GESTIUNEA MEMORIEI IN SISTEMELE DE OPERARE WINDOWS SI LINUX

Ancu Mihai (441A)
Țipțer Laurențiu Radu (431A)

Gestiunea memoriei in Windows

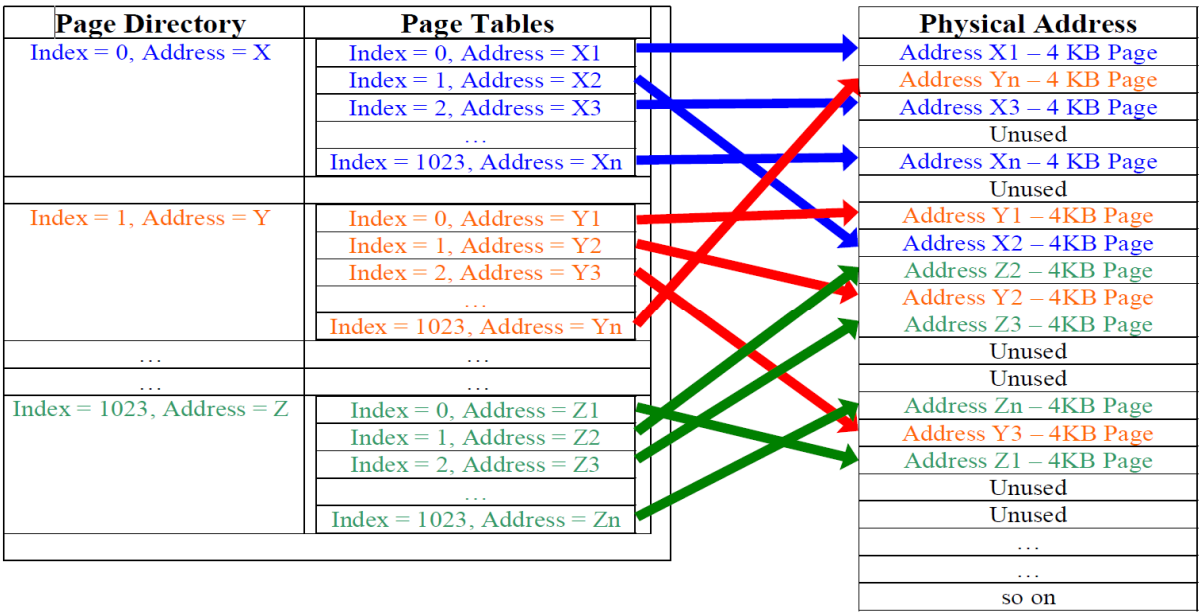
(Țipțer Laurențiu Radu)

Introducere

Windows-ul pe 32 de biți in sisteme x86 poate accesa până la 4GB de memorie fizică. Acest lucru se datorează faptului că adresa magistralei procesorului, care este de 32 de linii sau 32 de biți, poate accesa numai in intervalul de adrese de la 0x00000000 la 0xFFFFFFFF care este de 4GB. Windows permite, de asemenea, fiecare proces de a avea propriul sau loc in cei 4 GB de spațiu de alocare logică. 2GB de spațiu de adrese din partea inferiara este disponibil pentru aplicatiile folosite de utilizator si 2GB din spatiul de adrese superioara este rezervat pentru Windows. Deci cum ofera Windows 4GB de adrese pentru mai multe procese, atunci când memoria totală ce se poate accesa este limitat la 4GB? Pentru a realiza acest lucru Windows utilizează o caracteristică de procesor x86 (386 și mai sus), cunoscuta sub numele de paginare. Paginarea permite software-ului sa utilizeze o adresa de memorie diferita (cunoscut sub numele de adresa logică) in loc sa foloseasca adresa memoriei fizice. Unitatea de paginare a procesorului traduce transparent această adresă logică in cea fizică. Acest lucru permite fiecare proces în sistem să aibă propria adresă logică in cei 4GB de spațiu. Pentru a înțelege acest lucru în mai multe detalii, să ne aruncăm o privire la modul de paginare în tehnologia x86.

Paginarea intr-un procesor x86

Procesorul x86 împarte spațiul de adrese fizice (sau memoria fizică) în pagini de 4 KB. Astfel, pentru a adresa 4GB de memorie, vom avea nevoie de un Mega (1024x1024) de 4KB de pagini. Procesorul folosește o structura pe doua nivele pentru a se referi la aceste pagini de un mega. Poate fi gândia ca o matrice bidimensionala cu 1024x1024 elemente. Prima dimensiune este cunoscuta ca pagina directoare și a doua dimensiune este cunoscuta sub numele de tabela de pagini. Astfel, putem crea o pagina directoare cu 1024 intrări, fiecare cu puncte pentru o tabela de pagini. Acest lucru va permite de a avea 1024 de tabelele de pagini. Fiecare tabela de pagini, la rândul său, poate avea 1024 intrări, fiecare conținand o pagină de 4 KB. Grafic arata asa:



Fiecare intrare din pagina directoare (sau PDE) este de 4 octeți în dimensiune și duce la o tabela de pagini. Similar fiecare tabel de pagini (sau PTE) este de 4 octeți și duce la o adresă fizică de 4KB. Pentru a stoca 1024 PDE fiecare conținând 1024 PTE, vom avea nevoie de o memorie totală de 4 x 1024 x 1024 bytes adică 4MB. Astfel, pentru a împărți întregul spațiu de adrese de 4 GB în pagini de 4KB, avem nevoie de 4 MB de memorie. Așa cum sa discutat mai sus, întregul spațiu de adrese este împărțit în pagini de 4KB. Deci, atunci când un PDE sau PTE este folosit, 20 de bitii mai semnificativi ofera o pagina de adrese de 4KB iar cei 12 biti mai putin semnificativi sunt folositi pentru a stoca informatii de protecție a paginii și alte informații cerute de sistemul de operare pentru buna functionare. Acei 20 de biti semnificativi, care reprezinta adresa fizica, este cunoscuta ca Page Frame Number (or PFN).

Managementul tabelei de pagini in Windows

In Windows fiecare proces are propriile pagini directoare si tabele. Astfel, Windows aloca 4 MB din acest spațiu pentru fiecare proces. Cand un proces este creat, fiecare intrare în pagina directoare conține adresa fizica unui tabel de pagini. Intrările din tabelul de pagini sunt fie valide sau invalide. Intrările valide conține adresa fizică a paginii de 4KB alocate procesului. O intrare invalida conține biți speciali pentru a marca invaliditatea și aceste intrari sunt cunoscute ca PTE-uri invalide. In timp ce memoria este alocata proceselor, aceste intrări în tabelul de pagini sunt ocupata cu adresa fizică a pagini alocate. Trebuie să ne amintim un lucru, că un procesele nu știu nimic despre memoria fizică și o folosește numai adresele logice. Procesorul si Windows Memory Manager face transparent legaturile intre adresele logice si cele fizice. Adresa la care se afla pagina directoare a unui proces în memoria fizică este se numeste ca adresa de bază a paginii directoare. Această adresa de bază este stocată într-un registru special al procesorului, numit CR3 (pe arhitectura x86). Cand se schimba procesul, Windows incarca, in CR3, noua valoare a adresei de baza a paginii directoare. În acest fel fiecare proces are

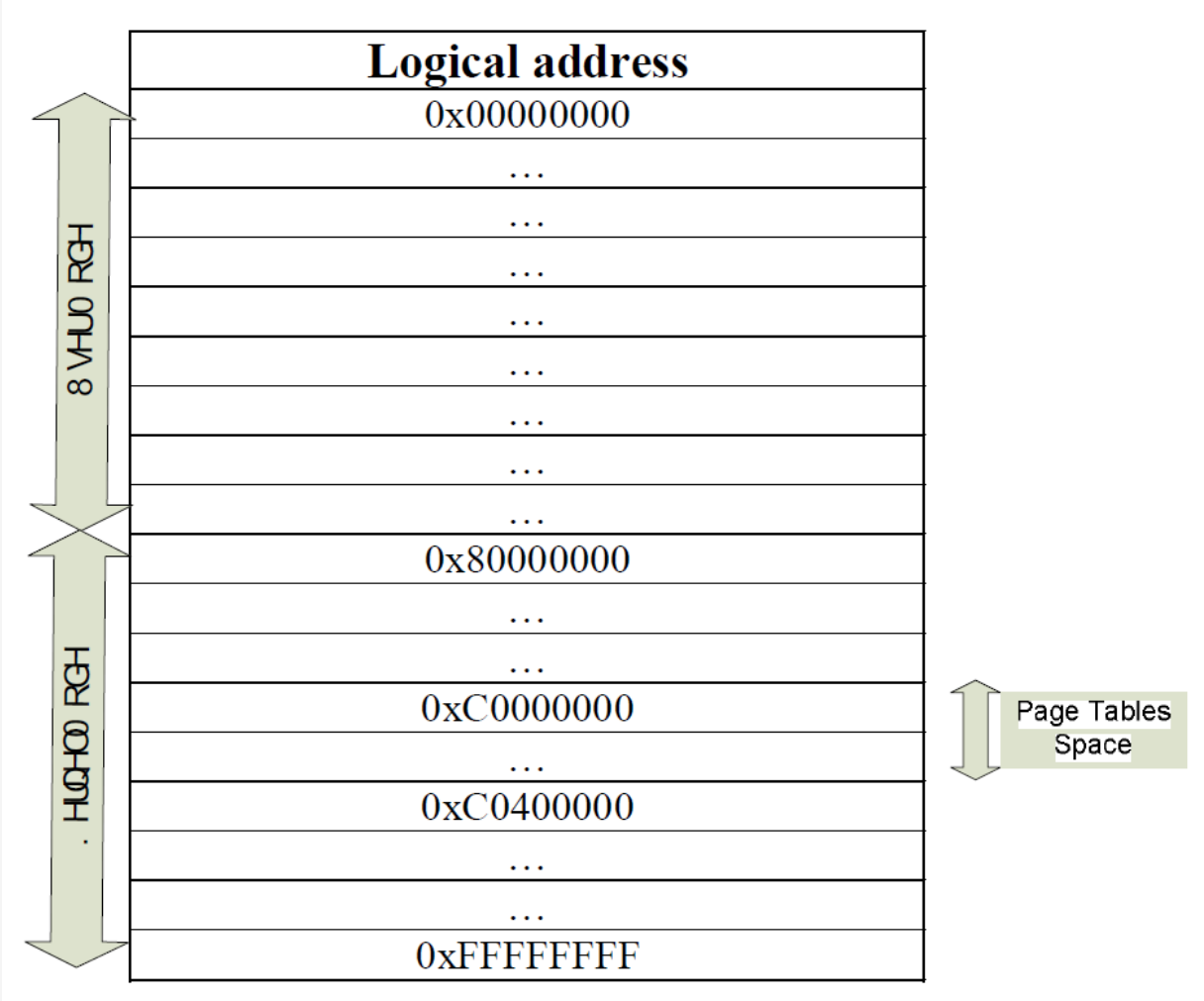
propriul spatiu in adresa fizică de 4GB. Desigur memoria totală alocată la un moment dat la toate procesele poate depăși valoarea totală de memorie RAM + dimensiunea fișierul de paginare, dar cu lucrurile discutate mai sus Windows permite să-i dea fiecarui proces propria adresa logica (sau virtuala) in spatiul de 4GB, dar poate folosi doar memoria alocata procesului. Dacă un proces încearcă să acceseze o adresă nealocata, acesta va primi o încălcare de acces, deoarece PTE-ul corespunzătoar adresei indică o valoare invalidă. De asemenea, procesul nu poate aloca mai multă memorie decât ceea ce este disponibil în sistem. Această metodă de separare a memoriei logice de memorie fizică are o mulțime de avantaje. Un proces primeste un spațiu continuu in memorie astfel încât programatorii nu trebuie să se ingrijoreze cu privire la fragmentarea datelor, spre deosebire de MS-DOS. De asemenea, permite Windows-ul a rula mai multe procese și să folosească memorie fizică a mașinii fara riscul, a mai multe procese, sa scrie in acelasi loc. O adresă logică într-un proces nu va indica, în memoria fizică, spatiul alocat un alt proces (cu excepția cazului în care se utilizează un fel de memorie partajată). Astfel, un proces nu poate citi sau scrie in memoria unui alt proces. Tranzitia de la adresa logica la adresa fizică se face de către procesor. O adresa logică de 32 de biti este împărția în trei bucati. Procesorul încarcă adresa fizică a paginii director stocata în CR3, apoi utilizează primii 10 biți semnificativi din adresa logica ca un index în pagina director. Acest lucru oferă procesorului o cale de la pagina directoare (PDE) catre o tabela de pagini. Următorii 10 biți sunt utilizați ca un index în tabela de pagini. Folosind acesti 10 de biți, acesta ia o adresa care duce la o pagna de 4KB din memoria fizica. Cei mai putini semnificativi 12 biti sunt folosite pentru a marca bitii individuali din acea pagina.

Protectia memoriei

Windows oferă protecție de memorie pentru toate procesele, astfel încât un proces nu poate accesa memoria unui alt proces. Acest lucru asigură în același timp buna funcționare a mai multor procese, rulate in acelas timp. Windows asigura această protecție facand următoarele:

- în PTE este pusa doar adresa fizica a memoriei alocate pentru un proces. Acest lucru asigură că, in cazul in care, procesul încearcă să acceseze o adresă care nu ii este alocata va primi acces interzis.
- un proces aleator poate încerca să isi modifice tabela de pagini, astfel încât să poată accesa memoria fizică aparținând unui alt proces. Windows protejeaza acest tip de atacuri depozitand tabelele in spatiul de adrese din nucleu. Din discuția anterioara stim ca din 4GB de spatiu logic alocat unui proces, 2GB este dat utilizatorului și 2 GB este rezervat pentru nucleu Windows-ului. Deci o aplicatie a utilizatorului nu poate accesa direct sau modifica tabelele de pagini. Desigur, în cazul în care un driver al nucleu lui vrea să facă asta, se poate face acest lucru pentru că o dată aflat în modul nucleu, practic are acces total la întregul sistem.

Harta memoriei logice in Windows



Windows oferă 2GB din partea inferioara (sau 3 GB, în funcție de setarile din fisierul boot.ini) spațiu de adrese logică proceselor utilizatorului și 2 GB din partea superioare (sau 1 GB, în funcție de boot.ini) nucleului Windows-lui. Din totalul spațiului de adrese nucleuluiului, acesta își rezervă adrese de la 0xC0000000 la 0xC03FFFFF pentru tabelele de pagini și pagina directoare. Fiecare proces are tabelele de pagini situate la adresa logică 0xC0000000 și pagina director situata la adresa logică 0xC0300000. Aceasta aranjare a memoriei arata in felul urmator:

Index / Logical Address	0x0	0x1	.	0x80	.	0x300 (768)	0x34A (842)	.	.	0x400 (1023)
0xC0000000 0x6A078###	0x10480###	-				-				
0xC0001000 0x45045###	-	-					0x34005###			
0xC0002000										
-										
-										
0xC0100000										
-										
-										
0xC0300000 0x13453###	P_PT 0x6A078###	-		P_PT 0x45045###	-	PDB 0x13453###			-	
0xC0301000										
-										
-										
0xC03FF000										

Adresa fizică la această director pagină este stocată în CR3. Cele 1024 de adrese incepand de la 0xC0300000 reprezintă pagina directoare de intrare (PDE). Fiecare PDE conține o adresă fizică pe 4 octeti care indică o tabela de pagini. Fiecare tabela de pagini are 1024 intrari care fie conține o adresă fizică indică spre o pagină fizică de 4KB sau conține o intrare invalidă. Deci, de ce Windows utilizează adresa logică 0xC000000000 pentru a stoca tabele cu pagini și adresa 0xC0300000 pentru a stoca pagina directoare? Necesitatea de stocare tabelelor de pagini în memorie este pentru securitate, deoarece aplicatiile utilizatorilor nu ar trebuie sa manipuleze tabelele cu pagini. Prin urmare, tabelele de pagini ar trebui să fie spațiul de adrese logice din nucleu. Windows oferă, de obicei partea inferioara, de 2 GB, catre procesele lansate de utilizator și isi rezerva cei 2GB din parta superioara pentru nucleu. Dar, cu o modificare in fisierul Boot.ini, permite proceselor utilizatorului a accesa 3GB de memorie din partea inferioara, 0xC0000000 fiind adresa următoare după 3GB. Există câteva alte aspecte importante despre tabele de pagini și aranjarea paginii director în memorie. Pentru a înțelege mai bine, să ne uităm la modul în care tabelele de pagini și pagina director sunt dispuse. Pentru a-l ușor să nderstand face, am atras tabelele de pagini pentru un proces fals cu intrări relevante evidențiate. Rețineți că fiecare intrare index este de 4 octeți în dimensiune. P_PT reprezintă adresa fizică a unei tabele de pagini. PPB reprezintă adresa fizică de baza a directorului de pagina a procesului corespunzător, adică reprezintă adresa fizică corespunzătoare pentru a adresa logica 0xC0300000 pentru acest proces. Aceasta valoaree este, de asemenea, stocata în CR3. Ne amintim, Windows poate utiliza numai adresa logica ca să acceseze orice locație in memorie, inclusiv directorul de pagină, astfel încât pentru a accesa directorul de pagina si tabele de pagini, este necesar a pune referinte propri in directorul de pagini. Adresa intrări fizice prezentate mai sus vor fi diferite pentru fiecare proces, dar fiecare proces va avea intrarea sa PPB depozitate la indicele de 0x300 a directorului de pagină.

Vom efectua o traducere de la o adresa logica la adresa fizică la 4 locatii diferite pentru a vedea semnificația intrării PPB, aspectul tabelelor de pagini și cum anume funcționează traducerea adreselor. Adresele pe care le va traduce sunt 0x2034AC54, 0xC0000000, 0xC0300000, 0xC0300000 [0x300] ie 0xC030C00. Prima adresă este o adresa logica normala a modului utilizator pentru un proces, a doua adresa este prima adresă logică a primei tabele de pagini în spațiul de adrese logice, a treia este adresa logică a paginii directoare de baza și adresa patra este adresa logică a unei intrare specială, asa cum se va vedea în timpul traducerii. Să presupunem CR3 duce la o adresă fizică 0x13453 # # #. Așa cum sa arătat mai devreme, 12mai putin semnificativi sunt folositi pentru a stoca informații pentru protecția paginii și alte informații necesare sistemului de operare. Cei 20 de biți semnificativi reprezintă Page Frame Number sau PFN care este adresa fizica a unei pagini de 4KB aliniate. Adresa fizică reală corespunzătoare PFN va fi 0x13453000. Să ne facem acum traducerea:

0x2034AC54 poate fi reprezentat ca 0010000000 1101001010 110001010100. Cei mai semnificativi 10 biti (0010000000) dă indicele paginii directoare. Transformand in hexazecimal, cei 10 de biți dau valoarea de 0x080.

Din CR3, știm că pagina directoare este situata la adresa fizică 0x13453000 și din discuția de mai sus știm de asemenea că pagina directoare este situat la adresa logică 0xC0300000.

Astfel 0xC0300000 [0x080] va da adresa de tabelul de pagini, care este P_PT. Din tabelul de mai sus, putem vedea că această adresă este reprezentată de tabelul de pagini, la adresa logică 0xC00001000 (sau adresă fizică 0x45045000). Acum vom folosi următorii 10 de biți (1101001010) (sau 0x34A) să indice tabela de pagini.

Adresa 0xC00001000 [0x34A] ne va da adresa fizică a unei pagini de 4KB (0x34005000) din tabelul de mai sus.

Cei mai putini semnificativi 12 biti (110001010100 sau 0xC54, in hexazecimal) sunt folositi pentru a marca la octetul din pagina de 4KB situat la 0x34005000. Adresa fizică finală, corespunzatoare adresei logice 0x2034AC54, va fi 0x34005C54.

Bibliografie:

https://en.wikipedia.org/wiki/Memory_management

http://msdn.microsoft.com/en-us/library/windows/desktop/aa366525(v=vs.85).aspx

http://tldp.org/LDP/tlk/mm/memory.html

Gestiunea memoriei in Linux

Ancu Mihai (441A)

Introducere

Linux este un sistem de operare bazat pe Unix dezvoltat de Linus Torvalds in 1991. Este un sistem multi-tasking, multi-user care se comporta ca Unix in ceea ce priveste kernelul si perifericele. Kernelul Linux este un kernel monolitic, construit din diverse module care se ocupa de diverse aspecte ale sistemului de operare ca sistemul de fisiere, managementul memoriei, programarea proceselor etc.

Managementul memoriei este cea mai complexa slujba a kernelului Linux. Cum procesorul poate accesa direct memoria RAM, procesele executate trebuie pastrate in memorie. Memoria trebuie sa tina atat codul sistemului de operare, cat si procesele utilizatorului. Deci sistemul de management al memoriei include suport pentru pastrarea in memorie a mai multor procese simultan, suport pentru memorie virtuala, acolo unde procesele sunt mai mari decat poate pastra memoria fizica, protejeaza procesele unele de celelalte si se ocupa de mutarea acestora in memorie in timpul rularii dupa nevoi. Memoria virtuala introduce probleme care implica fragmentarea memoriei. De asemenea, sistemul de management al memoriei ar trebui sa gaseasca solutii la aceasta problema. Systemul din Linux care se ocupa de aceste taskuri este un subsistem al kernelului denumit Kernel Memory Allocator. Acest subsistem este bazat pe alocatorul Slab construit peste Buddy System.

Un proces are cerinte proprii de memorie, iar spatiul de adrese ocupat de un proces in memoria fizica se numeste spatiul adresei fizice al procesului. In sisteme cu memorie virtuala, adresele de memorie virtuala sunt de obicei exprimate ca spatii de adrese logice sau virtuale si sunt referinte diferite de adresele fizice.

Memoria fizica ar trebui sa pastreze atat sistemul de operare, cat si procesele lansate de utilizator. O implementare simpla pentru a proteja codul sistemului de operare si procesele de interactiunea dintre ele ar fi impartirea memoriei in partitii si aranjarea codului si proceselor in aceste partitii. Echipamentul hardware necesar consta in registre de relocatare si registre de limitare. Pentru o adresa logica data, valoarea este comparata valoarea din registrul de limitare. Daca nu este mai mica, este generata o eroare. Altfel valoarea adresei este adaugata in registrul de relocatare pentru a obtine adresa fizica Gradul de multiprogramare depinde de numarul de partitii in care e impartita memoria.

O problema tipica ce trebuie adresata este fragmentarea. Aceasta are de obicei loc atunci cand memoria este sparta in bucati dintre care niciuna nu poate satisface o noua cerere, dar cand bucatile sunt asezate impreuna, cererea poate fi satisfacuta.

Fragmentarea externa este tipica alocarii de locatii invecinate atunci cand cerintele de memorie variaza ca marime. In asemenea cazuri, spatiul liber este fragmentat si nu permite indeplinirea unei cereri mari.

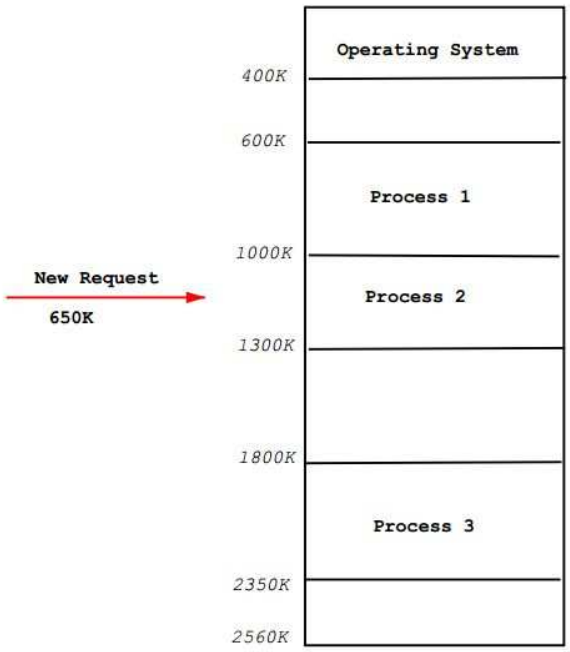


Fig. 1 - Exemplu de fragmentare externa (figura preluata din lucrarea "Linux Memory Management" a autorului : Ramani Yellapragada (26 mai 2003))

Algoritmul Buddy System

Managementul memoriei implica indeplinirea eficienta a nevoilor de memorie a proceselor cu memoria disponibila care este limitata. De obicei programele au nevoie de blocuri de memorie care implica mai multe pagini. De obicei un program de alocare se ocupa de asemenea taskuri. Acest program ar trebui sa aiba o strategie eficienta pentru alocarea de zone invecinate in memorie si pentru a avea in vedere prevenirea fragmentarii externe. Una din aceste strategii este algoritmul Buddy System.

In Buddy System, alocatorul aloca blocuri de dimensiune predefinita. Cand alocatorul primeste o cerere de memorie, acesta rotunjeste cererea la urmatoarea marime a blocului. Apoi cauta in memorie blocuri de o asemenea marime. Daca un asemenea bloc este gasit, aloca acel bloc procesului si il marcheaza ca nedisponibil. Daca nu este gasit niciun bloc de marime necesara, se cauta un bloc de urmatoarea marime, iar la gasire se imparte si se aloca spatiul necesar procesului, iar cealalta parte este inclusa in lista de spatii libere.

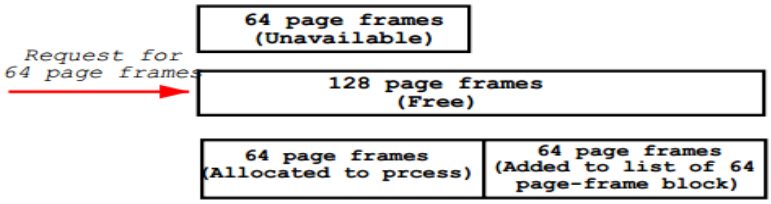


Fig. 2. Exemplu de algoritm Buddy System pentru un request de 64 de pagini

Numele algoritmului Buddy System vine de la modul in care elibereaza paginile. Acesta incearca sa lipeasca blocuri de spatii libere de marime k in blocuri de marime 2k. Blocurile prietene (“buddy”) sunt blocuri de aceeasi marime, situate in locatii de memorie invecinate. Lipirea este repetata recursiv pana la atingerea celei mai mari dimensiuni a blocului si nu mai exista alte blocuri libere in lista. Singura problema care poate aparea prin folosirea Buddy System este fragmentarea interna.

Descrierea memoriei fizice

Linux este disponibil pentru multe tipuri de arhitecturi, iar descrierea memoriei este facuta intr-un mod dependent de arhitectura.

Principalul concept prezent este NUMA (Non Uniform Memory Access). La masinile mari, memoria este aranjata in grupuri care au cost de acces diferit in functie de distanta pana la processor. De exemplu poate exista un bank de memorie pentru fiecare procesor sau un bank foarte aproape de dispozitive, cu cost redus pentru canalul DMA. Fiecare bank de memorie este numit un nod si conceptul este reprezentat in Linux de o structura pglist_data chiar daca arhitectura este UMA. Aceasta structura se poate referi prin pg_data_t. Fiecare nod din sistem este pastrat intr-o lista numita pgdat_list si nodurile sunt legate prin pg_data_t->node_next.

Fiecare nod este impartit in mai multe blocuri numite zone care reprezinta spatii in memorie. O zona este descrisa de structura zone_struct si poate fi de tipul ZONE_DMA, ZONE_NORMAL sau ZONE_HIGHMEM. Aceste zone sunt specializate pe aplicatii.

ZONE_DMA este memoria din spatiile mai joase ale memoriei fizice de care este nevoie pentru anumite tipuri de dispozitive ISA. Memoria din ZONE_NORMAL este direct asociata kernelului in regiunile inalte ale spatiului liniar de adrese. ZONE_HIGHMEM este restul memoriei si nu este direct asociata kernelului.

Zonele la x86:

ZONE_DMA Primii 16 MiB ai memoriei (1 MiB = 2^{20} bytes)

ZONE_NORMAL 16 MiB -> 896 MiB

ZONE_HIGHMEM 896 MiB pana la sfarsit.

Multe operatii efectuate de kernel nu pot avea loc decat in ZONE_NORMAL ceea ce face aceasta zona cea mai importanta din punctul de vedere al performantei.

Cand aloca o pagina, Linux foloseste o politica node-local allocation policy pentru a aloca memorie din nodul cel mai apropiat de procesor. In cazul in care procesele ruleaza pe acelasi procesor, este cel mai probabil ca nodul curent sa fie folosit. Structura este descrisa dupa cum urmeaza in <linux/mzone.h>:

```
Typedef struct pglist_data {

Zone_t node_zones[MAX_NR_ZONES];

Zonelist_t node_zonelists[GFP_ZONEMASK+1];

Int nr_zones;

Struct page *node_mem_map;

Unsigned long *valid_addr_bitmap;

Struct bootmem_data *bdata;

Unsigned long node_start_paddr;

Unsigned long node_start_mapnr;

Unsigned long node_size;

Int node_id

Struct pglist_data *node_next;

} pg_data_t;
```

Descriere:

Node_zones – zonele din acest nod (ZONE_HIGHMEM, ZONE_NORMAL, ZONE_DMA)

Node_zonelists – ordinea in care se prefera alocarea zonelor. Build_zonelists in mm/page_alloc.c specifica ordinea cand este chemata de free_area_init_core(). O alocare esuata in ZONE_HIGHMEM poate ajunge in ZONE_NORMAL sau in ZONE_DMA.

Nr_zones – numarul de zone in nodul curent (intre 1 si 3). Nu toate nodurile vor avea 3 zone. De exemplu un bank CPU poate sa nu aiba ZONE_DMA.

Node_mem_map – prima pagina a struct page array ce reprezinta fiecare cadru fizic din nod.

Valid_addr_bitmap –o harta de biti care descrie “gaurile” din nod. Este folosit numai de Sparc si Sparc64.

Bdata – prezinta interes numai alocatorului de memorie boot

Node_start_paddr – adresa fizica de inceput a nodului

Node_start_mapnr – arata offsetul in mem_map

Node_size – numarul total de pagini in zona curenta

Node_id – id-ul nodului, incepand de la 0

Node_next – pointer catre urmatorul nod intr-o lista terminata cu null

Zone:

Zonele sunt descrise de o structura zone_struct si sunt de obicei referite de typedef zone_t, care pastreaza informatii ca statistici de folosire a paginilor, informatii despre spatiul liber etc. Se declara in <linux/mmzone.h> astfel:

```
Typedef struct zone_struct {
Spinlock_t        lock;
Unsigned long   free_pages;
Unsigned long   pages_min, pages_low, pages_high;
Int                 need_balance;
Free_area_t      free_area[MAX_ORDER];
Wait_queue_head_t *wait_table;
Unsigned long   wait_table_size;
Unsigned long   wait_table_shift;
Struct pglist_data *zone_pgdat;
Struct page          *zone_mem_map;
Unsigned long      zone_start_paddr;
```

Unsigned long zone_start_mapnr;

Char *name;

Unsigned long size;

}zone_t;

Descriere:

Lock – protejeaza zona de accese concurente.

Free_pages – numarul total de pagini libere din zona

Wait_table – un hash table al cozii de procese in asteptare pentru eliberarea unei pagini.

Cand memoria disponibila este prea putina, este chemat kswapd pentru a incepe sa elibereze pagini. Fiecare zona are 3 caracteristici : pages_low, pages_min si pages_high care arata sub cata presiune se afla acea zona. Numarul de pagini pentru pages_min este calculat in functia free_area_init_core in timpul initializarii memoriei in functie de marimea zonei. Initial este calculat ca ZoneSizeInPages/128.

pages_low - cand numarul indicat de pages_low de pagini este atins, este chemat kswapd pentru a elibera pagini. Valoarea implicita este de doua ori valoarea pages_min.

pages_min - cand este atins numarul pages_min de pagini libere, alocatorul va chema kswapd pentru a elibera pagini intr-o maniera sincrona, numita calea de revendicare directa.

Pages_high – dupa ce a fost chemat kswapd pentru a elibera pagini, acesta nu se va opri din procesul de eliberare a paginilor pana cand nu va fi atins numarul de pages_high pagini libere. Dupa ce acest prag a fost atins, kswapd va intra in modul sleep.

Coada de asteptare a zonei

Cand se efectueaza operatii de I/O pe o pagina, cum ar fi procesele de page-in sau page-out, pagina respectiva va fi incuiata pentru a preveni accesari asupra ei, care ar duce la date inconsistente. Procesele care doresc sa foloseasca aceasta pagina vor intra intr-o coada de asteptare chemand metoda wait_on_page(). Cand operatia I/O se incheie, pagina va fi descuiata cu UnlockPage() si procesele care erau in asteptare vor fi “trezite”. Fiecare pagina ar putea avea o coada de asteptare, dar acest lucru ar duce la costuri foarte mari de memorie, de aceea coada este stocata in zone_t. Ar fi posibil sa existe o singura coada in aceasta zona, dar acest lucru ar duce la trezirea tuturor proceselor care asteapta eliberarea unei pagini specifice odata cu unei singure pagini din multimea de pagini. Acest lucru ar duce la o problema de tipul “thundering herd”(apare atunci cand este trezit un numar mare de procese odata cu aparitia unui eveniment, dar nu poate fi deservit decat cate un proces). De aceea este folosit un hash table de asteptari in zone_t ->wait_table.

Initializarea zonei

Zonele sunt initializate dupa incheierea procesului de paging_init() (paginile din tabelul kernel au fost puse la punct). Desi fiecare arhitectura are modalitatea proprie de a executa acest proces, obiectivul este intotdeauna acelasi: de a determina ce parametri trebuie trimisi catre free_area_init() pentru arhitectura UMA sau catre free_area_init_node() pentru arhitectura NUMA. Lista de parametri:

Nid – id-ul nodului care este identificatorul logic al nodului in care se afla zonele care trebuie initializate

Pgdat – proprietatea pg_data_t a nodului care trebuie initializata.

Pmap – este setat de catre free_area_init_core() pentru a arata inceputul array-ului lmem_map alocat nodului

Zones_sizes – este un array care contine dimensiunea fiecarei zone in pagini

Zone_start_paddr – este adresa fizica de inceput a primei zone

Zone_holes – este un array care contine dimensiunea totala a gaurilor

Este de datoria functiei de baza free_area_init_core() sa umple fiecare zone_t cu informatii relevanta si sa aloce mem_map array fiecarui nod.

Initializarea mem_map

Zona mem_map este creata in timpul pornirii sistemului in doua maniere: la NUMA mem_map este vazut ca un array virtual care incepe de la PAGE_OFFSET. Free_area_init_node() este chemat pentru fiecare nod activ in sistem care aloca portiuni din acest array nodului care trebuie initializat. La sisteme de tip UMA free_area_init() foloseste contig_page_data ca nod si mem_map global ca mem_map local pentru acest nod.

Maparea paginilor catre zone

Fiecare pagina fizica din sistem are asociata o structura page care este folosita pentru a ii urmari starea. Se obisnuia ca aceasta structura sa tina o referinta catre zona sa in page->zone, fapt care a fost considerat inutil deoarece chiar si un pointer atat de mic consuma multa memorie in conditiile in care exista mii de structuri page. De aceea campul zone a fost eliminat si in locul sau a fost introdus ZONE_SHIFT. Este declarat un tabel zone_table de zone in mm/page_alloc.c astfel:

Zone_t *zone_table[MAX_NR_ZONES*MAX_NR_NODES];

EXPORT_SYMBOL(zone_table);

High Memory

Spatiul de adrese folosibil de kernel (ZONE_NORMAL) este limitat ca marime, de aceea kernelul foloseste conceptul de high memory. Doua limite de high memory exista pe sistemele de 32 si 64 de biti, una la 4GiB si cealalta la 64GiB. Limita de 4GiB se refera la cantitatea de memorie care poate fi adresata de adrese fizice de 32 de biti. Pentru a accesa memorie intre 1GiB si 4GiB, kernelul mapeaza temporar pagini din high memory in ZONE_NORMAL cu kmap().

Cea de-a doua limita de 64GiB se refera la Physical Address Extension (PAE) care este o inventie Intel pentru a permite folosirea unei cantitati mai mari de memorie RAM de catre sistemele de 32 de biti. Aceasta tehnologie permite folosirea a inca 4 biti pentru adresarea memoriei ceea ce duce la o cantitate maxima de 64GiB de memorie totala adresabila.

Desi in teorie, PAE permite procesorului sa adreseze 64GiB , in practica, procesele din Linux tot nu pot accesa atat de mult RAM, deoarece spatiul de adrese virtuale are 4GiB. In al doilea rand, PAE nu permite kernelului sa aiba atata memorie. Structura page folosita pentru a descrie fiecare pagina are nevoie de 44 de octeti si aceasta foloseste spatiul virtual de adrese din ZONE_NORMAL. Aceasta insemna ca pentru a descrie 1GiB de memorie este nevoie de aproximativ 11MiB de memorie kernel. Deci cu 16 GiB, 176 MiB de memorie va fi consumata, punand o presiune semnificativa pe zone_normal. Limita practica va fi de aproximativ 16 GiB. Daca e nevoie de mai multa memorie ar trebui cumparata o masina pe 64 de biti.

Managementul tabelului de pagini

Daca alte sisteme de operare au obiecte care efectueaza managementul paginilor fizice, Linux foloseste conceptul de tabel cu trei nivele in codul sau care este independent de arhitectura, chiar daca arhitectura pe care este suprapus nu dispune de asa ceva. Desi conceptual, acest lucru este usor de inteles, distinctia dintre tipuri diferite de pagini este foarte vaga, iar tipurile de pagini sunt identificate dupa flaguri sau dupa listele pe care se afla si nu in functie de obiectele de care apartin.Arhitecturile cu un alt tip de MMu(memory management unit) vor trebui sa emuleze tabelele cu trei niveluri Linux.

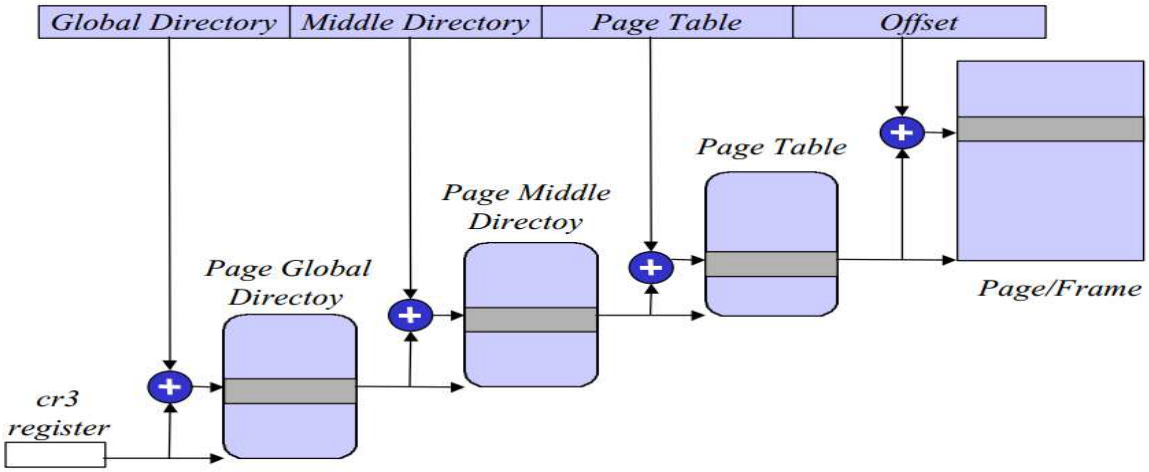


Fig. 3 . Exemplu de adresa logica.

(imagine preluata de la: http://www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/Lecture14.pdf)

Concluzie:

Domeniul managementului de memorie este mare, complex si necesita foarte mult timp de studiu, iar implementarile practice sunt foarte dificile. Aceasta se datoreaza si faptului ca este greu de modelat comportamentul sistemelor multi programate reale. Ceea ce este disponibil este o examinare teoretica a algoritmilor memoriei virtuale, care de foarte multe ori depind de datele specifice pe care le proceseaza. Sunt necesare simulari, deoarece modelarea a modului de programare a proceselor, de paginare si de interactiune dintre procese este foarte dificil de realizat. Politicile de inlocuire a paginilor sunt un domeniu in care s-a investit multa cercetare ca si problema ajustarii algoritmilor si politicilor la date diverse pentru a obtine rezultate optime.

Linux este de asemenea mare, complex si nu este pe deplin inteles decat de un grup foarte restrans de oameni. Dezvoltarea sa este rezultatul contributiei a mii de programatori cu grupuri diverse de specializari, din domenii diverse, cu timp mai mult sau mai putin disponibil. Primele implementari sunt dezvoltate avand in vedere fundatiile teoretice importante. Peste aceasta baza, s-a construit prin observatii ale comportamentului in timp real .

Bibliografie:

1. Ramani Yellapragada – “Linux Memory Management” (26 mai 2003)
2. Mel Gorman – “Understanding The Linux Virtual Memory Manager” (9 iulie 2007)
3. http://www.inf.fu-berlin.de/lehre/SS01/OS/Lectures/
4. http://www.chudov.com/tmp/LinuxVM/html/understand/node87.html