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
  - structuri de date mai complexe, prevăzute cu metode specifice de acces

Comunicare (2)

- este necesar ca două sau mai multe procese să acceseze aceeași zonă de memorie
  - același segment să apară simultan în tabelele de descriptori ale mai multor procese
  - același cadru de pagină să apară simultan în tabelele de paginare ale mai multor procese
- în oricare caz, sistemul de operare controlează zonele comune
  - 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 KB
  - Windows poate folosi şi pagini de 4 MB

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

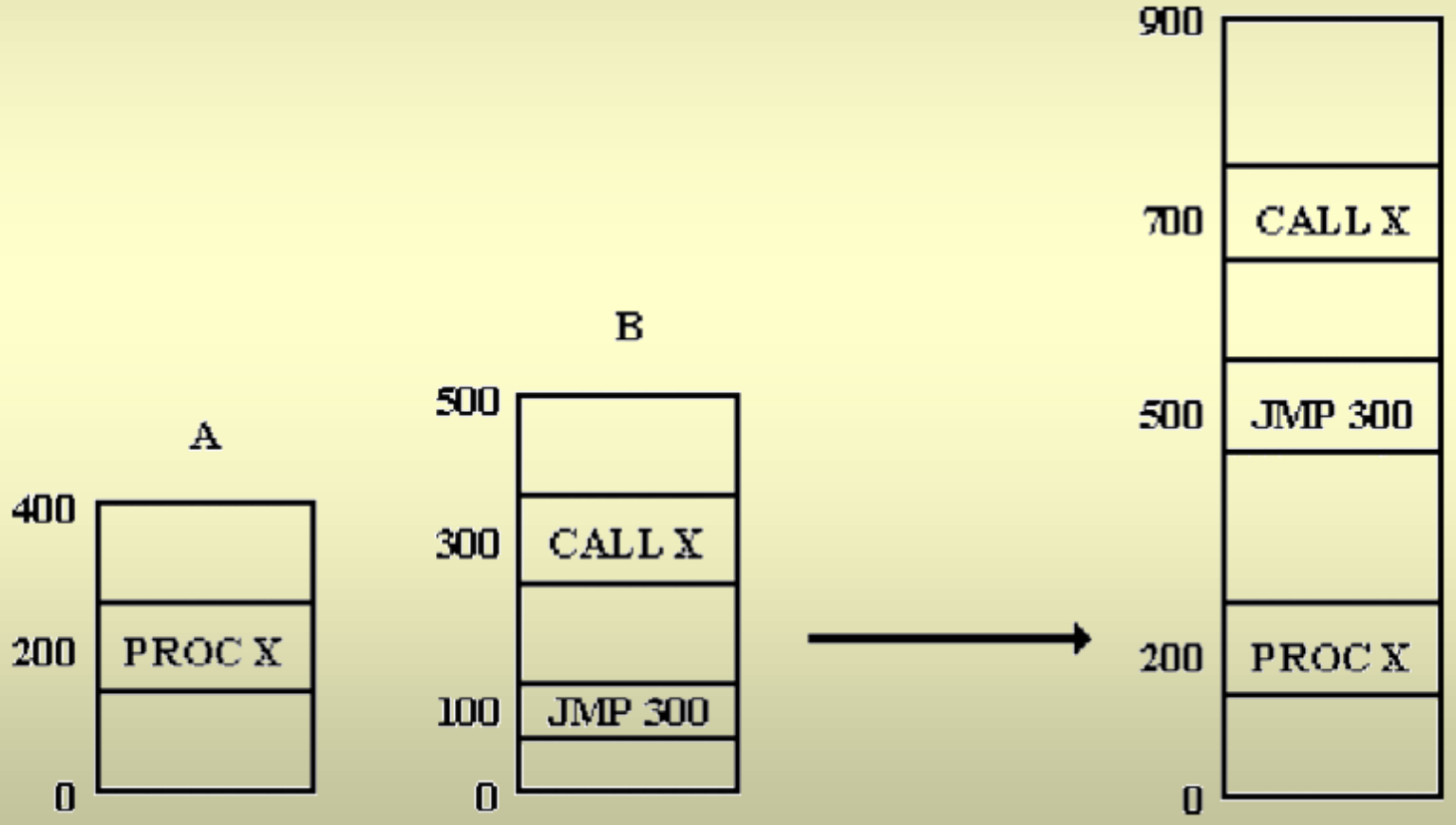
- scăderea de performanţă poate fi compensată folosind cache-uri
- procesoarele de azi oferă suficientă viteză
- sisteme multitasking - risc mare de interferenţe
- mecanismele MMU trebuie folosite

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
```