MO806/MC914
Tópicos em Sistemas Operacionais
2s2007

Processos e Threads 3

Objetivos

- Tentativas de extensão do algoritmo de Dekker para N threads
- Algoritmo de Dijkstra
- Algoritmo de Hyman

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) sempre consegue entrar na região crítica

Garante progresso sempre?

- 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;
```