Curs 06: Memoria virtuală

Sumar curs anterior

memoria este folosită pentru a reţine cod/instrucţiuni şi date folosite de procesor
memoria fizică (RAM) este mai înceată decât procesorului: folosim memorie cache
pentru izolare şi utilizare facilă folosim memorie virtuală
fiecare proces are asociat un spaţiu virtual de adrese compus din zone
sistemul de operare asociază/mapează spaţiul virtual de adrese al fiecărui proces la spaţiul fizic (memorie RAM)
o formă veche de translatare era segmentarea
forma curentă este paginarea: împărţirea spaţiului virtual şi a spaţiului fizic în pagini
tabela de pagini face asocierea între pagini virtuale (pages) şi pagini fizice (frames)
translatarea este realizată de o componentă hardware (de pe procesor) numită MMU (Memory Management Unit)
tabela de pagini este reţinută în memorie şi este: mare (ocupă spaţiu) şi înceată (accesarea tabelei de pagini înseamnă acces la memorie)
PTBR (Page Table Base Register) reţine adresa în memorie a tabelei de pagini pentru procesul curent
tabela de pagini ierarhică combate dezavantajul spaţiului ocupat în memorie de tabela de pagini clasică (neierarhică)
TLB (Translation Lookaside Buffers) combate dezavantajul overhead-ului de translatare (nevoie de acces la memorie)
este nevoie de TLB flush la address space switch, când se schimbă tabelele de pagini

Spaţiul virtual de adrese al unui proces

un proces are un spaţiu virtual de adrese propriu
mecanismul de memorie virtuală asociază adrese virtuale cu adrese fizice, la nivel de pagini
zone de memorie virtuală:
* alocate static (la load time): cod/text, rodata, data, bss, biblioteci
* alocate dinamic (la run time): biblioteci, stivă, heap
load time: momentul în care este pornit un proces, lansarea în execuţie, când folosim ./a.out în linia de comandă, când se apelează exec()
run time: în momentul în care procesul rulează, se află în execuţie; bibliotecile se încarcă dinamică cu apeluri de tipul dlopen() (POSIX) sau LoadLibrary (Windows)
alocarea comandată din program (de exemplu malloc()) în cadrul unei zone înseamnă alocare de memorie virtuală

Alocarea la cerere (demand paging)

atunci când alocă o pagină virtuală nouă (folosind un apel malloc() sau mmap() sau alocând pe stivă) nu e obligatoriu să alocăm şi pagină fizică

spunem că "rezervăm" o pagină de memorie virtuală
amânăm alocarea efectivă (a paginii fizice) până la primul acces (lazy allocation)
numim acest proces de tip "lazy" on-demand paging (paginare/alocare la cerere) sau, mai simplu, demand paging memoriei: rezervare, alocare, mapare
spunem că rezervăm pagini virtuale, fără a avea neapărat corespondent într-o pagină fizică
rezervarea înseamnă marcarea paginii virtuale ca fiind folosite, în spaţiul virtual de adrese al procesului
atunci când avem un prim acces, alocăm pagina fizică
o dată cu alocarea paginii fizice realizăm asocierea între pagina virtuală şi pagina fizică, adică maparea
asocierea se face prin completarea intrării în tabela de pagini: în locul aferent paginii virtuale completăm adresa paginii fizice

Internele alocării la cerere (demand paging)

tabela de pagini e indexată după pagini virtuale: conţine adresa pagini fizice corespunzătoare, un bit de validitate, biţi de permisiuni
tabela de pagini e interpretată la nivel hardware de MMU (Memory Management Unit)
sistemul de operare păstrează informaţii suplimentare despre tabelele de pagini
atunci când se rezervă primă oară o pagină virtuală (la load time sau la run time), se marchează pagina ca nevalidă în tabela de pagini, şi ca "validă-nemapată" în informaţiile sistemului de operare
la primul acces, MMU generează excepţie (intrarea în tabela de pagini este nevalidă), excepţie e capturată de sistemul de operare, sistemul de operare alocă o pagină fizică, o completează în tabela de pagini şi marchează intrarea validă (şi în informaţiile sale interne)
excepţia este numită page fault (excepţie de acces la pagină)

Page fault (excepţie de acces la pagină)

atunci când se face acces la o pagină marcată "nevalidă" în tabela de pagini, MMU generează "page fault" către procesor
procesorul execută un page fault handler, o rutină de tratare înregistrată de sistemul de operare
sistemul de operare implementează în page fault handler mecanismul de demand paging şi alte mecanisme precum swapping sau copy-on-write
dacă o pagină este "nevalidă" în tabela de pagini şi este marcată ca "nevalidă/nealocată" în informaţiile sistemului de operare se generează către procesul în cauză excepţie de memorie de tipul "segmentation fault"
+demo cu page fault-uri pentru demand paging

Memorie partajată

două sau mai multe procese pot partaja pagini de memorie
intrările din fiecare tabelă de pagini (a fiecărui proces) referă aceeaşi pagină fizică

paginile virtuale pot diferi
+ diagramă memorie partajată
este folositoare pentru partajarea codului de executabil sau pentru codul pentru biblioteci partajate: mai multe procese create din acelaşi executabil vor partaja zona de cod (zona code/text) şi zona read-only (zona rodata); mai multe procese care folosesc aceeaşi bibliotecă partajată (shared library) vor partaja codul acelei biblioteci (zone code/text) şi zona read-only (zona rodata)
nu se partajează datele bibliotecilor, sunt modificate de fiecare proces în parte şi sunt proprii fiecărui proces
+ diagramă pentru partajare zone + biblioteci
util atunci când este creat un proces copil folosind fork(); iniţial se partajează *tot* spaţiul fizic: fiecare dintre procesul părinte şi procesul copil are un spaţiu virtual propriu, adică o tabelă de pagini proprie; dar spațiile virtuale referă acelaşi spaţiu fizic, adică tabelele de pagini au acelaşi conţinut

copy-on-write

tradiţional, la fork() se crea un spaţiu fizic nou, pentru noul proces, nu doar unul virtual; adică se face un duplicat al spaţiului fizic şi se populează în tabela de pagini a procesului copil nou creat
devine problematic pentru că de obicei după fork() se apelează exec() şi se înlocuieşte tot spaţiul virtual şi fizic, ceea ce înseamnă că duplicarea de la fork() a fost degeaba
soluţia este folosirea mecanismului copy-on-write atunci când facem fork()
când mecanismul copy-on-write, imediat după fork() fiecare spaţiu virtual (al procesului părinte şi al procesului copil) referă acelaşi spaţiu fizic; nu se alocă spaţiu fizic suplimentar (în afară de cel pentru noua tabelă de pagini)
procesul părinte şi copil partajează întreg spaţiul de adrese; e OK pentru zone non-writable (code/text şi rodata) dar nu pentru zone writable
atunci când unul dintre procese realizează o scriere, pagina corespunzătoare este duplicată, se actualizează intrarea în tabela de pagini a procesului care a făcut scrierea: referă pagina nouă; modificarea are loc doar în pagina nouă
este o formă de operaţie "lazy" (similar demand paging): se amână duplicarea paginii până la primul acces de scriere
duplicarea nu are loc la apel de citire
în detaliu:
* imediat după fork() paginile sunt marcate valide/read-only în tabela de pagini
* la o acţiune de scriere MMU generează page fault şi se apelează page fault handler-ul
* în informaţiile internele ale sistemului de operare pagina respectivă este marcată "validă/copy-on-write"
* sistemul alocă o pagină fizică nouă, duplică în ea conţinutul paginii iniţiale şi actualizează intrarea în tabela de pagini a procesului care a realizat scrierea, pagina este marcată read-write
* este reapelată instrucţiunea de scriere care a cauzat page fault-ul iniţial şi acum se face modificarea efectivă în pagina fizică nou alocată

dacă se fac mai multe apeluri fork() (mai multe procese copil sau nepot sau strănepot) acestea partajează toate spaţiul fizic şi au paginile marcate read-only; duplicarea paginii se întâmplă atunci când un proces realizează o acţiune de scriere ducând la actualizarea intrării în tabela de pagini a procesului; intrările în tabelele de pagini ale celorlalte procese nu se modifică
principiul copy-on-write (acronimul COW) este întâlnit şi în virtualizare (snapshot-restore, migrarea maşinilor virtuale) şi în sisteme de fişiere (reţinerea versiunilor anterioare)

Swapping

memoria fizică fiind limitată există şansa să fie nevoie să alocăm o pagină fizică dar să nu fie disponibilă
o situaţie este să fie terminat un proces pentru a elibera spaţiul fizic folosind de acesta (out-of-memory handler, numit şi OOM)
soluţia folosită în sistemele de operare moderne este folosirea spaţiului de swap
spaţiul de swap este o zonă din memoria secundară (disc, zonă persistentă) folosită ca suport de stocare a paginilor
atunci când nu există o pagină fizică disponibilă, se alege o pagină fizică şi se evacuează în spaţiul de swap: swap out
pagina eliberată e marcată nevalidă în tabela de pagini şi "validă-swapped out" în informaţiile procesului
pagina fizică este acum mapată în spaţiul virtual de adrese al procesului care a avut nevoie de ea; dacă este o pagină nouă, ideal se umple cu zero-uri, pentru a preveni information leak din primul proces
+ diagramă cu swap out
atunci când se accesează o pagină swapată, MMU generează page fault (este marcată nevalidă în tabela de pagini)
se găseşte o pagină liberă (potenţial se face swap out la o pagină) şi se face swap in (se aduce pagina necesară din swap)
+ diagramă cu swap in
spaţiul de swap e o partiţie dedicată (pe Linux) sau o zonă dintr-o partiţie dată (pe Windows)
dacă există o utilizare intensă a memoriei fizice apar operaţii de swap out/swap in dese
atunci când apar des operaţii de swap out/swap in, paginile sunt înlocuite foarte des şi se consumă timp în schimbarea lor
numim acest fenomen thrashing, este prezent şi la folosirea memoriei cache

Algoritmi de înlocuire de pagini

atunci când o pagină trebuie evacuată pe spaţiul de swap (swap out), trebuie găsită o pagina potrivită
alegerea paginii de evacuat ţine, în general, cont de cât de recent a fost modificată şi cât de recent a fost folosită
paginile au în general un bit (dirty) care spune că a fost modificată
se preferă paginile care au fost cel mai puţin recent utilizate (least recently used)

paginile care nu au fost modificate şi sunt deja pe swap nu trebuie să fie swapped out la înlocuire; conţinutul lor este deja acolo
thrashing

Maparea fişierelor

pentru a simplifica lucrul cu fişierele, acestea pot fi mapate în spaţiul de adresă al unui proces
adică scrierea într-o pagină virtuală conduce la scrierea în blocul corespunzător de pe disc al procesului
are loc uzual pentru fişiere executabile şi biblioteci partajate: sunt mapate în spaţiul virtual al proceselor
+ demo cu pmap care arată numele fişierelor
+ demo cu lsof şi vizualizarea zonelor de tip "txt"
operaţiile cu fişierele sunt acum operaţii cu memoria, nu mai sunt apeluri de sistem read/write
avantaje: overhead scăzut temporal (nu se fac apeluri de sistem) şi spaţial (nu se alocă buffere în user space pentru apelurile read/write)
dezavataj: fişierele trebuie să aibă dimensiunea ştiută pentru mapare, nu se poate creşte dimensiunea (cum se întâmplă atunci când folosim write() pentru a scrie dincolo de dimensiunea fişierului)

Sumar

spaţiul virtual de adrese al procesului e compus din zone statice şi dinamice, zone read-only şi read-write
în general, paginile fizice nu se alocă la comanda de alocare a utilizatorului ci la primul access: demand paging
tabela de pagini are marcată intrarea nevalidă, urmând ca informaţiile sistemului de operare să ştie că e vorba de acces nevalid sau demand paging
se fac în ordine: rezervarea unei pagini virtuale, alocarea unei pagini fizice, maparea paginii fizice la pagina virtuală
un acces la o pagină marcată nevalidă face ca MMU să transmită o excepţie de acces la pagină numită page fault, care apelează un page fault handler înregistrat de sistemul de operare
zonele read-only sunt partajate între procese
iniţial la fork() procesul părinte şi procesul copil partajează toate paginile fizice
imediat după fork(), paginile fizice sunt marcate read-only; la primul acces are loc copy-on-write: duplicarea paginii şi marcarea acesteia read-write în procesul ce a generat acţiunea de scriere
pentru a preveni epuizarea spaţiului fizic de memorie, se extinde prin folosire spaţiului de swap
swap out: evacuarea unei pagini fizice pe swap
swap in: readucerea unei pagini din swap în memoria fizică

dacă au loc operaţii de swap in şi swap out pe aceeaşi pagină, sistemul devine ineficient, apare fenomenul de thrashing
fişierele pot fi mapate în spaţiul virtual de adrese al procesului pentru eficienţă temporală şi spaţială: este cazul fişierelor executabile şi al bibliotecilor partajate