

**MO806/MC914**  
**Tópicos em Sistemas Operacionais**  
2s2008

**Processos e Threads 3**

# Objetivos

- Tentativas de extensão do algoritmo de Dekker para N threads
- Algoritmo de Dijkstra
- Algoritmo de Hyman
- Extensão para N threads do algoritmo do Desempate
  - Técnica do campeonato
  - Algoritmo em fases

# 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 os códigos `dijkstra.c` e `starvation-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;
```

# Algoritmo do Desempate

## 3 Threads (bug!)

```
int s=0, ultimo=0, interesse[3];
```

### Thread 0

```
while (true)
    interesse[0] = true;
    ultimo = 0;
    while (ultimo == 0 && (interesse[1] || interesse[2]));
    s = 0;
    print ("Thr 0:" , s);
    interesse[0] = false;
```

# Algoritmo do desempate

## Extensão para 3 threads

- Para 2 threads, podemos estabelecer que a thread de identificador `ultimo` perde;
- Caso 3 threads alterem a variável `ultimo` simultaneamente, só poderemos identificar a que fez a última alteração.
- Como indicar que 2 threads perderam?

# Algoritmo do Desempate

## 3 Threads

```
int s=0, ultimo, penultimo, interesse[3];
```

### Thread 0

```
while (true)
    interesse[0] = true;
    ultimo = 0;
    while (ultimo == 0 && interesse[1] && interesse[2]);
    penultimo = 0;
    while (penultimo == 0 && (interesse[1] || interesse[2]));
    s = 0;
    print ("Thr 0:" , s);
    interesse[0] = false;
```



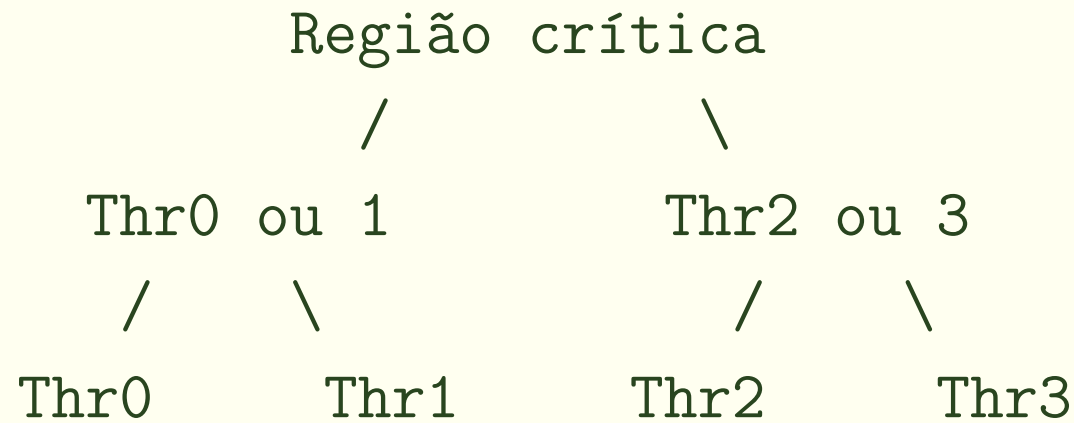
# Algoritmo do Desempate

## 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};
```

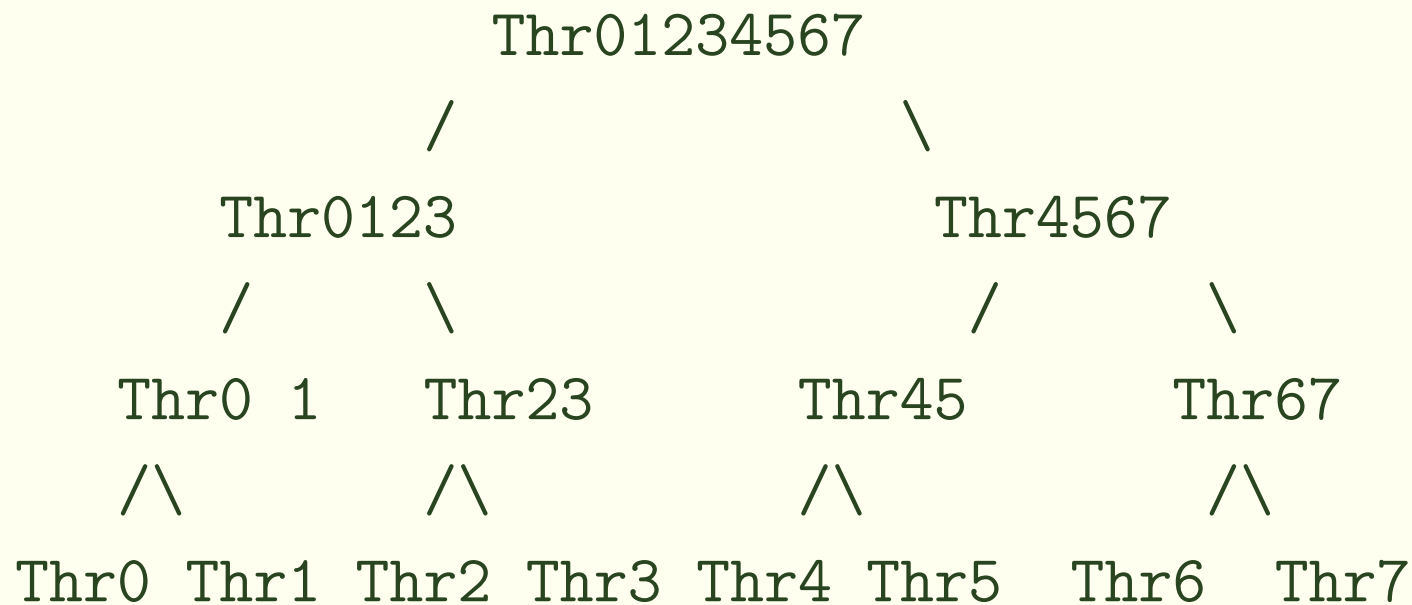
```
int ultimo01 = 0;  
int interesse01[2] = {false, false};
```

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

- Veja código: camp4.c

# Exclusão mútua entre N threads

## Abordagem do campeonato



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

# Algoritmo do desempate

## Extensão para $N$ threads

- Caso  $M$  threads alterem a variável ultimo simultaneamente, só poderemos identificar a que fez a última alteração.
- Como indicar que  $M - 1$  threads perderam?

# Algoritmo do desempate

## N threads

- Dividimos o problema em  $N-1$  fases ( $0..N-2$ )
- A cada fase, conseguimos identificar uma thread perdedora, que fica esperando
- Variáveis de controle:

```
int interesse[N]; /* -1..N-2 */  
int ultimo[N-1];
```

# Desempate para N threads

## Estado inicial

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	-1	-1	-1	-1	-1

	Fase0	Fase1	Fase2	Fase3
ultimo	—	—	—	—

# Desempate para N threads

Todas as threads interessadas

interesse	Thr0	Thr1	Thr2	Thr3	Thr4
	0	0	0	0	0

ultimo	Fase0	Fase1	Fase2	Fase3
	2	—	—	—

- Thread 2 não poderá mudar de fase



# Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	1	1	0	1	1

	Fase0	Fase1	Fase2	Fase3
ultimo	2	1	—	—

- Thread 1 não poderá mudar de fase

# Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	2	1	0	2	2

	Fase0	Fase1	Fase2	Fase3
ultimo	2	1	0	—

- Thread 0 não poderá mudar de fase

# Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	2	1	0	3	3

	Fase0	Fase1	Fase2	Fase3
ultimo	2	1	0	4

- Thread 3 pode entrar na região crítica

# Desempate para N threads

## Algumas threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	1	0	-1	-1	-1

	Fase0	Fase1	Fase2	Fase3
ultimo	1	0	—	—

- Thread 1 deverá esperar
- Thread 0 pode progredir pois as outras threads não estão interessadas

# Desempate para N threads

```
int interesse[N], ultimo[N-1];
```

## Thread\_i:

```
    for (f = 0; f < N-1; f++)  
        interesse[i] = f;  
        ultimo[f] = i;  
        for (k = 0; k < N && ultimo[f] == i; k++)  
            if (k != i)  
                while (f <= interesse[k] && ultimo[f] == i);  
s = i;  
print ("Thr ", i, s);  
interesse[i] = -1;
```

# Desempate para N Threads

- Garante exclusão mútua
- Garante ausência de deadlock
- Garante ausência de starvation
  - *deve haver um limite no número de vezes que outras threads podem entrar na região crítica (rodadas) a partir do momento que uma thread submete o pedido e o momento em que ela executa a região crítica.*
  - espera máxima =  $N(N-1)/2$  rodadas?

# Desempate para N Threads

## Pior cenário?

- $Thr_0$  perde de  $n-1$  threads na fase 0
- Estas  $N-1$  threads tentam novamente
- $Thr_0$  é desbloqueada e uma outra thread fica bloqueada na fase 0.
- $Thr_0$  perde de  $n-2$  threads na fase 1
- ...
- Como ilustrar este cenário?
- Como mostrar que este cenário pode não ser o pior?