Aula 5

Implementação de Exclusão Mútua

Prof. Sérgio Campos / Prof. Marcus Rocha {scampos,mvrocha}@dcc.ufmg.br

Implementando Exclusão Mútua

- Via software
  - Propriedades;
  - Dois processos;
  - Múltiplos processos
- Via hardware
  - Desabilitando interrupções;
  - read-modify-write
  - test & set

Exclusão Mútua

```
while (!fim) {
  seção_não_crítica;
  lock();
  seção_crítica;
  unlock();
}
```

- Uma solução para o problema tem que satisfazer:
  - Exclusão mútua;
  - Progresso - decisão não pode ser evitada;
  - Espera limitada - sem inanição.

Mas não se sabe nada sobre a **velocidade** de execução de cada processo.

Dois Processos: via software

Algoritmo 1: Software

```
lock()
{
  while (vez != i);
};

unlock()
{
  vez = j;
};
```

Dois Processos: via software

Algoritmo 1:

Propriedades:

- Exclusão mútua: OK
- Progresso: Não OK.
  - Protocolo exige alternância estrita.
- Espera limitada: Não OK.

- Problema:
  - Não guarda informação suficiente:
    - Guarda somente quem entrou da última vez.

Dois Processos: via software

Algoritmo 2:

```
lock()
{
  quer_entrar[i] = 1;
  while (quer_entrar[j]);
};

unlock()
{
  quer_entrar[i] = 0;
};
```

Dois Processos: via software

Algoritmo 2:

Propriedades:

- Exclusão mútua: OK
- Progresso: Não OK.
  - Deadlock!
- Espera limitada: Não OK.

Dois Processos: via software

Algoritmo 3:

```
lock() {
  quer_entrar[i] = 1;
  vez = j;
  while
    (quer_entrar[j] && turn == j);
};

unlock() {
  quer_entrar[i] = 0;
};
```

Dois Processos: via software

Algoritmo 3:

Propriedades:

- Exclusão mútua: OK
- Progresso: OK
- Espera limitada: OK

Múltiplos Processos

Algoritmo do padeiro:

- Ao entrar na padaria o cliente recebe um número.
- Quem tiver o número menor é atendido.
- Pode acontecer de dois processos receberem o mesmo número:
  - Neste caso o processo de menor número é atendido primeiro (escolha arbitrária).

Múltiplos Processos

```
lock() {

  numero[i] = max(numero[0..n-1]) + 1;

  for (j = 0; j < n; j++) {

    while ((numero[j] != 0) &&
           (numero[j],j)<(numero[i],i));
  };
}
unlock() {
  numero[i] = 0;
};
```

(a,b) < (c,d) se
(a < c) ou (a = c e b < d)

Múltiplos Processos

```
lock() {
  escolhendo[i] = 1;
  numero[i] = max(numero[0..n-1]) + 1;
  escolhendo[i] = 0;
  for (j = 0; j < n; j++) {
    while (escolhendo[j]);
    while ((numero[j] != 0) &&
           (numero[j],j)<(numero[i],i));
  };
}
unlock() {
  numero[i] = 0;
};
```

(a,b) < (c,d) se
(a < c) ou (a = c e b < d)

Múltiplos Processos

Correto porque:

- Processo sempre escolhe um número maior (ou igual) ao maior existente.
- O processo que consegue tem sempre o menor número (ou menor número de processo).
- Política de escolha FIFO.

Exclusão Mútua via Hardware

Desabilitando Interrupções

Em um uniprocessador operações serão atômicas se não houver troca de contexto.

Trocas de contexto acontecem quando o escalonador é chamado: por eventos internos ou eventos externos:

- Eliminar eventos internos é fácil.
- Eliminar eventos externos pode ser feito desabilitando-se interrupções.

Exclusão Mútua via Hardware

Desabilitando Interrupções

Vantagem: simples e eficiente.

Desvantagens:

- Não funciona em multiprocessadores.
- Se o usuário puder desabilitar interrupções o SO perde controle da CPU.

Exclusão Mútua via Hardware

Desabilitando Interrupções

Mas que fica fácil fica:

```
lock()
{
  disable_interrupts();
};

unlock()
{
  enable_interrupts();
};
```

Exclusão Mútua via Hardware

Uma maneira mais eficiente de se implementar exclusão mútua desabilitando interrupções é:

```
int M = 0;
lock(M)
{
  disable_irq();
  while (M) {
    enable_irq();
    disable_irq();
  };
  M = 1;
  enable_irq();
};
```

```
unlock(M)
{
  disable_irq();
  M = 0;
  enable_irq();
};
```

Read-Modify-Write

Em multiprocessadores desabilitar interrupções não funciona.

A maioria dos processadores modernos implementa alguma forma de read-modify-write:

- Estas instruções leem um valor da memória, o atualizam e gravam na memória de forma **atômica**, implementadas por hardware.
- Implementação em multiprocessadores é complicada: necessita modificações no protocolo de coerência de cache.

Read-Modify-Write

Exemplos:

- Test & set: mais comum
- Exchange: x86.
- Compare & swap: 68000

Test & Set

Implementando exclusão mútua:

```
int M = 0;
lock(M)
{
  while (test&set(M) == 1);
};

unlock(M)
{
  M = 0;
};
```

Hardware X Software

Se é tão mais simples acessar exclusão mútua via hardware, porque estudar via software ?

- Porque existem processadores que não possuem primitivas para exclusão mútua via hardware! Exemplo: MIPS (DecStation).
- Porque exclusão mútua via hardware só fornece primitivas de muito baixo nível.
- Exclusão mútua via software serve de introdução a problemas mais complexos e importantes.