V.5.3. Memoria virtuală

Ideea de pornire

Problema

- aplicaţiile - consum mare de memorie
- memoria disponibilă - insuficientă

Cum se poate rezolva?

- capacitatea discului hard - foarte mare
- nu toate zonele de memorie ocupate sunt accesate la un moment dat

Memoria virtuală

Soluţia - memoria virtuală (*swap*)

- unele zone de memorie - evacuate pe disc
- când este nevoie de ele, sunt aduse înapoi în memorie

Cine gestionează memoria virtuală?

- sunt necesare informaţii globale
- sistemul de operare

Fişierul de paginare

- conţine zonele de memorie evacuate pe disc
- informaţii pentru regăsirea unei zone stocate
  - adresele din memorie
  - programul căruia îi aparţine
  - dimensiunea
  - etc.

Politica de înlocuire (1)

- problema - aceeaşi ca la memoria cache
- aducerea unei zone de memorie din fişierul de paginare implică evacuarea alteia
  - care?
- scop - minimizarea acceselor la disc
- politică ineficientă → număr mare de accese la disc → scăderea vitezei

Politica de înlocuire (2)

- set de lucru (*working set*) - zonele de memorie necesare programului la un moment dat
- uzual mult mai mic decât totalitatea zonelor folosite de program
- dacă încape în memorie - puţine accese la disc

Politica de înlocuire (3)

- se va selecta pentru evacuare zona care nu va fi necesară în viitorul apropiat
- nu se poate şti cu certitudine - estimare
  - pe baza comportării în trecutul apropiat
- paginare la cerere (*demand paging*) - evacuare pe disc numai dacă este strict necesar

Implementare

- prin intermediul mecanismelor de gestiune a memoriei, deja discutate
  - dacă un program încearcă să acceseze o locație aflată temporar pe disc, este necesar același tip de detecție
  - memoria virtuală poate fi folosită împreună atât cu segmentarea, cât şi cu paginarea
- rolul sistemului de întreruperi - sporit

Accesul la memorie (1)

Cazul paginării

1. programul precizează adresa virtuală
2. se determină pagina din care face parte
3. se caută pagina în tabelul de paginare
4. dacă pagina este găsită - salt la pasul 9
5. generare excepţie
6. rutina de tratare caută pagina în fişierul de paginare

Accesul la memorie (2)

Cazul paginării (cont.)

7. dacă pagina nu este în fişierul de paginare - programul este terminat
8. se aduce pagina în memoria fizică
9. se determină cadrul de pagină corespunzător
10. calcul adresă fizică
11. acces la adresa calculată

Reducerea acceselor la disc (1)

- duce la creşterea performanţei
- o pagină este salvată pe disc şi readusă în memorie de mai multe ori
- readucerea în memorie - copia de pe disc nu este ştearsă
- pagina şi copia sa de pe disc sunt identice până la modificarea paginii din memorie

Reducerea acceselor la disc (2)

- evacuarea unei pagini din memorie
  - dacă nu a fost modificată de când se află în memorie - nu mai trebuie salvată
  - util mai ales pentru paginile de cod
- este necesar sprijin hardware pentru detectarea acestei situaţii
  - este suficient să fie detectate operaţiile de scriere

Reducerea acceselor la disc (3)

- tabelul de paginare - structură extinsă
  - fiecare pagină are un bit suplimentar (*dirty bit*)
  - indică dacă pagina a fost modificată de când a fost adusă în memorie
  - resetat la aducerea paginii în memorie
- instrucţiune de scriere în memorie
  - procesorul setează bitul paginii care conţine locaţia modificată

V.5.4. Comunicarea între procese

Comunicare (1)

- pentru a putea coopera, procesele trebuie să-și poată transmite date
  - uneori volume mari
- implementare fizică
  - zone de memorie comune
    - variabile partajate
  - zone de memorie controlate de nucleu
    - structuri de date mai complexe, prevăzute cu metode specifice de acces

Comunicare (2)

- în prima variantă, două sau mai multe procese accesează aceeași zonă de memorie
  - același segment apare simultan în tabelele de descriptori ale mai multor procese
  - același cadru de pagină apare simultan în tabelele de paginare ale mai multor procese
- sistemul de operare controlează zonele comune, indiferent de tehnica folosită
  - iar procesele sunt conștiente de caracterul partajat al acestora

Excludere mutuală (1)

- accesul la o resursă comună poate dura
  - și poate consta în mai multe operaţii
- apare pericolul interferenţelor
- exemplu
  - un proces începe accesul la o variabilă comună
  - înainte de a termina, alt proces începe să o acceseze
  - variabila poate fi modificată în mod incorect

Excludere mutuală (2)

- accesul la o resursă comună - doar în anumite condiţii
- excludere mutuală
  - la un moment dat, un singur proces poate accesa o anumită resursă
- mecanisme de control
  - semafor - cel mai simplu
  - structuri partajate ale căror metode de acces asigură excluderea mutuală (ex. monitor)

Implementare

- accesul la o resursă poate fi controlat (și blocat) doar de către sistemul de operare
- deci orice formă de accesare a unei resurse partajate implică un apel sistem
- dacă se lucrează la nivel jos (variabile partajate + semafoare), este sarcina programatorului să se asigure că apelul este realizat corect

Fire de execuție - comunicare

- în cazul firelor de execuție ale aceluiași proces, variabilele globale sunt automat partajate
  - viteză mai mare
  - crește riscul erorilor de programare
- necesitatea excluderii mutuale este prezentă și aici

V.5.5. Utilizarea MMU

Hardware

Cazul Intel

- segmentarea
  - nu poate fi dezactivată
  - dar poate fi "evitată" prin software
- paginarea
  - poate fi activată/dezactivată

Sistemul de operare

Cazurile Windows, Linux

- segmentarea
  - nu este utilizată în practică
  - toate segmentele sunt dimensionate asfel încât să acopere singure întreaga memorie
- paginarea
  - pagini de 4 Ko
  - Windows poate folosi şi pagini de 4 Mo

Utilitatea MMU (1)

Avantaje

- protecţie la erori
- o aplicaţie nu poate perturba funcţionarea alteia
- verificările se fac în hardware
  - mecanism sigur
  - viteză mai mare

Utilitatea MMU (2)

Dezavantaje

- gestiune complicată
- memorie ocupată cu structurile de date proprii
  - tabelul de descriptori
  - tabelul de paginare
- viteză redusă - dublează numărul acceselor la memorie (sau mai mult)

Utilitatea MMU (3)

Concluzii

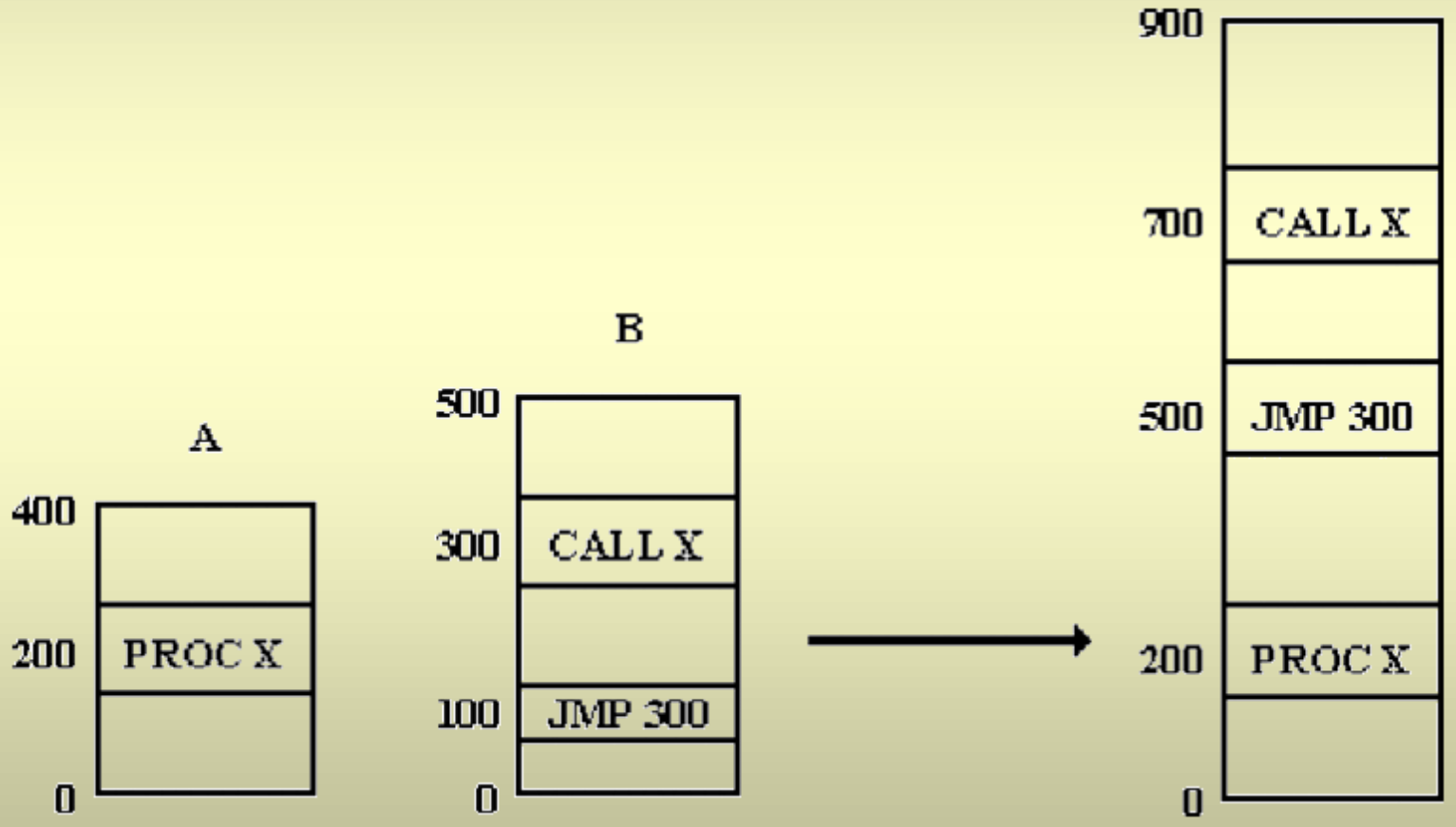
- scăderea de performanţă poate fi compensată folosind cache-uri
- procesoarele de azi oferă suficientă viteză
- sisteme multitasking - risc mare de interferenţe
- mecanismele MMU trebuie folosite

V.6. Crearea şi execuţia programelor

Crearea unui program - faze

- compilarea
  - traducerea comenzilor scrise într-un limbaj sursă în instrucţiuni pentru procesor
- editarea legăturilor (*linking*)
  - tratează aspecte privitoare la gestiunea memoriei într-un program

Crearea unui fişier executabil din mai multe module sursă



Problema relocării

- instrucţiunea de salt - adresa de salt nu mai este corectă
- module compilate independent - fiecare presupune că începe la adresa 0
- afectează şi instrucţiunile care accesează date (adrese de memorie)
- adresele sunt relocate (deplasate) faţă de momentul compilării

Problema referinţelor externe

- funcția X - apelată din alt modul decât cel în care este definită
- la momentul compilării
  - se ştie că este definită în alt modul
  - este imposibil de determinat la ce adresă se va găsi funcția în programul final

Crearea programelor

- se poate scrie un program dintr-un singur modul?
- nu întotdeauna
- programe foarte complexe - modularitate
- biblioteci de funcţii - module separate
  - precompilate
  - codul sursă nu este disponibil

Fazele creării unui program

- compilarea modulelor
  - fişier sursă → fişier obiect
  - fişierele obiect conţin informaţii necesare în faza editării de legături
- editarea legăturilor
  - fişiere obiect → fişier executabil
  - se folosesc informaţiile din fişierele obiect

Structura unui fişier obiect (1)

1. antetul
   - informaţii de identificare
   - informaţii despre celelalte părţi ale fişierului
2. tabela punctelor de intrare
   - conţine numele simbolurilor (variabile şi funcţii) din modulul curent care pot fi apelate din alte module

Structura unui fişier obiect (2)

3. tabela referinţelor externe
   - conţine numele simbolurilor definite în alte module, dar utilizate în modulul curent
4. codul propriu-zis
   - rezultat din compilare
   - singura parte care va apărea în fişierul executabil

Structura unui fişier obiect (3)

5. dicţionarul de relocare
   - conţine informaţii despre localizarea instrucţiunilor din partea de cod care necesită modificarea adreselor cu care lucrează
   - forme de memorare
     - hartă de biţi
     - listă înlănţuită

Editorul de legături (1)

1. construieşte o tabelă cu toate modulele obiect şi dimensiunile acestora
2. pe baza acestei tabele atribuie adrese de start modulelor obiect
   - adresa de start a unui modul = suma dimensiunilor modulelor anterioare

Editorul de legături (2)

3. determină instrucţiunile care realizează accese la memorie şi adună la fiecare adresă o constantă de relocare
   - egală cu adresa de start a modulului din care face parte
4. determină instrucţiunile care apelează funcții sau date din alte module şi inserează adresele corespunzătoare

Execuţia programelor

- la ce adresă începe programul când este încărcat în memorie?
- nu se ştie la momentul când este creat
- toate adresele din program depind de adresa de început
- concluzie: problema relocării apare din nou la lansarea programului în execuţie

Soluţia 1

- Fişierul executabil conţine informaţii de relocare
  - aceste informaţii sunt utilizate de sistemul de operare la încărcarea programului în memorie
  - pentru a actualiza referinţele la memorie
  - exemplu: sistemul de operare DOS

Soluţia 2

- Utilizarea unui registru de relocare
  - încărcat întotdeauna cu valoarea adresei de început a programului curent
  - acces la memorie - la adresa precizată prin instrucţiune se adună valoarea din registrul de relocare
  - dependentă de hardware
    - nu toate procesoarele au registru de relocare

Soluţia 3

- Programele conţin numai referiri la memorie relative la contorul program
  - program independent de poziţie
  - poate fi încărcat în memorie la orice adresă
  - foarte greu de scris
    - instrucţiuni de salt relative - cu restricţii
    - instrucţiuni care lucrează cu adrese de date relative la contorul program - nu există

Soluţia 4

- Paginarea memoriei
  - programul poate fi mutat oriunde în memoria fizică
  - programul crede că începe de la adresa 0, chiar dacă nu este aşa
  - dependentă de suportul hardware (mecanismul de paginare)

Biblioteci partajate (1)

Legare dinamică

- proceduri şi variabile care nu sunt incluse permanent în program
  - numai atunci când este nevoie de ele
- proceduri şi variabile partajate de mai multe programe

Biblioteci partajate (2)

Utilitatea legării dinamice

- proceduri care tratează situaţii excepţionale
  - rar apelate
  - ar ocupa inutil memoria
- proceduri folosite de multe programe
  - o singură copie pe disc
  - o singură instanţă încărcată în memorie

Biblioteci partajate (3)

Tipuri de legare dinamică

- implicită
- explicită

Legare implicită

- foloseşte biblioteci de import
  - legate static în fişierul executabil
  - indică bibliotecile partajate necesare programului
- la lansarea programului
  - sistemul de operare verifică bibliotecile de import
  - încarcă în memorie bibliotecile partajate care lipsesc

Legare explicită (1)

- programul face un apel sistem specific
- cere legarea unei anumite biblioteci partajate
- dacă biblioteca nu există deja în memorie, este încărcată
- legătura cu o bibliotecă partajată poate fi realizată sau distrusă în orice moment

Legare explicită (2)

- exemplu - Windows

```
//legare explicită a unui modul
hLib=LoadLibrary("module");
//se obţine un pointer la o funcţie
fAddr=GetProcAddress(hLib,"func");
(fAddr)(2,3,8);//apel funcţie
FreeLibrary(hLib);//eliberare modul
(fAddr)(2,3,8);//eroare, funcţia nu
  mai este disponibilă
```

Legare explicită (3)

- exemplu - Linux

```
//legare explicită a unui modul
hLib=dlopen("module",RTLD_LAZY);
//se obţine un pointer la o funcţie
fAddr=dlsym(hLib,"func");
(fAddr)(2,3,8);//apel funcţie
dlclose(hLib);//eliberare modul
(fAddr)(2,3,8);//eroare, funcţia nu
  mai este disponibilă
```