MC504/MC514 - Sistemas Operacionais

Processos e Threads 3

Islene Calciolari Garcia

Segundo Semestre de 2013

Sumário

Objetivos

- Exclusão mútua para N threads
  - Abordagem de Dekker
  - Algoritmo de Dijkstra
  - Thread gerente

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

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

Vetor de Interesse e Alternância

2 threads

```
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

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

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...

```
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)

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

Algoritmo de Dijkstra
Análise

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
Desafio

Altere o código do algoritmo de Dijkstra para 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 */
```

Thread Gerente

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