Processos e Threads

Aula 4

Algoritmo de Dekker

- Combina as idéias de alternância (variável vez) e do vetor de interesse
- Garante exclusão mútua, ausência de deadlock e progresso
- Código complexo
- É possível estender este algoritmo para N threads?

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

Vetor de Interesse

Pode entrar em deadlock

```
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

Esta simplificação funciona?

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

Thread 0

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

Thread 1

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

Veja o código: dekker_bug.c

Por que não funciona?

- Uma thread não tem como diferenciar se a outra está na região crítica ou está tentando entrar na região crítica.
- E se criássemos uma variável só para isso?

Esta nova simplificação funciona?

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

Thread 0

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

Thread 1

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

Por que não funciona?

- Porque pode haver uma interrupção entre
  - a decisão de entrar na região crítica e
  - o comando que marca a entrada
- Veja o código: dekker_bug_rc.c

Algoritmo de Dekker

```
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] = 0;
```

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?

- Pelo menos uma thread (a da vez) sempre 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 = 0, interesse = {false, ..., false},
    esperando = {false, ..., false};
while (true) {  /* Código da Thread_i */
   interesse[i] = true;
   esperando[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;
   esperando[i] = false;
```

Outra sugestão...

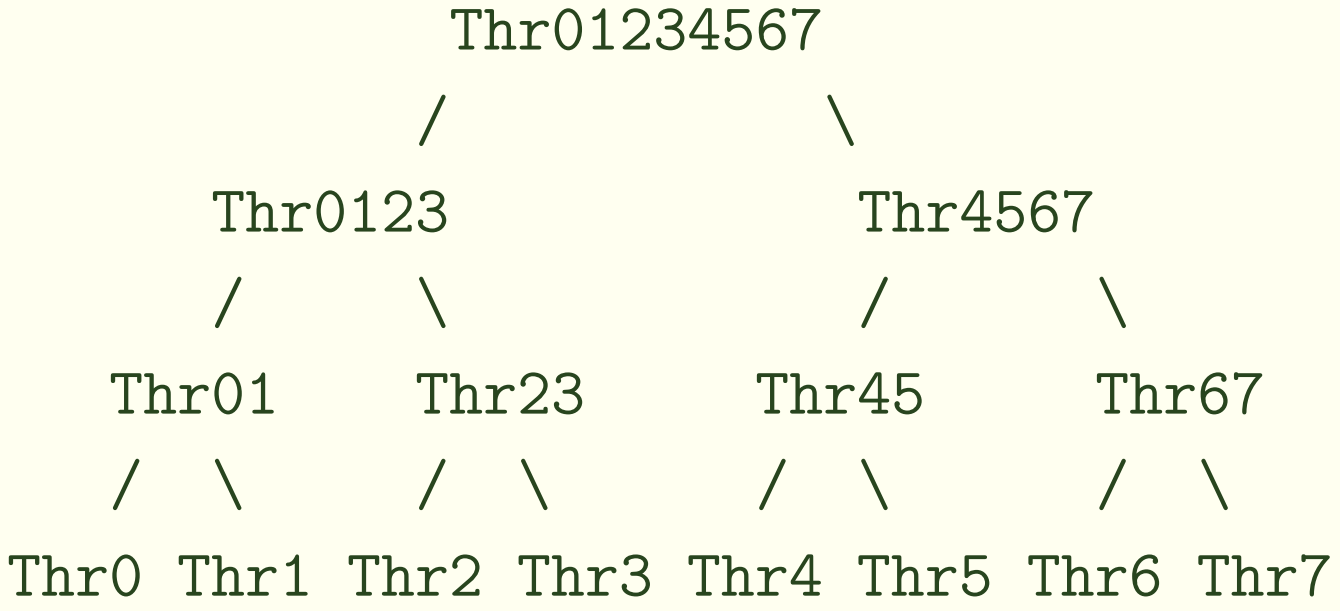
```
s = i;
print ("Thr ", i, ": ", s);
vez = identificador de alguma thread que
      está esperando ou -1;
interesse[i] = 0;
```

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

Exclusão mútua entre N threads

Abordagem do campeonato



- As threads concorrem duas a duas
- Pode ser aplicada a qualquer algoritmo

Laboratório 01

Algoritmo do desempate para N threads

- Estender o algoritmo do desempate usando a abordagem do campeonato
- N deve ser definido dinamicamente
- Pense no problema, enquanto aguarda definição completa