

VII.5.3. Memoria virtuală

Memoria virtuală

- extensie a mecanismelor de gestiune a memoriei
- poate fi folosită împreună atât cu segmentarea, cât și 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 excepție
6. rutina de tratare caută pagina în fișierul de paginare

Accesul la memorie (2)

Cazul paginării (cont.)

7. dacă pagina nu este în fișierul 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 creșterea performanței
- o pagină este salvată pe disc și readusă în memorie de mai multe ori
- readucerea în memorie - copia de pe disc nu este ștearsă
- pagina și 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 situații
 - este suficient să fie detectate operațiile 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
- instrucțiune de scriere în memorie
 - procesorul setează bitul paginii care conține locația modificată

VII.5.4. 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 astfel încât să acopere singure întreaga memorie
- paginarea
 - pagini de 4 Ko
 - Windows poate folosi și pagini de 4 Mo

Utilitatea MMU (1)

Avantaje

- protecție la erori
- o aplicație nu poate perturba funcționarea 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

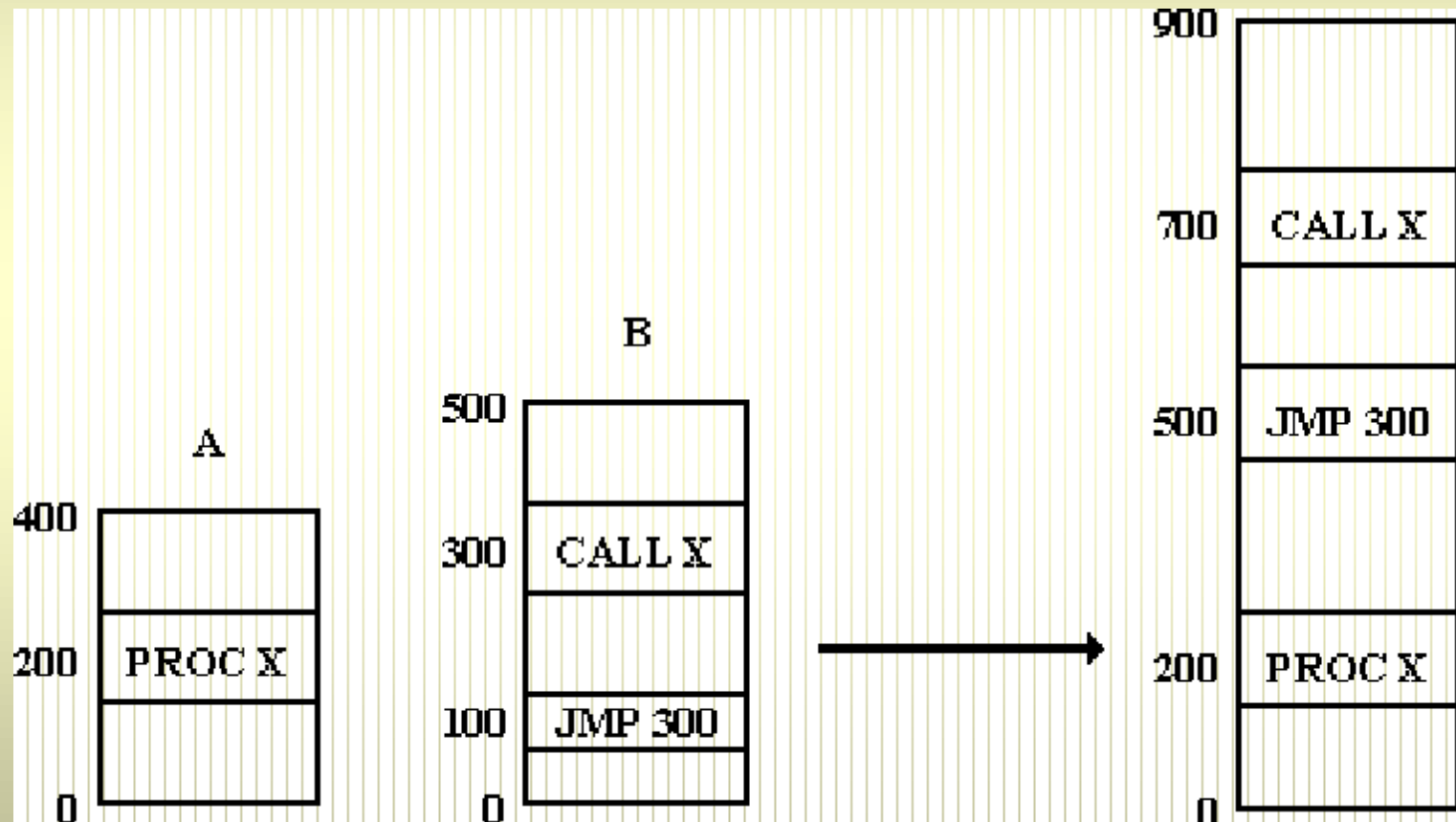
- scăderea de performanță poate fi compensată folosind cache-uri
- procesoarele de azi oferă suficientă viteză
- sisteme multitasking - risc mare de interferențe
- mecanismele MMU trebuie folosite

VII.6. Crearea și execuția programelor

Crearea unui program - faze

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

Crearea unui fișier executabil din mai multe module sursă



Problema relocării

- instrucțiunea de salt - adresa de salt nu mai este corectă
- module compilate independent - fiecare presupune că începe la adresa 0
- afectează și instrucțiunile care accesează date (adrese de memorie)
- adresele sunt relocate (deplasate) față de momentul compilării

Problema referințelor externe

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

Crearea programelor

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

Fazele creării unui program

- compilarea modulelor
 - fișier sursă → fișier obiect
 - fișierele obiect conțin informații necesare în faza editării de legături
- editarea legăturilor
 - fișiere obiect → fișier executabil
 - se folosesc informațiile din fișierele obiect

Structura unui fișier obiect (1)

1. antetul

- informații de identificare
- informații despre celelalte părți ale fișierului

2. tabela punctelor de intrare

- conține numele simbolurilor (variabile și proceduri) din modulul curent care pot fi apelate din alte module

Structura unui fișier obiect (2)

3. tabela referințelor externe

- conține numele simbolurilor definite în alte module, dar utilizate în modulul curent

4. codul propriu-zis

- rezultat din compilare
- singura parte care va apărea în fișierul executabil

Structura unui fișier obiect (3)

5. dicționarul de relocare

- conține informații despre localizarea instrucțiunilor din partea de cod care necesită modificarea adreselor cu care lucrează
- forme de memorare
 - hartă de biți
 - listă înlănțuită

Editorul de legături (1)

1. construiește o tabelă cu toate modulele obiect și 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 instrucțiunile care realizează accese la memorie și adună la fiecare adresă o constantă de relocare
 - egală cu adresa de start a modulului din care face parte
4. determină instrucțiunile care apelează proceduri sau date din alte module și inserează adresele corespunzătoare

Execuția programelor

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

Soluția 1

- Fișierul executabil conține informații de relocare
 - aceste informații sunt utilizate de sistemul de operare la încărcarea programului în memorie
 - pentru a actualiza referințele la memorie
 - exemplu: sistemul de operare DOS

Soluția 2

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

Soluția 3

- Programele conțin numai referiri la memorie relative la contorul program
 - program independent de poziție
 - poate fi încărcat în memorie la orice adresă
 - foarte greu de scris
 - instrucțiuni de salt relative - cu restricții
 - instrucțiuni care lucrează cu adrese de date relative la contorul program - nu există

Soluția 4

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

Biblioteci partajate (1)

Legare dinamică

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

Biblioteci partajate (2)

Utilitatea legării dinamice

- proceduri care tratează situații excepționale
 - rar apelate
 - ar ocupa inutil memoria
- proceduri folosite de multe programe
 - o singură copie pe disc
 - o singură instanță încărcată în memorie

Biblioteci partajate (3)

Tipuri de legare dinamică

- implicită
- explicită

Legare implicită

- folosește biblioteci de import
 - legate static în fișierul 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 bibliotecă 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 obține un pointer la o funcție  
fAddr=GetProcAddress(hLib,"func");  
(fAddr)(2,3,8); //apel funcție  
FreeLibrary(hLib); //eliberare modul  
(fAddr)(2,3,8); //eroare, funcția nu  
mai este disponibilă
```

Legare explicită (3)

- **exemplu - Linux**

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