Processos e Threads

Aula 5

Algoritmo do Desempate

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

Thread 0

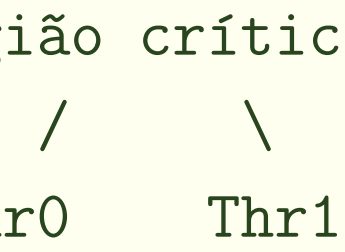
```
while (true)
  interesse[0] = true;
  vez = 0;
  while (vez == 0 &&
         interesse[1]);
  s = 0;
  print ("Thr 0:", s);
  interesse[0] = false;
```

Thread 1

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

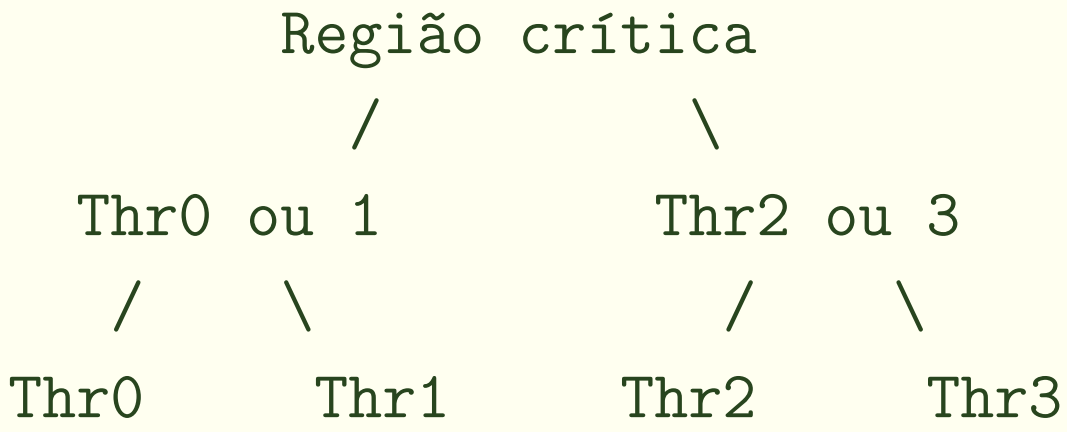
Algoritmo do Desempate

Características



- Funciona para 2 threads
- Variável vez 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

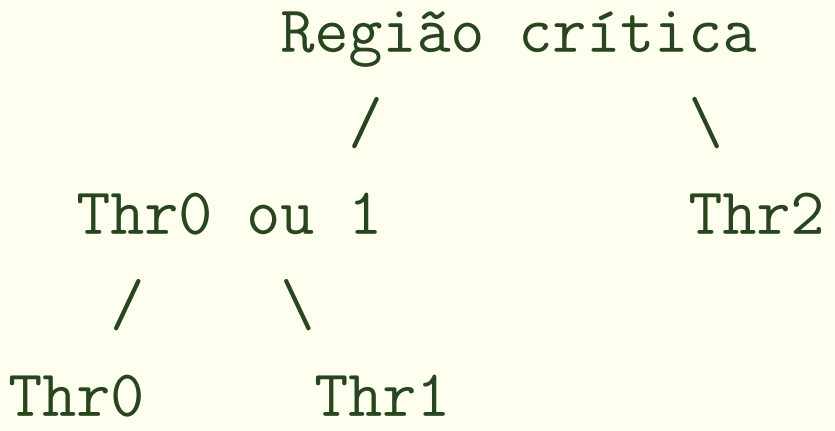
Variáveis de controle replicadas

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

int vez01 = 0;
int interesse01[2] = {false, false};

int vez23 = 2;
int interesse23[2] = {false, false};
```

- Veja código: camp4.c

Campeonato entre 4 threads

E se N não for potência de 2?



- Veja código: camp3.c

Laboratório 01

Algoritmo do desempate para N threads

- Estender o algoritmo do desempate para N threads, usando a abordagem do campeonato;
- N deve ser definido dinamicamente e passado como argumento para o programa:

  ```
  $ camp 9
  ```

- Faça alocação dinâmica;
- Escreva uma única função para as threads.

Algoritmo do Desempate

Função única para 2 threads

```
int s = 0;  /* Variável compartilhada */
int vez = 0, interesse[2] = {false, false};
Thread_i:
   int adv = i^1; /* Id da thread adversária */
   while (true)
     interesse[i] = true;
     vez = i;
     while (vez == i && interesse[adv]) ;
     s = i;
     print ("Thr %d: s = %d", i, s);
     interesse[i] = false;
```

Algoritmo do desempate

Outra extensão para N threads

- Para 2 threads, podemos estabelecer que a thread que alterou vez por último perde;
- Caso M threads alterem a variável vez 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];
int fase[N];
int vez[N-1];
```

Desempate para N threads

Estado inicial

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

	Fase0	Fase1	Fase2	Fase3
vez	–	–	–	–

Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	true	true	true	true	true
fase	0	0	0	0	0

	Fase0	Fase1	Fase2	Fase3
vez	2	–	–	–

- Thread 2 não poderá mudar de fase

Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	true	true	true	true	true
fase	1	1	0	1	1

	Fase0	Fase1	Fase2	Fase3
vez	2	1	–	–

- Thread 1 não poderá mudar de fase

Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	true	true	true	true	true
fase	2	1	0	2	2

	Fase0	Fase1	Fase2	Fase3
vez	2	1	0	–

- Thread 0 não poderá mudar de fase

Desempate para N threads

Todas as threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	true	true	true	true	true
fase	2	1	0	3	3

	Fase0	Fase1	Fase2	Fase3
vez	2	1	0	4

- Thread 3 pode entrar na região crítica

Desempate para N threads

Algumas threads interessadas

	Thr0	Thr1	Thr2	Thr3	Thr4
interesse	true	true	false	false	false
fase	2	0	-1	-1	-1

	Fase0	Fase1	Fase2	Fase3
vez	1	0	0	–

- Thread 1 deverá esperar

Desempate para N threads

```
int interesse[N], fase[N], vez[N-1];
```

Thread_i:

```
   interesse[i] = true;
   for (f = 0; f < N-1; f++)
      fase[i] = f;
      vez[f] = i;
      for (j = 0; j < N && vez[f] == i; j++ )
         if (j != i && interesse[j])
            while (f <= fase[j] && vez[f] == i);
   s = i;
   print ("Thr ", i, s);
   interesse[i] = false;
   fase[i] = -1;
```

Algoritmo da Padaria

- 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;
```

Filas de prioridades diferentes

- Suponha que o gerente da padaria está pensando em implantar atendimento especial a idosos e gestantes
- Existem threads prioritárias e outras menos prioritárias;
- Nenhuma thread menos prioritária é atendida se houver uma thread mais prioritária esperando;
- Se uma thread menos prioritária estiver sendo atendida, a mais prioritária deve esperar;

Modificação para duas filas

Duas instâncias do algoritmo da padaria

```
#define N 10
#define M 5

esc[N] = { false, false, ..., false }
num[N] = { 0, 0, ..., 0 }

esc_pri[M] = { false, false, ..., false }
num_pri[M] = { 0, 0, ..., 0 }
```

Modificação para duas filas

Uma instância do algoritmo do desempate

```
#define PRI 0
#define NAO_PRI 1
int vez;
int interesse[2];
```

Thread não prioritária

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

Thread prioritária (?)

```
esc_pri[i] = true;
num_pri[i] = max (num_pri[0]...num_pri[M-1]) + 1
esc_pri[i] = false;
for (j = 0; j < M; j++)
  while (esc_pri[j]) ;
  while (num_pri[j] != 0 && (num_pri[j] < num_pri[i] ||
         num_pri[i] == num_pri[j] && j < i));
interesse[PRI] = 1;
vez = PRI;
while (vez == PRI && interesse[NAO_PRI]);
s = i;
print ("Thr ", i, s);
interesse[PRI] = 0;
num_pri[i] = 0;
```

Thread prioritária

```
/* Código da padaria simples */
if (!interesse[PRI]){
  interesse[PRI] = 1;
  vez = PRI;
  while (vez == PRI && interesse[NAO_PRI]);
}
/* Região crítica */
if (não existe j!= i : num_pri[j] > 0)
  interesse[PRI] = 0;
num_pri[i] = 0;
```