Exclusão Mútua por Espera-Ocupada

Walter Fetter Lages

w.fetter@ieee.org

Universidade Federal do Rio Grande do Sul

Escola de Engenharia

Departamento de Engenharia Elétrica

ENG04008 Sistemas de Tempo Real

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Introdução

- Dificuldades devido à interação ente processos
  - Raramente processo são independentes
- Sincronização
- Comunicação
  - Variáveis compartilhadas
  - Passagem de mensagens

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Exclusão Mútua

- x=x+3
  - Carregar x no acumulador
  - Somar 3
  - Armazenar x de volta
- Operação não atômica
  - Se mais de um processo estiver executando a operação o resultado depende do interleaving
- Seção crítica
  - Seqüência de comandos que deve ser executada de forma indivisível do ponto de vista dos demais processos

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Seção Crítica

- Pré-protocolo
- Seção crítica
- Pós-protocolo
- A falha de um processo fora da seção crítica não deve afetar os demais processos
- Não são feitas hipóteses sobre o tempo de execução dos processos (a não ser que um processo não para na seção crítica)

Espera-ocupada

- Sincronização através flags
- Spin-lock

```
void p1(void)                    void p2(void)
{                                {
     ....                             ....
     while(flag==DOWN);               flag=UP;
     .....                            .....
}                                }
```

Exclusão Mútua (Tentativa 1)

```
void p1 (void)
{
    for(;;)
    {
        flag1=UP;
        while(flag2==UP);
        /* seção crítica */
        flag1=DOWN;
        /* seção não crítica */
    }
}
```

Exclusão Mútua (Tentativa 1)

```
void p1 (void)
{
    for(;;)
    {
        flag1=UP;
        while(flag2==UP);
        /* seção crítica */
        flag1=DOWN;
        /* seção não crítica */
    }
}
```

```
void p2 (void)
{
    for(;;)
    {
        flag2=UP;
        while(flag1==UP);
        /* seção crítica */
        flag2=DOWN;
        /* seção não crítica */
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Tentativa 1

- Permite livelock
  - p1 seta a sua flag (UP)
  - p2 seta a sua flag (UP)
  - p2 verifica a sua flag
  - p2 entra na espera-ocupada
  - p1 verifica a sua flag
  - p1 entra na espera-ocupada
- Cada processo anuncia a sua intenção de entrar na seção crítica antes de verificar se é possível faze-lo

Exclusão Mútua (Tentativa 2)

```
void p1 (void)
{
     for(;;)
     {
          while(flag2==UP);
          flag1=UP
          /* seção crítica */
          flag1=DOWN;
          /* seção não crítica */
     }
}
```

Exclusão Mútua (Tentativa 2)

```
void p1 (void)
{
    for(;;)
    {
        while(flag2==UP);
        flag1=UP
        /* seção crítica */
        flag1=DOWN;
        /* seção não crítica */
    }
}
```

```
void p2 (void)
{
    for(;;)
    {
        while(flag1==UP);
        flag2=UP
        /* seção crítica */
        flag2=DOWN;
        /* seção não crítica */
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Tentativa 2

- Não garante a exclusão mútua
  - p1 e p2 estão na sua seção não crítica (flag1=flag2=DOWN)
  - p1 verifica a flag2
  - p2 verifica a flag1
  - p2 seta a sua flag (flag2=UP)
  - p2 entra na seção crítica
  - p1 seta a sua flag (flag1=UP)
  - p1 entra na seção crítica

Exclusão Mútua (Tentativa 3)

```
void p1(void)
{
    for(;;)
    {
        while(turn==2);
        /* seção crítica */
        turn=2;
        /* seção não crítica */
    }
}
```

Exclusão Mútua (Tentativa 3)

```
void p1(void)
{
    for(;;)
    {
        while(turn==2);
        /* seção crítica */
        turn=2;
        /* seção não crítica */
    }
}
```

```
void p2(void)
{
    for(;;)
    {
        while(turn==1);
        /* seção crítica */
        turn=1;
        /* seção não crítica */
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Tentativa 3

- Garante a exclusão mútua
- Livelock não é possível
- Se p1 falhar for a da seção crítica turn receberá o valor 1 e ficará com este valor, impedindo p2 de entrar na seção crítica
- Os dois processos entram, obrigatoriamente, na seção crítica o mesmo número de vezes

Exclusão Mútua (Tentativa 4)

```
void p1(void)
{
    for(;;)
    {
        c1=0;
        while(c2==0)
        {
            c1=1;
            /* não faz nada */
            c1=0;
        }
        /* seção crítica */
        c1=1;
    }
}
```

Exclusão Mútua (Tentativa 4)

```
void p1(void)
{
    for(;;)
    {
        c1=0;
        while(c2==0)
        {
            c1=1;
            /* não faz nada */
            c1=0;
        }
        /* seção crítica */
        c1=1;
        /* seção não crítica */
    }
}
```

```
void p2(void)
{
    for(;;)
    {
        c2=0;
        while(c1==0)
        {
            c2=1;
            /* não faz nada */
            c2=0;
        }
        /* seção crítica */
        c2=1;
        /* seção não crítica */
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Tentativa 4

- Cada processo dá uma chance para o outro entrar na seção crítica
- Lockout
  - p1 faz c1=0
  - p2 faz c2=0
  - p1 verifica c2
  - p2 verifica c1
  - p1 faz c1=1
  - p2 faz c2=1
  - p1 faz c1=0
  - p2 faz c2=0

Algoritmo de Dekker

```
void p1(void)
{
    for(;;)
    {
        c1=0;
        while(c2==0)
        {
            if(turn==2)
            {
                c1=1;
                while(turn==2);
                c1=0;
            }
        }
        /* seção crítica */
        turn=2;
        c1=1;
        /* seção não crítica */
    }
}
```

```
void p2(void)
{
    for(;;)
    {
        c2=0;
        while(c1==0)
        {
            if(turn==1)
            {
                c2=1;
                while(turn==1);
                c2=0;
            }
        }
        /* seção crítica */
        turn=1;
        c2=1;
        /* seção não crítica */
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Algoritmo de Dekker

- Difícil generalização para n processos
- Possível, mas interesse puramente acadêmico
- while significa desperdício de tempo da CPU
- A exclusão mútua entre n processos exige n+1 variáveis

Algoritmo de Peterson

```
void p1(void)
{
     for(;;)
     {
          flag1=UP;
          turn=2;
          while(flag2==UP && turn==2);
          /* seção crítica */
          flag1=DOWN;
          /* seção não crítica */
     }
}
```

Algoritmo de Peterson

```
void p1(void)
{
    for(;;)
    {
        flag1=UP;
        turn=2;
        while(flag2==UP && turn==2);
        /* seção crítica */
        flag1=DOWN;
        /* seção não crítica */
    }
}
```

```
void p2(void)
{
    for(;;)
    {
        flag2=UP;
        turn=1;
        while(flag1==UP && turn==1
        /* seção crítica */
        flag2=DOWN;
        /* seção não crítica */
    }
}
```

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Algoritmo de Peterson

- Simplificação do algoritmo de Dekker
- Generalização para n processo fácil
- Necessita cuidado na implementação
- Teste duplo no while deve ser atômico

UFRGS
UNIVERSIDADE FEDERAL
DO RIO GRANDE DO SUL

Espera-Ocupada

- Protocolos difíceis de projetar, entender e provar que estão corretos
- Teste de programas podem não considerar todos os interleavings possíveis e que podem violar a exclusão mútua ou levar a livelock, deadlock ou lockout
- Espera-ocupada é ineficiente
- Um processo que não use corretamente as variáveis podem corromper todo o sistema