

MC504 - Sistemas Operacionais

Processos e Threads

Exclusão mútua para N threads

Profa. Islene Calciolari Garcia

Primeiro Semestre de 2017

Sumário

Algoritmo de Dijkstra

Algoritmo de Lamport (fast mutual exclusion)

Algoritmo do Desempate

Thread Gerente

Algoritmo da Padaria

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

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

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)

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

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`

Algoritmo de Dijkstra

Como ilustrar o problema de starvation?

- ▶ Trecho de código válido:

```
if (thr_id == 3)
    sleep(1);
```

- ▶ Trechos de código inválido:

```
if (thr_id == 3)
    sleep(1000000); /* dorme para sempre e
                    morre de fome... */

if (thr_id != 3)
    vez = i;        /* Nunca passa a vez para thread 3 */
```

- ▶ Veja o código `starvation-dijkstra.c`

Fast mutual exclusion (1985)

```
int interesse = {false, ..., false};
int fast_lock = 0; slow_lock = 0;

while (true) { /* Código da Thread_i */
inicio:
    interesse[i] = true;
    fast_lock = i;
    if (slow_lock != 0) {
        interesse[i] = false;
        while (slow_lock != 0);
        goto inicio;
    }
}
```

Fast mutual exclusion (continuação)

```
slow_lock = i;
if (fast_lock != i) {
    interesse[i] = false;
    for (int j = 1; j < n; j++)
        while (interesse[j]);
    if (slow_lock != i)
        while (slow_lock != 0);
    goto inicio;
}
s = i;
print ("Thr ", i, ": ", s);
slow_lock = 0;
interesse[i] = 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;
```

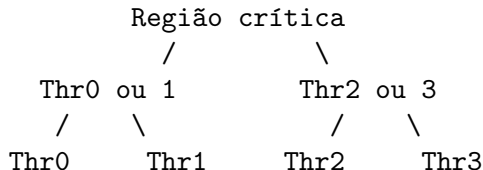
Algoritmo do Desempate

Características

Região crítica
/ \
Thr0 Thr1

- ▶ Funciona para 2 threads
- ▶ Variável ultimo é acessada pelas 2 threads
- ▶ Variável interesse[i] é acessada
 - ▶ para escrita pela thread i
 - ▶ para leitura pela thread adversária

Campeonato entre 4 threads



- ▶ A thread campeã da disputa entre Thr0 e Thr1 disputa a região crítica com a thread campeã da disputa entre Thr2 e Thr3.
- ▶ Todas as partidas são instâncias do algoritmo do desempate.

Campeonato entre 4 threads

Variáveis de controle replicadas

```
int ultimo_final = 0;  
int interesse_final[2] = {false, false};
```

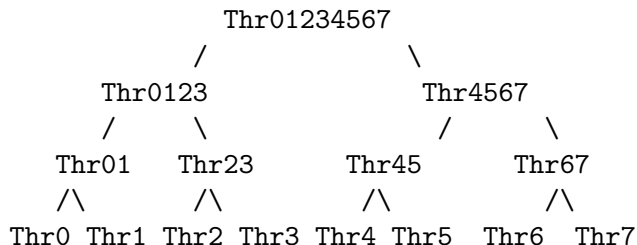
```
int ultimo01 = 0;  
int interesse01[2] = {false, false};
```

```
int ultimo23 = 2;  
int interesse23[2] = {false, false};
```

- ▶ Veja código: camp4.c
- ▶ Como implementar com uma única função para todas as threads?

Exclusão mútua entre N threads

Abordagem do campeonato



- ▶ As threads podem concorrer duas a duas
- ▶ Garante ausência de starvation?

- ▶ E se tivéssemos uma thread gerente?
 - ▶ Os algoritmos seriam mais simples?
 - ▶ Qual o grande ponto negativo desta abordagem?
- ▶ Veja os programas: gerente?.c

Algoritmo da Padaria - Lamport

- ▶ Análogo a um sistema de distribuição de senhas a clientes em uma loja
- ▶ A thread com a senha de menor número é atendida
- ▶ A própria thread deve escolher o seu número

Algoritmo da padaria

Primeira tentativa

```
num[N] = { 0, 0, ..., 0 }
```

Thread_i:

```
num[i] = max (num[0]...num[N-1]) + 1
```

```
for (j = 0; j < N; j++)  
    while (num[j] != 0 && num[j] < num[i]) ;
```

```
s = i;  
print ("Thr ", i, s);
```

```
num[i] = 0;
```

Algoritmo da padaria

Segunda tentativa

```
num[N] = { 0, 0, ..., 0 }
```

Thread_i:

```
num[i] = max (num[0]...num[N-1]) + 1
```

```
for (j = 0; j < N; j++)  
    while (num[j] != 0 &&  
           (num[j] < num[i] || num[i] == num[j] && j < i));
```

```
s = i;  
print ("Thr ", i, s);
```

```
num[i] = 0;
```

Algoritmo da padaria

```
escolhendo[N] = { false, false, ..., false }  
num[N] = { 0, 0, ..., 0 }
```

Thread_i:

```
escolhendo[i] = true;  
num[i] = max (num[0]...num[N-1]) + 1  
escolhendo[i] = false;  
for (j = 0; j < N; j++)  
    while (escolhendo[j]) ;  
    while (num[j] != 0 &&  
           (num[j] < num[i] || num[i] == num[j] && j < i));  
s = i;  
print ("Thr ", i, s);  
num[i] = 0;
```

Black-White Bakery

Gadi Taubenfeld

- ▶ The Black-White Bakery Algorithm. Proceedings of the 18th international symposium on distributed computing, Amsterdam, The Netherlands, October 2004. In: LNCS 3274 Springer Verlag 2004, 56-70
- ▶ rodadas de senhas coloridas
- ▶ permite senhas de tamanho fixo

Algoritmo da padaria: versão branco e preto

```
escolhendo[N] = { false, false, ..., false }  
num[N] = { 0, 0, ..., 0 }   cor = preto;
```

Thread_i:

```
escolhendo[i] = true;  
minha_cor = cor;  
num[i] = max (num[j] | minha_cor[j] = minha_cor[i]) + 1  
escolhendo[i] = false;  
for (j = 0; j < N; j++)  
    while (escolhendo[j]) ;  
    while (meu_ticket_eh_o_maior(i,j));  
s = i;  
print ("Thr ", i, s);  
cor = oposto(minha_cor);  
num[i] = 0;
```

Algoritmo da padaria: versão branco e preto

Comparação entre os tickets

- ▶ Se dois tickets têm cores diferentes
 - ▶ o ticket com a mesma cor da variável compartilhada é o maior;
- ▶ Se dois tickets têm a mesma cor:
 - ▶ o ticket com o número maior é o maior;
 - ▶ ou o desempate é feito pelo identificador da thread.