

MC514–Sistemas Operacionais: Teoria e Prática
1s2010

Processos e Threads 4

Objetivos

- Revisão de problemas de exclusão mútua
- Tentativas de algoritmos para N threads
- Algoritmo de Dijkstra
- Algoritmos de Hyman e Peterson

Exclusão mútua

- Devemos garantir: exclusão mútua, ausência de deadlock e ausência de starvation

```
volatile int s; /* Variável compartilhada */
while (1) {
    /* Região não crítica */
    /* Protocolo de entrada */
    /* Região crítica */
    s = thr_id;
    printf ("Thr %d: %d", thr_id, s);
    /* Protocolo de saída */
}
```

Abordagem da Alternância

```
int s = 0;  
int vez = 1; /* Primeiro a thread 1 */
```

Thread 0

```
while (true)  
    while (vez != 0);  
    s = 0;  
    print ("Thr 0:" , s);  
    vez = 1;
```

Thread 1

```
while (true)  
    while (vez != 1);  
    s = 1;  
    print ("Thr 1:" , s);  
    vez = 0;
```

- Veja o código: `alternancia.c`

Abordagem da Alternância

N threads

Thread_i:

```
while (true)
    while (vez != i);
    s = i;
    print ("Thr ", i, ": ", s);
    vez = (i + 1) % N;
```

- Veja o código: `alternanciaN.c`

Vetor de Interesse

```
int s = 0;  
int interesse[2] = {false, false};
```

Thread 0

```
while (true)  
    interesse[0] = true;  
while (interesse[1]);  
s = 0;  
print("Thr 0:" , s);  
interesse[0] = false;
```

Thread 1

```
while (true)  
    interesse[1] = true;  
while (interesse[0]);  
s = 1;  
print("Thr 1:" , s);  
interesse[1] = false;
```

- Veja o código: interesse.c

Interesse para N threads

```
int interesse[N] = {false, ..., false}
while (true) { /* Código da Thread_i */
    interesse[i] = true;
    while (existe j!=i tal que (interesse[j]));
    s = i;
    print ("Thr ", i, ": ", s);
    interesse[i] = false;
```

- Veja o código: interesseN.c

Vetor de Interesse e Alternância

```
int s = 0, vez = 0;  
int interesse[2] = {false, false};
```

Thread 0

```
while (true)  
    interesse[0] = true;  
    if (interesse[1])  
        while (vez != 0);  
    s = 0;  
    print("Thr 0:", s);  
    vez = 1;  
    interesse[0] = false;
```

Thread 1

```
while (true)  
    interesse[1] = true;  
    if (interesse[0])  
        while (vez != 1);  
    s = 1;  
    print("Thr 1:", s);  
    vez = 0;  
    interesse[1] = false;
```

- Veja o código: interesse_vez.c

Interesse e vez para N threads

```
int interesse[N] = {false, ..., false}
int vez = 0;
while (true) { /* Código da Thread_i */
    interesse[i] = true;
    if (existe j!=i tal que (interesse[j]))
        while (vez != i);
    s = i;
    print ("Thr ", i, ": ", s);
    vez = (i + 1) % N;
    interesse[i] = false;
```

- Veja o código: interesse_vezN.c

Quase o algoritmo de Dekker

```
int s = 0, vez = 0, interesse[2] = {false, false};
```

Thread 0

```
while (true)
    interesse[0] = true;
    while (interesse[1])
        interesse[0] = false;
        while (vez != 0);
        interesse[0] = true;
s = 0;
print ("Thr 0:" , s);
vez = 1;
interesse[0] = false;
```

Thread 1

```
while (true)
    interesse[1] = true;
    while(interesse[0])
        interesse[1] = false;
        while(vez != 1);
        interesse[1] = true;
s = 1;
print ("Thr 1:" , s);
vez = 0;
interesse[1] = false;
```

Algoritmo de Dekker (1965)

```
int s = 0, vez = 0, interesse[2] = {false, false};
```

Thread 0

```
while (true)
    interesse[0] = true;
    while (interesse[1])
        if (vez != 0)
            interesse[0] = false;
            while (vez != 0);
            interesse[0] = true;
    s = 0;
    print ("Thr 0:" , s);
    vez = 1;
    interesse[0] = false;
```

Thread 1

```
while (true)
    interesse[1] = true;
    while(interesse[0])
        if (vez != 1)
            interesse[1] = false;
            while(vez != 1);
            interesse[1] = true;
    s = 1;
    print ("Thr 1:" , s);
    vez = 0;
    interesse[1] = false;
```

Sugestão para N threads

```
int vez = 0, interesse = {false, ..., false}
while (true) { /* Código da Thread_i */
    interesse[i] = true;
    while (existe j!=i tal que (interesse[j]))
        if (vez != i)
            interesse[i] = false;
            while (vez != i) ;
            interesse[i] = true;
    s = i;
    print ("Thr ", i, ": ", s);
    vez = (i+1) % N;
    interesse[i] = false;
```

Sugestão para N threads

Garante exclusão mútua?

- Uma thread só entra na região crítica após percorrer o vetor e verificar que nenhuma outra está interessada.

Garante ausência de deadlock?

- Se todas estiverem interessadas, pelo menos uma thread (a da vez) consegue entrar na região crítica

Garante progresso?

- Não. A vez pode ser passada para uma thread desinteressada.
- Veja o código `dekkerN.c`

Outra sugestão...

```
int vez = -1, interesse = {false, ..., false}
while (true) { /* Código da Thread_i */
    interesse[i] = true;
    while (existe j!=i tal que (interesse[j]))
        if (vez != -1 && vez != i)
            interesse[i] = false;
        while (vez == -1 || vez != i) ;
        interesse[i] = true;
```

Outra sugestão... (continuação)

```
s = i;  
print ("Thr ", i, ": ", s);  
vez = alguma interessada ou -1;  
interesse[i] = false;
```

Por que não funciona?

- Porque mais de uma thread pode achar que é a vez dela ao encontrar `vez == -1`
- Veja o código: `outro_dekkerN.c`

Algoritmo de Dijkstra (1965)

```
int vez = -1, interesse = {false, ..., false}
while (true) { /* Código da Thread_i */
    interesse[i] = true;
    while (existe j!=i tal que (interesse[j]))
        if (vez != i)
            interesse[i] = false;
            while (vez != -1);
            vez = i;
            interesse[i] = true;
```

Algoritmo de Dijkstra (1965) (continuação)

```
s = i;  
print ("Thr ", i, ": ", s);  
vez = -1  
interesse[i] = false;
```

Algoritmo de Dijkstra

Garante exclusão mútua?

- Uma thread só entra na região crítica após percorrer o vetor e verificar que nenhuma outra está interessada.

Garante ausência de deadlock?

- Entre as interessadas, pelo menos a última a alterar a variável vez consegue entrar na região crítica

Garante ausência de starvation?

- Não. Uma thread pode nunca conseguir ser a última a alterar vez.
- Veja o código `dijkstra.c`

Proposta incorreta de Hyman (1966)

```
int s = 0, vez = 0, interesse[2] = {false, false};
```

Thread 0

```
while (true)
    interesse[0] = true;
    while (vez != 0)
        while (interesse[1]);
        vez = 0;
s = 0;
print ("Thr 0:" , s);
interesse[0] = false;
```

Thread 1

```
while (true)
    interesse[1] = true;
    while(vez != 1)
        while(interesse[0]);
        vez = 1;
s = 1;
print ("Thr 1:" , s);
interesse[1] = false;
```

Algoritmo do Desempate (1981)

```
int s = 0, ultimo = 0, interesse[2] = {false, false};
```

Thread 0

```
while (true)
    interesse[0] = true;
    ultimo = 0;
    while (ultimo == 0 &&
           interesse[1]);
    s = 0;
    print ("Thr 0:" , s);
    interesse[0] = false;
```

Thread 1

```
while (true)
    interesse[1] = true;
    ultimo = 1;
    while (ultimo == 1 &&
           interesse[0]);
    s = 1;
    print ("Thr 1:" , s);
    interesse[1] = false;
```