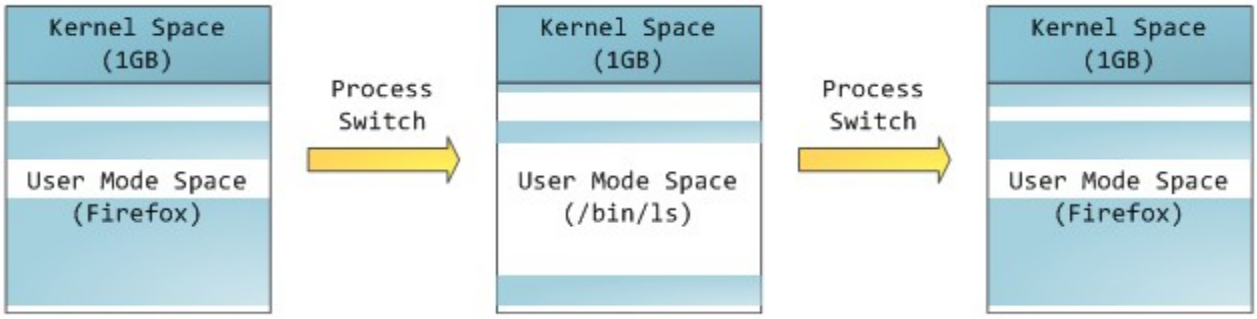
Cu un spatiu utilizator de 2GB disponibili pentru SO Windows NT, asa cum a fost proiectat initial, avea sens ca librariile sa fie incarcate la limita celor 2GB -pentru ai lasa utilizatorului un spatiu liber continuu cat mai mare.

Cand spatiul utilizator este extins la 3Gb; librariile sistemului inca se incarca in apropierea zonei de 2GB -acum in mijlocul spatiului utilizator. Desi dispunem de un spatiu utilizator de 3GB, este imposibil sa alocam un bloc de 3GB de memorie pentru ca librariile fragmenteaza aceasta zona.

Orice scapare in alocarea de memorie poate genera diferite probleme care difera de la o situtie la alta, daca s-a epuizat spatiul de adresa sau daca se atinge limita fizica a memoriei. Terminarea spatiului de adrese apare doar la procesoarele de 32 de biti -deoarece spatiul maxim de 4GB alocat este foarte usor de alocat. Un proces pe 64 de biti are un spatiu de adrese de cateva sute sau chiar mii de GB, care este foarte greu de incarcat.



La Linux, spatiul kernel este mereu prezent si mapeaza aceeasi zona de memorie la toate procesele. Codul si datele kernelului sunt mereu adresabile, si capabile sa gestioneze intreruperi sau apeluri de sistem in fiecare moment. Prin contrast, maparea pentru modul utilizator spatiul de adrese se modifica de fiecare data cand se produce un switch (comutare) de proces.



Zonele albastre semnifica adresele virtuale care sunt mapate in memoria fizica, iar zonele albe regiunile nemapate.

In exemplul de mai sus arata cum programul Firefox isi foloseste spatiul virtual de adrese. Benzile distincte in spatiul de adrese corespund segmentelor de memorie cum ar fi heap-ul, stiva samd. Aceste segmente reprezinta simple intervale de adrese de memorie.

Referinte:

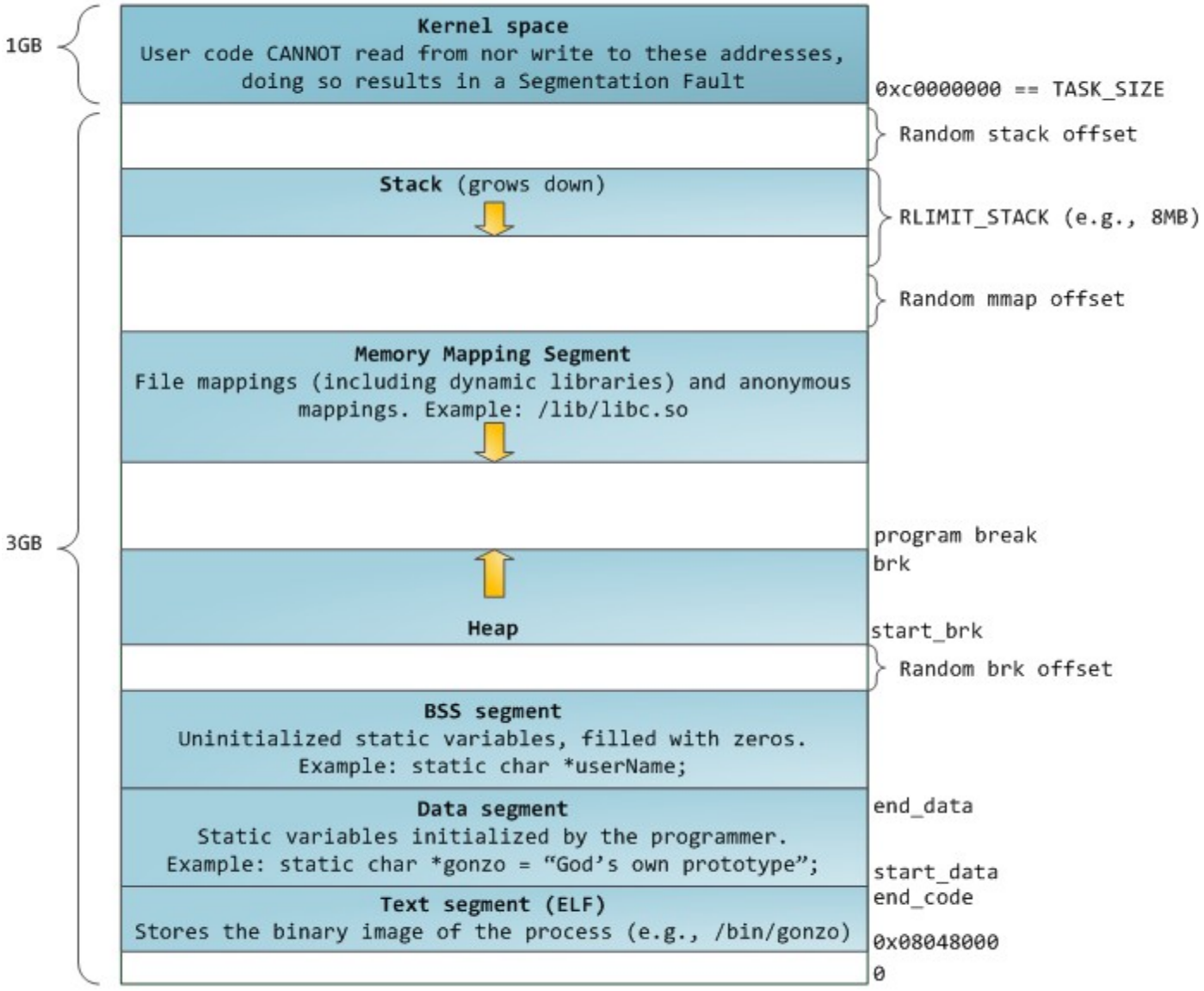
[1] Martin Trotter, Software Engineer, IBM UK Labs -Don't forget about memory
[2] Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004
[3] Gustavo Duarte, Anatomy of a Program in Memory

Mangementul Memoriei la Linux

Spatiul de adresa al unui proces la Linux

Spatiul de adrese al unui proces, sau, mai bine spus, spatiul virtual de adresa al unui proces reprezinta zona de memorie virtuala utilizabila de un proces. Fiecare proces are un spatiu de adresa propriu. Chiar in situatiile in care doua procese partajeaza o zona de memorie, spatiul virtual este distinct, dar se mapeaza peste aceeasi zona de memorie fizica

In figura de mai jos este ilustrat spatiul de adresa al unui proces in Linux



Cele 4 zone importante din spatiul de adresa al unui proces sunt zona de date, zona de cod, stiva si heap-ul.

Dupa cum se observa si din figura, stiva si heap-ul sunt zonele care pot creste. De fapt, aceste doua zone sunt dinamice si au sens doar in contextul unui proces. De partea cealalta, informatiile din zona de date si din zona de cod sunt descrise in executabil.

STIVA

Segmentul cel mai de sus in spatiul de adresa al procesului este stiva, care stocheaza variabile locale si parametrii ale functiilor din majoritatea limbajelor de programare. Stiva este gestionata automat de compilator. Apelarea unei metode sau a unei functii adauga in stiva un nou cadru -stiva. La fiecare revenire din functie stiva este golita. Acesta structura simpla, ordonata dupa principiul LIFO, sugereaza ca nu este nevoie de structuri complexe pentru a tine evidenta elementelor, un simplu pointer care indica catre varful stivei este suficient. Punerea si scoaterea elementelor din stiva este foarte rapida

Este posibil sa se termine zona mapata pentru stiva prin incarcarea peste limita cu date a acesteia. Aceasta situatie declanseaza un acces invalid de tip page fault care este tratata in Linux de *expand_stack(),* care la randul ei apeleaza *acct_stack_growth()* care verifica daca este adecvat sa greasca dimensiunea stivei. Daca dimensiunea stivei este sub RLIMIT_STACK (uzual 8MB), atunci, in mod normal, dimensiunea stivei creste si programul continua sa ruleze. Acesta este mecanismul normal in care dimensiunea stivei se ajusteaza dupa cerintele impuse de program. Cu toate acestea, daca dimensiunea maxima admisa a stivei este atinsa, apare situatia in care stiva este supraincarcata(*stack overflow*) iar progrmaul genereaza o eroare de tip Segmentation Fault. Cand zona mapata pentru stiva este extinsa pentru a satisface cerintele, aceasta nu se reduce cand stiva revine la dimensiuni mai mici.

Cresterea dinamica a stivei este singura situatie in care accesul la o zona nemapata de memorie, marcata cu alb in graficul de mai sus, este permisa. Orice alta cerinta de acces catre o zona nemapata de memorie determina o eroare de tip page fault care detemina o eroare de segmentare Segmentation Fault. Unele zone mapate de memorie sunt doar pentru citit(*read only*) si astfel incercari de accesare a acestor zone duc la segfaults.

Segmentul de mapare a memoriei

Sub stiva, avem segmentul de mapare al memoriei. Aici kernel-ul mapeaza continutul fisierelor direct in memorie. Orice aplicatie poate sa solicite o astfel de mapare prin intermediul apelului de sistem Linux *mmap()* sau *CreateFileMapping()/ MapViewOfFile()* in cazul sistemelor de operare Windows. Maparea de memorie este un mod performant si convenabil sa se creeze fisiere I/O, si astfel este folosit pentru incarcarea librariilor dinamice. Este posibil de asemenea sa se creeze mapari anonime de memorie care sa nu corespunda nici unui fisier, dar care sa fie folosit in schimb pentru date de program. In Linux, daca se solicita un bloc mare de memorie utilizand *malloc(),* libraria C va crea o mapare anonima in loc sa foloseasca memoria heap. Bloc mare desemneaza un bloc mai mare decat MMAP_THRESHOLD bytes, implicit 128 kB si ajustabil

prin *mallopt()*.

HEAP-ul

Heap-ul este zona de memorie dedicata alocarii dinamice a memoriei. Heap-ul este folosit pentru alocarea de regiuni de memorie a caror dimensiune se afla doar la runtime. La fel ca si stiva, heap-ul este o regiune dinamica si care ii modifica dimensiunea. Spre deosebire de stiva, insa, heap-ul nu este gestionat de compilator. Este de datoria programatorului sa stie cata memorie trebuie sa aloce si sa retina cat a alocat si cand trebuie sa dezaloce. Problemele frecvente in majoritatea programelor apar la pierderea referintelor la zonele alocate (memory leaks) sau referirea de zone nealocate sau insuficient alocate (accese invalide).

La limbaje precum Java, Lisp, etc. unde nu exista "*pointer freedom*", eliberarea spatiului alocat se face automat prin intermediul unui garbage collector (link). Pe aceste sisteme se previne problema pierderii referintelor, dar inca ramane activa problema referirii zonelor nealocate.

Daca exista spatiu suficient in heap care sa satisfaca cerintele de memorie, atunci alocarea este realizata de limbajul de programare fara interventia kernelului. Altfel zona de heap este largita prin apelul de sistem (implementarea) brk() pentru a face loc blocului solicitat. Managementul zonei de heap este complex, si necesita algoritmi sofisticati care sa utilizeze memoria eficient si sa ruleze si rapid in fata alocarii, uzual, haotice a programelor definite de utilizator. Timpul necesar pentru a satisface o cerinta de heap poate sa fie substantial.

In sfarsit, in partea de jos, se afla segmentele de memorie pentru: BSS, date si zona de cod(*text segment*)

ZONA DE COD

Zona/segmentul de cod (denumit si *'text segment'*) reprezinta instructiunile programului. Registrul de tip *'instruction pointer'* va referi adrese din zona de cod. Se citeste instructiunea indicata, se decodifica si se interpreteaza, dupa care se incrementeaza contorul programului si se trece la urmatoarea instructiune. Zona de cod este, de obicei, o zona read-only.

Zona BSS

Zona BSS contine variabilele globale neinitializate sau initializate la zero ale unui program. De exemplu:

```
static int a;
char b;
```

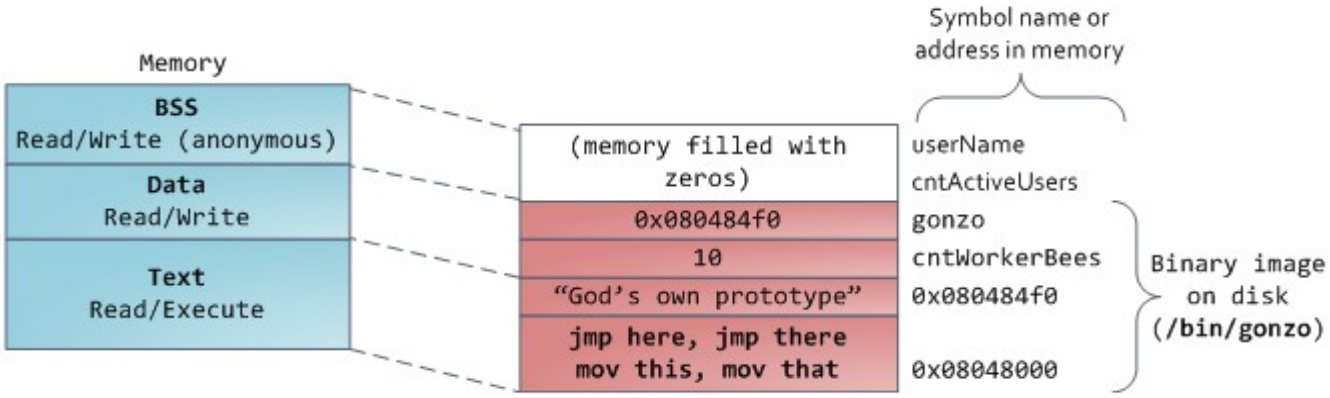
In general acestea nu vor fi prealocate in executabil ci in momentul crearii procesului. Alocare zonei BSS se face peste pagini fizice zero (*zeroed frames*).

ZONA DE DATE

Zona de date contine variabilele globale initializate la valori nenule ale unui program. De exemplu:

```
static int a = 3;
char b = 'a';
```

Un exemplu pentru cele 3 segmente de memorie



Referinte:
[1] Gustavo Duarte, Anatomy of a Program in Memory
[2] Andrew Hall, Software Engineer, IBM - Thanks for the memory
[3] http: //marconi. unitbv. ro/ti

Probleme de lucru cu memoria

Lucrul cu heap-ul este una dintre cauzele principale ale aparitiilor problemelor de programare. Lucrul cu pointerii, necesitatea folosirii unor apeluri de sistem/biblioteca pentru alocare/dezalocare, pot conduce la o serie de probleme care afecteaza functionarea unui program.

Problemele cele mai des intalnite in lucrul cu memoria sunt:

- accesul invalid la memorie
- leak-urile de memorie

Accesul invalid la memorie prespune accesarea unor zone care nu au fost alocate sau au fost eliberate.

Leak-urile de memorie sunt situaiile in care se pierde referinta la o zona alocata anterior. Acea zona va ramane ocupata pana la incheierea procesului. Ambele probleme si utilitarele care pot fi folosite pentru combaterea acestora vor fi prezentate in continuare.

De obicei, accesarea unei zone de memorie invalide rezulta intr-o eroare de pagina (*page fault*) si terminarea procesului (in Unix inseamna trimiterea semnalului SIGSEGV - afisarea mesajului *'Segmentation fault'*). Totusi, daca eroarea apare la o adresa invalida dar intr-o pagina valida, hardware-ul si sistemul de operare nu vor putea sesiza actiunea ca fiind invalida. Acest lucru se datoreaza faptului ca alocarea memoriei se face la nivel de pagina. Pot exista situatii in care sa fie folosita doar jumatate din pagina. Desi cealalta jumatate contine adrese invalide, sistemul de operare nu va putea detecta accesele invalide la acea zona.

Un tip special de acces invalid este **buffer overflow**. Acest tip de atac presupune referirea unor regiuni valide din spatiul de adresa al unui proces prin intermediul unei variabile care nu ar trebui sa poata referential aceste adrese. De obicei, un atac de tip buffer overflow rezulta in rularea de cod nesigur. Protectia la accese de tip buffer overflow se realizeaza prin verificarea limitelor unui buffer/vector fie la compilare, fie la rulare.

Un leak de memorie apare in doua situatii:

- un program omite sa elibereze o zona de memorie
- un program pierde referinta la o zona de memorie dezalocata si, drept consecina, nu o poate elibera

Memory leak-urile au ca efect reducerea cantitatii de memorie existenta in sistem. Se

poate ajunge, in situatiile extreme, la consumarea intregii memorii a sistemului si la imposibilitatea de functionare a diverselor aplicatii ale acestuia.

Mtrace - Un utilitar care poate fi folosit la depanarea erorilor de lucru cu memoria este mtrace. Acest utilitar ajuta la identificarea leak-urilor de memorie ale unui program.

Alocarea memoriei Linux

In Linux alocarea memoriei pentru procesele utilizator se realizeaza prin intermediul functiilor de biblioteca **malloc, calloc** si **realloc** iar dezalocarea ei prin intermediul functiei **free**. Aceste functii reprezinta apeluri de biblioteca si rezolva cererile de alocare si dezalocare de memorie pe cat posibil in user space. Asadar, se tin niste tabele care specifica zonele de memorie alocate in heap. Daca exista zone libere pe heap, un apel malloc care cere o zona de memorie care poate fi incadrata intr-o zona libera din heap va fi satisfacut imediat marcand in tabel zona respectiva ca fiind alocata si intorcand programului apelant un pointer spre ea. Daca in schimb se cere o zona care nu incape in nicio zona libera din heap, malloc va incerca extinderea heap-ului prin apelul de sistem brk sau mmap.

```
void *calloc(size_t nmemb, size_t size);
void *malloc(size_t size);
void *realloc(void *ptr, size_t size);
void free(void *ptr);
```

Memoria alocata se elibereaza cu free().

Memoria alocata de proces este eliberata automat la terminarea procesului insa in cazul unui proces server, de exemplu, care ruleaza foarte mult timp si nu elibereaza memoria alocata acesta va ajunge sa ocupe toata memoria disponibila in sistem cauzand astfel consecine nefaste.

Nu se elibereaza de doua ori aceeasi zona de memorie intrucat acest lucru va avea drept urmare coruperea tabelelor tinute de malloc ceea ce va duce din nou la consecinte nefaste. Intrucat functia free se intoarce imediat daca primeste ca parametru un pointer NULL, este recomandat ca dupa un apel free, pointer-ul sa fie resetat la NULL.

Referinte:

[1] http: //marconi. unitbv. ro/ti

Managementul Memoriei la Windows

Managementul memoriei in sistemele de operare Microsoft Windows a evoluat pana la o arhitectura foarte bine dezvoltata, capabila sa scaleze de la platformele embedded (in care Windows se executa din ROM) pana la arhitecturile cu configuratie NUMA (Non-Uniform Memory Access), reusind astfel sa utilizeze eficient hardwer-ul.

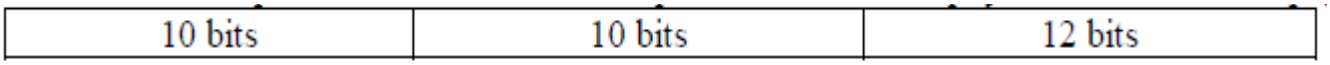
Managementul tabelei de pagina

In sistemele de operare Windows care lucreaza pe 32 de biti fiecare proces are propriul director de pagina("*Page Directory*") si propriile Tabele de pagina. Astfel sistemul de operare aloca 4Mb din spatiul sau pentru fiecare proces. Cand un proces este creat, fiecare element din directorul de pagina contine adresa fizica a unui tabel de pagina.

Elementele din tabela de pagina sunt fie valide fie invalide. Elementele valide contin adrese fizice ale unei pagini de 4KB alocata procesului. Elementele invalide contin niste biti speciali care marcheaza faptul ca sunt invalide cunoscti sub numele de *Invalid PTEs*. Cat timp memoria se aloca procesului, elementele din tabela de pagina sunt umplute cu adrese fizice ale paginilor alocate.

Procesul nu cunoaste nimic despre memoria fizica si foloseste doar adrese logice. Detaliile legate de corespondenta dintre adresele logice si adresele fizice sunt rezolvate transparent de Windows manager si de procesor.

Translatarea de la o adresa logica catre o adresa fizica este realizata de procesor. O adresa logica de 32 de biti este divizata in 3 parti, ca in figura de mai jos.



Procesorul incarca adresa fizica a paginii directorului stocata in CR3. Apoi utilizeaza cei mai semnificativi 10 biti din adresa logica ca index in directorul de paginaAcesti biti selecteaza un director de pagina pentru procesor (*PDE- Page Directory Entry*) care pointeaza catre o tabela de pagina. Ceilalti 10 biti sunt utilizati pentru a indica nu element din tabela de pagina. Folosirea acestor 10 biti, determina o tabela de pagina (*PTE -Page Table Entry*) care pointeaza catre o pagina fizica de 4KB. Cei mai putini semnificativi 12 biti sunt utilizati pentru a adresa bytes (B) individuali dintr-o pagina.

Protectia memoriei

Windows-ul protejeaza memoria tuturor proceselor astfe incat un proces nu poate sa acceseze memoria altui proces. Pentru a asigura aceata protectie Windowsul impune urmatoarele restrictii:

- pune doar adresa fizica a memoriei alocate in PTE pentru un proces. Acest lucru asigura faptul ca procesul primeste o eroare de acces daca incearca sa acceseze o adresa care nu ii este alocata
- un proces poate sa incerce sa si modifice tabela de pagina ca sa acceseze adresa fizica alocata altui proces. Windows-ul blocheaza astfel de atacuri prin stocarea tabelei de pagina in spatiul de adresa al kernelului

Alocarea memoriei in Windows

Primul si cel mai important alocator se ocupa de managementul spatiului virtual de adrese. Functiile care permit interactiunea cu acest alocator au nume care incep cu **Virtual** (*VirtualAlloc, VirtualAllocEx, VirtualFree* etc). Cu ajutorul acestui alocator pot fi efectuate diferite operatii care opereaza la nivel de pagina.

Un prim exemplu de operatie este rezervarea unui interval din spatiul virtual de adrese al procesului. O rezervare este exact ce ii spune numele. Ea nu consuma pagini fizice din memoria sistemului, dar previne utilizarea blocului respectiv de adrese pentru alte operatii. Evident, o rezervare nu influenteaza cu nimic alocarile din alte procese. O rezervare permite unui proces sa puna deoparte un interval de adrese pentru o structura de date care initial este mica si poate creste dupa necesitati. Rezervarea unui interval se face cu ajutorul functiei ***VirtualAlloc*** prin transmiterea valorii MEM_RESERVE pentru parametrul *flAllocationType*.

Maparea unui set de pagini rezervate este o alta operatie importanta in lucrul cu pagini. Prin mapare un interval de adrese rezervate anterior este asociat cu un set de pagini fizice. Daca rezervarea nu conduce la alocarea efectiva de memorie, maparea consuma pagini din memoria sistemului. Aceasta operatie se realizeaza tot cu VirtualAlloc, prin transmiterea valorii MEM_COMMIT pentru parametrul *flAllocationType*. Rezervarea si maparea pot fi realizate intr-o singura operatie, combinand cele doua valori cu un „SAU” pe biti.

In cazul in care cele doua operatii se fac separat, maparea se poate face treptat, in functie de necesitati. Cu alte cuvinte, este posibil ca un proces sa-si rezerve un spatiu de adrese in valoare de 12 MB din care mapeaza apoi blocuri mai mici (prima data 256 KB,

apoi 128 KB, apoi 1 MB samd). Cifrele sunt alese aproape arbitrar; ele nu sunt complet arbitrare deoarece cantitatile de memorie cerute trebuie sa satisfaca niste proprietati legate de granularitatea alocarilor si de dimensiunea paginii pe sistemul pe care ruleaza.

La mapare, paginile primesc un set de atribute de protectie. Programul specifica daca paginile sunt accesibile pentru citire, scriere sau executie. Aceste atribute pot fi modificate ulterior cu ajutorul functiei **VirtualProtect**, tinand cont de niste limitari impuse de atributele stabilite la mapare.

Paginile mapate pot fi eliberate cu ajutorul functiei **VirtualFree**, folosind valoarea MEM_DECOMMIT pentru argumentul *dwFreeType*. O rezervare poate fi stearsa cu totul apeland aceeasi functie cu valoarea MEM_RELEASE pentru parametrul dwFreeType. La stergerea unei mapari, trebuie specificate pagini valide. In caz contrar, operatia de eliberare esueaza. La stergerea unei rezervari, trebuie specificat un interval de adrese obtinut anterior prin VirtualAlloc, dar nu conteaza daca intervalul contine pagini mapate, rezervate sau un amestec.

Interfata de management al memoriei virtuale din Windows mai ofera si alte functionalitati pentru *interogarea proprietatilor paginilor* si *manipularea* lor, dar nu vom intra in detalii. Trebuie retinut doar ca pe acest alocator se construiesc toate celelalte functii de alocare disponibile la diverse niveluri.

Al doilea alocator important din Windows ofera **functionalitate de heap** si este accesibil (surpriza) prin intermediul unui set de functii al caror nume incepe cu Heap. Daca functiile pentru managementul memoriei virtuale lucreaza cu blocuri de pagini, functiile pentru heap sunt utilizate in alocarea unor blocuri relativ mici de memorie.

Fiecare proces dispune implicit de un heap accesibil prin intermediul functiei **GetProcessHeap**. Aplicatiile pot crea heapuri suplimentare apeland **HeapCreate** si pot specifica dimensiunea initiala a heapului si, eventual, dimensiunea maxima a acestuia.

Alocarea propriu-zisa de memorie se face cu **HeapAlloc**, care returneaza un bloc contiguu de memorie de marimea ceruta in cazul in care un astfel de bloc este disponibil. Redimensionarea blocurilor alocate anterior se poate face cu **HeapReAlloc**, iar eliberarea memoriei alocate se face cu **HeapFree.** Daca un heap nu mai este necesar, el poate fi eliberat cu un apel catre **HeapDestroy**.

Prin natura implementarii, un heap functioneaza optim atunci cand blocurile alocate au dimensiuni egale. In cazul in care o aplicatie aloca un numar extrem de mare de blocuri de dimensiuni variate, fragmentarea heapului poate reprezenta o problema serioasa. Prin fragmentare se intelege situatia cand memoria libera din heap este impartita in multe

blocuri de dimensiuni mici.

De exemplu, o cerere pentru alocarea unui bloc de 50 KB de memorie dintr-un heap care dispune de 512 KB de memorie va esua daca aceasta memorie este imprastiata sub forma multor blocuri de cate 32 KB. Pentru a preveni fragmentarea, poate fi utilizat un mod special de functionare a heapului, numit heap cu fragmentare scazuta.

Functiile de heap ofera si alte functionalitati, ceva mai avansate. De exemplu, in mod implicit accesul la heap este serializat, dar pot fi create heapuri fara serializare daca se stie ca aplicatia nu solicita memorie din acelasi heap in mod concurent, din mai multe fire de executie. De asemenea, implicit HeapAlloc returneaza NULL daca nu poate satisface cererea de alocare primita, dar heapul poate fi configurat sa ridice o exceptie in aceasta situatie.

Pe langa functiile Heap, Windows mai contine doua interfete de alocare, numite generic **Global** si **Local**. Aceste interfete exista doar pentru compatibilitatea cu platforma Windows pe 16 biti si sunt implementate intern prin apeluri Heap.

Referinte

[1] Charles Petzold, Programming Windows (Fifth Edition)
[2] Jeffrey Richter, Advanced Windows (Third Edition)
[3] Andrew S. Tanenbaum, Sisteme de operare moderne, Byblos, 2004

Concluzii

Intelegerea limitarilor impuse de partea hard a unui sistem de calcul si a modului de gestiune a memoriei de catre un sistem de operare este foarte importanta in elaborarea de aplicatii si pentru a evita brese de securitate in programe.

Algoritmii implementati in gestiunea memoriilor in cele doua sisteme de operare analizate Linux si Windows sunt complicati si sunt mereu supusi unor modificarii pentru a se adapta la noile tehnologii implementate.