# Coerenţa memoriei

- trebuie ca toate procesoarele să folosească ultima valoare scrisă pentru o variabilă partajată
- problema - memoriile cache
- scopul - orice variabilă partajată să aibă aceeaşi valoare
  - în toate cache-urile
  - în memoria principală

# Coerenţa - cache *write-back*

`x` - variabilă partajată

| Procesor | Acţiune | Valoare cache A | Valoare cache B | Valoare memorie |
|---|---|---|---|---|
| | | | | 9 |
| A | `i=x;` | 9 | | 9 |
| B | `j=x;` | 9 | 9 | 9 |
| A | `x=5;` | 5 | 9 | 9 |
| B | `k=x;` | 5 | 9 | 9 |

# Coerenţa - cache *write-through*

`x` - variabilă partajată

| Procesor | Acţiune | Valoare cache A | Valoare cache B | Valoare memorie |
|---|---|---|---|---|
| | | | | 9 |
| A | `i=x;` | 9 | | 9 |
| B | `j=x;` | 9 | 9 | 9 |
| A | `x=5;` | 5 | 9 | 5 |
| B | `k=x;` | 5 | 9 | 5 |

# Ce înseamnă coerenţa? (1)

1. ordinea execuţiei
   a) procesorul P scrie în variabila X
   b) apoi procesorul P citeşte X
   – între a) şi b) nu există alte scrieri în X

   → citirea b) returnează valoarea scrisă de a)

# Ce înseamnă coerenţa? (2)

2. viziune coerentă a memoriei
   a) procesorul P scrie în variabila X
   b) apoi procesorul Q (Q≠P) citeşte X
   – între a) şi b) nu există alte scrieri în X
   – între a) şi b) a trecut suficient timp

   → citirea b) returnează valoarea scrisă de a)

# Ce înseamnă coerenţa? (3)

3. serializarea scrierilor
   a) procesorul P scrie în variabila X
   b) procesorul Q (Q=P sau Q≠P) scrie în variabila X

→ toate procesoarele văd cele două scrieri în aceeaşi ordine

– nu neapărat a) înaintea b)

# Menţinerea coerenţei cache-urilor

- protocoale de menţinere a coerenţei
- se bazează pe informaţiile despre liniile de cache
  - *invalid* - datele nu sunt valide
  - *dirty* - doar cache-ul curent deţine valoarea actualizată
  - *shared* - cache-ul curent deţine valoarea actualizată, la fel memoria principală şi eventual alte cache-uri

# Tipuri de protocoale

- *directory based*
  - informaţiile despre fiecare linie de cache - ţinute într-un singur loc
- *snooping*
  - fiecare cache are o copie a liniei partajate
  - fără centralizarea informaţiei
  - cache-urile monitorizează magistrala
    - detectează schimbările produse în liniile de cache

# Actualizarea cache-urilor

- fiecare cache anunţă modificările făcute
- celelalte cache-uri reacţionează
- contează doar operaţiile de scriere
- variante
  - scriere cu invalidare (*write invalidate*)
  - scriere cu actualizare (*write update*)

# Scriere cu invalidare (1)

- un procesor modifică o dată
- modificarea se face în cache-ul propriu
  - toate celelalte cache-uri sunt notificate
- celelalte cache-uri
  - nu au o copie a datei modificate - nici o acţiune
  - au o copie a datei modificate - îşi invalidează linia corespunzătoare
  - valoarea corectă va fi preluată când va fi nevoie

# Scriere cu invalidare (2)

`x` - variabilă partajată

| Proc. | Acţiune | Reacţie cache | Cache A | Cache B | Memorie |
|---|---|---|---|---|---|
| | | | | | 9 |
| A | `i=x;` | read miss | 9 | | 9 |
| B | `j=x;` | read miss | 9 | 9 | 9 |
| A | `x=5;` | invalidation | 5 | inv. | 9 |
| B | `k=x;` | read miss | 5 | 5 | 5 |

# Scriere cu actualizare (1)

- un procesor modifică o dată
- modificarea se face în cache-ul propriu
  - toate celelalte cache-uri sunt notificate
  - se transmite noua valoare
- celelalte cache-uri
  - nu au o copie a datei modificate - nici o acţiune
  - au o copie a datei modificate - preiau noua valoare

# Scriere cu actualizare (2)

`x` - variabilă partajată

| Proc. | Acţiune | Reacţie cache | Cache A | Cache B | Memorie |
|---|---|---|---|---|---|
| | | | | | 9 |
| A | `i=x;` | read miss | 9 | | 9 |
| B | `j=x;` | read miss | 9 | 9 | 9 |
| A | `x=5;` | invalidation | 5 | 5 | 5 |
| B | `k=x;` | read hit | 5 | 5 | 5 |

# Invalidare vs. actualizare (1)

- mai multe scrieri succesive în aceeaşi locaţie
  - *write invalidation* - o singură invalidare (prima dată)
  - *write update* - câte o actualizare pentru fiecare scriere
  - mai avantajos - invalidare

# Invalidare vs. actualizare (2)

- mai multe scrieri în aceeaşi linie de cache
  - modificarea unei locaţii necesită invalidarea/actualizarea întregii linii
  - *write invalidation* - o singură invalidare (prima dată)
  - *write update* - câte o actualizare pentru fiecare scriere
  - mai avantajos - invalidare

# Invalidare vs. actualizare (3)

- "timpul de răspuns"
  - timpul între modificarea unei valori la un procesor şi citirea noii valori la alt procesor
  - *write invalidation* - întâi invalidare, apoi citire (când este necesar)
  - *write update* - actualizare imediată
  - mai avantajos - actualizare

# Invalidare vs. actualizare (4)

- ambele variante au avantaje şi dezavantaje
- scrierea cu invalidare - ocupare (mult) mai redusă a memoriei şi magistralelor
- scrierea cu actualizare - rata de succes a cache-urilor mai mare
- mai des folosită - invalidarea

# V. Dispozitivele periferice

# Magistrale (1)

- căi de comunicaţie a informaţiei
- o magistrală leagă între ele mai mult de 2 componente printr-o cale unică
- descrierea unei magistrale
  - semnalele electrice folosite
  - reguli de comunicare - trebuie respectate de toate părţile implicate
  - mod de conectare

# Magistrale (2)

Accesul la magistrală

- mai multe entităţi pot solicita simultan accesul
- este necesară o procedură de arbitraj
  - decide cine va primi accesul
  - celelalte trebuie să aştepte eliberarea magistralei

# Arbitrarea magistralelor

Tipuri de arbitrare a magistralelor

- centralizat
  - decizia o ia un circuit specializat (arbitru)
- descentralizat
  - componentele se înţeleg între ele
  - pe baza regulilor care definesc funcţionarea magistralei

# Conectarea la magistrală

- probleme de natură electrică
- mai multe circuite conectate împreună
  - în intrare şi ieşire
- nu se pot conecta mai multe ieşiri
  - nivelele diferite de tensiune ar distruge circuitele
- o soluţie - multiplexarea
  - toate ieşirile sunt conectate la un multiplexor

# Circuite *tri-state*

- ieşirea are 3 stări posibile
  - 0
  - 1
  - impedanţă infinită (*High-Z*)
- primele două corespund valorilor obişnuite
- a treia implică decuplarea de pe magistrală
  - ca şi cum ieşirea circuitului nici nu ar fi conectată la magistrală

# Circuite *open-collector*

- în unele cazuri se numesc *open-drain*
  - în funcţie de tehnologia utilizată
- este posibilă conectarea mai multor ieşiri simultan
- valoarea rezultată - funcţia AND între ieşirile conectate

# Magistrale - privire generală

Avantaje

- activitatea pe magistrală - uşor de controlat
- economic - structură relativ simplă

Dezavantaj

- performanţe mai scăzute
  - doar 2 componente pot comunica la un moment dat

# VI. Sistemul de întreruperi

# Ce este o întrerupere?

- procesorul poate suspenda execuţia programului curent
- scop - tratarea unor situaţii neprevăzute
- după tratare se reia programul întrerupt
- scopul iniţial - comunicarea cu perifericele
- procesorul nu "aşteaptă" perifericele
  - acestea îl solicită când este necesar

# Întreruperi hardware

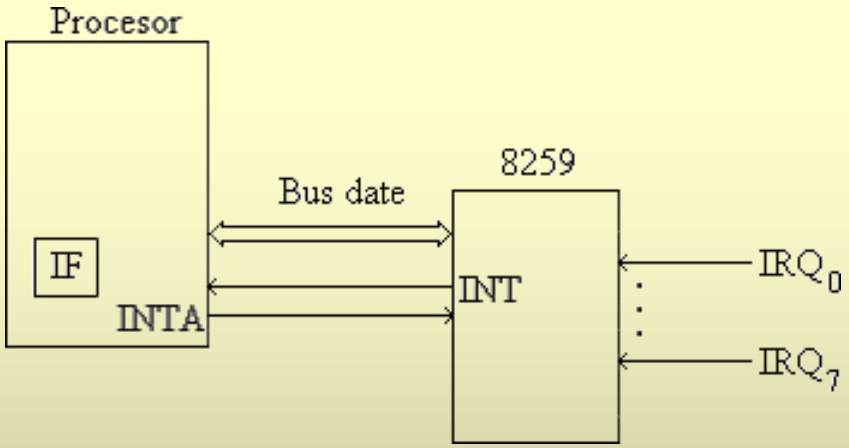
- dezactivabile (*maskable*)
  - procesorul poate refuza tratarea lor
  - depinde de valoarea bistabilului IF (*Interrupt Flag*): 1 - acceptare, 0 - refuz
  - IF poate fi modificat prin program
- nedezactivabile (*non-maskable*)
  - procesorul le tratează întotdeauna

# Controllerul de întreruperi (1)

- circuit specializat
- preia cererile de întrerupere de la periferice
- le trimite procesorului
- rezolvă conflictele (mai multe cereri simultane) - arbitrare
  - fiecare periferic are o anumită prioritate

# Controllerul de întreruperi (2)

- iniţial - Intel 8259
  - se pot folosi mai multe (cascadare)




- astăzi - integrat în chipset

# Tratarea întreruperilor - etape (1)

- dispozitivul periferic generează o cerere de întrerupere pe linia sa $IRQ_i$
- controllerul trimite un semnal pe linia INT
- procesorul verifică valoarea bistabilului IF
  - numai pentru întreruperi dezactivabile
  - dacă este 0 - refuză cererea; stop
  - dacă este 1 - răspunde cu un semnal INTA

# Tratarea întreruperilor - etape (2)

- se întrerupe execuţia programului curent
- se salvează în stivă regiştrii (inclusiv PC)
- se şterge bistabilul IF
  - se blochează execuţia altei întreruperi în timpul execuţiei programului pentru întreruperea în curs
  - poate fi repus pe 1 prin program

# Tratarea întreruperilor - etape (3)

- identificare periferic (sursa cererii)
  - controllerul depune pe magistrala de date un octet *type*
  - indică perifericul care a făcut cererea
  - maximum $2^8$=256 surse de întrerupere
  - fiecare sursă are rutina proprie de tratare
  - periferice diferite - tratări diferite

# Tratarea întreruperilor - etape (4)

- determinarea adresei rutinei de tratare
  - adresa fizică 0 - tabelul vectorilor de întreruperi
  - conţine adresele tuturor rutinelor de tratare
  - dimensiune: 256 adrese × 4 octeţi = 1 Ko
  - octet type = n → adresa rutinei de tratare: la adresa n × 4

# Tratarea întreruperilor - etape (5)

- salt la rutina de tratare a întreruperii
- execuţia rutinei de tratare
- revenire în programul întrerupt
  - restabilire valoare bistabil IF
  - restabilire valoare regiştri (din stivă)
  - reluarea execuţiei programului de unde a fost întrerupt

# Extindere

- sistem puternic şi flexibil
- poate fi extins - utilizare mai largă
- programele trebuie întrerupte şi în alte situaţii (nu doar pentru comunicarea cu perifericele)
- util mai ales pentru sistemul de operare

# Tipuri de întreruperi

- *hardware* - generate de periferice
- *excepţii (traps)* - generate de procesor
  - semnalează o situaţie anormală
  - exemplu: împărţire la 0
- *software* - generate de programe
  - folosite pentru a solicita anumite servicii sistemului de operare

# VII. Sistemul de operare

# Rolul sistemului de operare (1)

- program cu rol de gestiune a sistemului de calcul
- face legătura între hardware şi aplicaţii
- pune la dispoziţia aplicaţiilor diferite servicii
- supraveghează buna funcţionare a aplicaţiilor
  - poate interveni în cazurile de eroare

# Rolul sistemului de operare (2)

- pentru a-şi îndeplini sarcinile, are nevoie de suport hardware
  - cel mai important - sistemul de întreruperi
- componente principale
  - nucleu (*kernel*)
  - drivere

# VII.1. Nucleul sistemului de operare

# Nucleul sistemului de operare

- în mare măsură independent de structura hardware pe care rulează
- "creierul" sistemului de operare
- gestiunea resurselor calculatorului - hardware şi software
  - funcţionare corectă
  - alocare echitabilă între aplicaţii

# Moduri de lucru ale procesorului

- utilizator (*user mode*)
  - restricţionat
  - accesul la memorie - numai anumite zone
  - accesul la periferice - interzis
- nucleu (*kernel mode*)
  - fără restricţii

# Modul de rulare al programelor

- sistemul de operare - în mod nucleu
  - poate efectua orice operaţie
- aplicaţiile - în mod utilizator
  - nu pot realiza anumite acţiuni
  - apelează la nucleu

# Trecerea între cele două moduri

- prin sistemul de întreruperi
- utilizator → nucleu
  - apel întrerupere software
  - generare excepţie (eroare)
- nucleu → utilizator
  - revenire din rutina de tratare a întreruperii

# Consecinţe

- codul unei aplicaţii nu poate rula în modul nucleu
- avantaj: erorile unei aplicaţii nu afectează alte programe
  - aplicaţii
  - sistemul de operare
- dezavantaj: pierdere de performanţă

# Structura nucleului

- nu este un program unitar
- colecţie de rutine care cooperează
- funcţii principale
  - gestiunea proceselor
  - gestiunea memoriei
  - gestiunea sistemelor de fişiere
  - comunicarea între procese

# VII.2. Apeluri sistem

# Apeluri sistem (*system calls*)

- cereri adresate nucleului de către aplicaţii
- acţiuni pe care aplicaţiile nu le pot executa singure
  - numai în modul nucleu al procesorului
  - motiv - siguranţa sistemului
- realizate prin întreruperi software
- similar apelurilor de funcţii

# Etapele unui apel sistem (1)

1. programul depune parametrii apelului sistem într-o anumită zonă de memorie
2. se generează o întrerupere software
   - procesorul trece în modul nucleu
3. se identifică serviciul cerut şi se apelează rutina corespunzătoare

# Etapele unui apel sistem (2)

4. rutina preia parametrii apelului şi îi verifică
   – dacă sunt erori - apelul eşuează
5. dacă nu sunt erori - realizează acţiunea cerută
6. terminarea rutinei - rezultatele obţinute sunt depuse într-o zonă de memorie accesibilă aplicaţiei care a făcut apelul

# Etapele unui apel sistem (3)

7. procesorul revine în mod utilizator
8. se reia execuţia programului din punctul în care a fost întrerupt
   - se utilizează informaţiile memorate în acest scop la apariţia întreruperii
9. programul poate prelua rezultatele apelului din zona în care au fost depuse

# Apeluri sistem - concluzii

- comunicare între aplicaţie şi nucleu
  - depunere parametri
  - preluare rezultate
- acţiunile critice sunt realizate într-un mod sigur
- mari consumatoare de timp
- apeluri cât mai rare - lucru cu buffere

# Cum folosim bufferele

Exemplu - funcţia *printf*

- formatează textul, apoi îl trimite spre ecran
- nu are acces direct la hardware
  - se foloseşte de un apel sistem
  - *write* (în Linux)
- de fapt, *printf* depune textul formatat într-o zonă proprie de memorie (buffer)
  - doar când bufferul e plin se face un apel sistem