MAC0438 - Programação Concorrente

Daniel Macêdo Batista

IME - USP, 15 de Março de 2013

Roteiro

Soluções justas para o problema da seção crítica

Primeira solução justa – Tie-Breaker

Segunda solução justa – Ticket

Soluções justas para o problema da seção crítica

Motivação

- As soluções até agora não garantem a entrada na seção crítica quando escalonadas por um escalonador com justiça fraca

  É possível que um processo nunca seja executado

  É possível que um processo que tenha começado primeiro, seja executado depois

- Obs.: embora na prática seja muito difícil um processo nunca entrar na seção crítica

Três soluções

- Algoritmo Tie-Breaker
- Algoritmo Ticket
- Algoritmo Backery

- Basta um escalonador com justiça fraca

Primeira solução justa – Tie-Breaker

Algoritmo Tie-Breaker ou de Petersen

- Faz um revezamento para os processos entrarem na seção crítica
- Desempata a disputa quando os processos tentam entrar na seção crítica
- A ideia é saber quem entrou na seção crítica por último
- Como fazer isso?

Algoritmo Tie-Breaker ou de Petersen

- Variável para marcar qual foi o último processo que entrou na seção crítica
- Não depende de instruções especiais de máquina
- Simples para 2 processos
- Complexo para n processos

Tie-Breaker (primeira tentativa)


```
while (in2) skip; /* Prot. entrada CS1 */
in1 = true;

in1 = false; /* Prot. saida CS1 */

while (in1) skip; /* Prot. entrada CS2 */
in2 = true;

in2 = false; /* Prot. saida CS2 */
```

- O protocolo de entrada não é atômico
- Não garante exclusão mútua
- O objetivo é que quando o loop do while termine, o outro processo não esteja na seção crítica

Tie-Breaker (segunda tentativa)


```
in1 = true; /* Prot. entrada CS1 */
while (in2) skip;

in1 = false; /* Prot. saida CS1 */

in2 = true;
while (in1) skip; /* Prot. entrada CS2 */

in2 = false; /* Prot. saida CS2 */
```

Tie-Breaker (segunda tentativa)


```
in1 = true; /* Prot. entrada CS1 */
while (in2) skip;

in1 = false; /* Prot. saida CS1 */

in2 = true;
while (in1) skip; /* Prot. entrada CS2 */

in2 = false; /* Prot. saida CS2 */
```

- O protocolo de entrada não é atômico mas garante exclusão mútua :)
- E as outras propriedades?

Tie-Breaker (segunda tentativa)


```
in1 = true; /* Prot. entrada CS1 */
while (in2) skip;

in1 = false; /* Prot. saida CS1 */

in2 = true;
while (in1) skip; /* Prot. entrada CS2 */

in2 = false; /* Prot. saida CS2 */
```

- Não garante ausência de deadlock :(
- Então vamos evitar o deadlock desempatando quando isso acontecer

Tie-Breaker (terceira tentativa)

- Ter uma variável `last` que armazena qual foi o último processo a começar a executar o protocolo de entrada
- Se mais de um processo tenta entrar na seção crítica (`in1` e `in2` são verdade), o último processo a começar o protocolo de entrada vai esperar
- Ideias para o algoritmo?

Tie-Breaker (terceira tentativa)

- Por enquanto com `await`

```
bool in1=false, in2=false;
int last=1;

process CS1 {
   while (true) {
      in1=true; last=1; /* Prot. entrada */
      <await (!in2 or last==2);>
      secao critica;
      in1 = false /* Prot. saida */
      secao nao critica;
   }
}
```

Tie-Breaker (terceira tentativa)


```
process CS2 {
   while (true) {
      in2=true; last=2; /* Prot. entrada */
      <await (!in1 or last==1);>
      secao critica;
      in2 = false /* Prot. saida */
      secao nao critica;
   }
}
```

Tie Breaker (quarta tentativa)


```
        in2=true; last=2; /* Prot. entrada */
        <await (!in1 or last==1);>
```

- Como tirar o `await`? Pode fazer `while !(!in1 or last==1)`?

Tie Breaker (quarta tentativa)


```
in2=true; last=2; /* Prot. entrada */
<await (!in1 or last==1);>
```

- Só pode fazer `<await (B);>` $\rightarrow$ `while (!B) skip;` se `(B)` tem a propriedade no máximo uma vez!
- Podemos relaxar o requisito de ter a propriedade no máximo uma vez?

Tie Breaker (quarta tentativa)


```
in1=true; last=1; /* Prot. entrada */
<await (!in2 or last==2);>
...
in2=true; last=2; /* Prot. entrada */
<await (!in1 or last==1);>
```

- Suponha que para CS1, a condição do `await` é verdade porque `in2` era falso

  Mas pode ser que o `in2` tenha mudado :(

  Mas o processo 2 vai terminar mudando o valor de `last`

  Se `last` mudar, vai ser para 2 e isso não afeta o resultado da expressão booleana :)

Tie Breaker (quarta tentativa)


```
in1=true; last=1; /* Prot. entrada */
<await (!in2 or last==2);>
...
in2=true; last=2; /* Prot. entrada */
<await (!in1 or last==1);>
```

- Suponha que para CS1, a condição do `await` é verdade porque `last` era 2

  Mas `last` não tem como mudar antes do CS1 executar sua seção crítica :)

  Como o `last` não muda, isso não afeta o resultado da expressão booleana :)

- Pode colocar `while`! :)

Tie Breaker (quarta tentativa)


```
bool in1=false, in2=false;
int last=1;

process CS1 {
   while (true) {
      in1=true; last=1; /* Prot. entrada */
      while (in2 and last==1) skip; /* Era await */
      secao critica;
      in1 = false /* Prot. saida */
      secao nao critica;
   }
}
```

Tie Breaker (quarta tentativa)


```
process CS2 {
   while (true) {
      in2=true; last=2; /* Prot. entrada */
      while (in1 and last==2) /* Era await */
      secao critica;
      in2 = false /* Prot. saida */
      secao nao critica;
   }
}
```

Segunda solução justa – Ticket

Ideia do algoritmo

- Mesma ideia de estabelecimentos que atendem por senha
- Há um contador único em um visor (variável compartilhada) que mostra quem é o próximo a ser atendido
- Há uma impressora que imprime uma senha (variável compartilhada) para o próximo que chega
- O que é preciso fazer?

Objetivo e ideia geral do algoritmo

- Ser mais simples que o Tie Breaker
- Usa contadores inteiros para ordenar processos (ao invés de usar o `last`)
- Um processo retira uma senha e espera a sua vez

Objetivo e ideia geral do algoritmo

- `senhaGeral` marca a senha disponível para o próximo processo. É uma variável global. Começa igual a 1
- `senhaProxima` marca a próxima senha a ser atendida. Também é uma variável global. Começa igual a 1
- `senha[i]` armazena a senha que o processo pegou. Também é uma variável global, mas sem necessidade de controle de acesso. Começa igual a 0

Protocolos de entrada e saída

- CSEntrada para o processo CS[i]:

```
<senha[i]=senhaGeral; senhaGeral++;>
<await (senha[i]==senhaProxima);>
```

- CSSaida para o processo CS[i]:

```
<senhaProxima++;>
```

Algoritmo completo


```
int senhaGeral=1, senhaProxima=1, senha[1:n]=([n] 0);

process CS[i=1 to n] {
  while (true) {
    <senha[i]=senhaGeral; senhaGeral++;> /* Prot. Entrada */
    <await (senha[i]==senhaProxima);>
    secao critica;
    <senhaProxima++;> /* Prot. Saida */
    secao nao critica;
  }
}
```

- Há entrada garantida porque uma hora a `senhaProxima` vai ser igual à `senha[i]`
- Problemas?

Algoritmo completo


```
int senhaGeral=1, senhaProxima=1, senha[1:n]=([n] 0);

process CS[i=1 to n] {
  while (true) {
    <senha[i]=senhaGeral; senhaGeral++;> /* Prot. Entrada */
    <await (senha[i]==senhaProxima);>
    secao critica;
    <senhaProxima++;> /* Prot. Saida */
    secao nao critica;
  }
}
```

- Há 3 operações atômicas definidas com < e >

  `<await (senha[i]==senhaProxima);>`

  `<senhaProxima++;>`

  `<senha[i]=senhaGeral; senhaGeral++;>`

Algoritmo completo


```
int senhaGeral=1, senhaProxima=1, senha[1:n]=([n] 0);

process CS[i=1 to n] {
  while (true) {
    <senha[i]=senhaGeral; senhaGeral++;> /* Prot. Entrada */
    <await (senha[i]==senhaProxima);>
    secao critica;
    <senhaProxima++;> /* Prot. Saida */
    secao nao critica;
  }
}
```

- Há 3 operações atômicas definidas com < e >

  `while (senha[i]!=senhaProxima)`

  `senhaProxima++;`

  Usar instrução `Fetch-and-add`

Protocolo de entrada


```
<senha[i]=senhaGeral; senhaGeral++;>
```

- Fetch-and-Add:

  ```
  FA(var, incr):
  <int tmp=var; var+=incr; return(tmp);>
  ```

Algoritmo completo sem < e >


```
int senhaGeral=1, senhaProxima=1, senha[1:n]=([n] 0);

process CS[i=1 to n] {
  while (true) {
    senha[i]=FA(senhaGeral,1); /* Prot. Entrada */
    while (senha[i]!=senhaProxima) skip;
    secao critica;
    senhaProxima++; /* Prot. Saida */
    secao nao critica;
  }
}
```

- E se o computador não tiver a instrução FA?

Protocolo de entrada sem o FA

- Qual o objetivo do protocolo?

  Tratar o caso em que dois processos chegam no protocolo de entrada na mesma hora

  Poderia colocar apenas a segunda atribuição de forma atômica? As senhas dos próximos processos sempre serão diferentes

```
senha[i]=senhaGeral; <senhaGeral++;>
```

  Resolve?

Protocolo de entrada sem o FA

- A solução é aplicar o protocolo completo dentro do protocolo de entrada

```
CSEntrada;
senha[i]=senhaGeral;
senhaGeral++;
CSSaida;
```

- Poderia implementar usando `Test-and-Set`
- Com isso alguma propriedade é perdida?

Uma última observação


```
int senhaGeral=1, senhaProxima=1, senha[1:n]=([n] 0);

process CS[i=1 to n] {
  while (true) {
    senha[i]=FA(senhaGeral,1); /* Prot. Entrada */
    while (senha[i]!=senhaProxima) skip;
    secao critica;
    senhaProxima++; /* Prot. Saida */
    secao nao critica;
  }
}
```

- Ocorrerá algum problema se o algoritmo rodar o laço do while muitas vezes? Ou ainda, se o valor de n for muito grande?