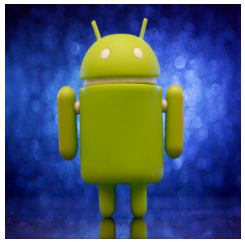
Sistemas Operacionais

Othon Oliveira

Fatec – Faculdade de Informática — PE

17 de março de 2016

Monitores – Monitores



1 Semáforos

2 Semáforos

Semáforos fora de ordem

Suponha que os dois “downs” no código do produtor, estivessem invertidos:

Semáforos fora de ordem

Suponha que os dois "downs" no código do produtor, estivessem invertidos:

```
void producer(void){
  while(true){ /* True é a constante 1 */
    item = produzirItem(); /* gera o próximo item */
    down(&empty); /* decresce o contador de empty */
    down(&mutex); /* entra na região crítica */
    inserirItem(item); /* ponha um item no buffer */
    up(&mutex); /* o buffer está vazio? */
    up(&full); /* incrementa o contador de lugares preenchidos */
  }
```

Erros comuns – inversão de ordem

Operações de semáforos

Invertidos de modo que o mutex seria decrescido antes do "empty", em vez de depois.

Erros comuns – inversão de ordem

Operações de semáforos

Invertidos de modo que o mutex seria decrescido antes do "empty", em vez de depois.

Como fica o buffer?

Se o buffer estivesse completamente cheio, o que aconteceria com o produtor?

Erros comuns – inversão de ordem

Operações de semáforos

Invertidos de modo que o mutex seria decrescido antes do "empty", em vez de depois.

Como fica o buffer?

Se o buffer estivesse completamente cheio, o que aconteceria com o produtor?

O produtor seria bloqueado quando o mutex chegasse a 0 (zero)

Deadlocks

Processos bloqueados

Essa situação que um processo bloqueia outro por erro nos semáforos buffer é chamada de **Deadlock**

Deadlocks

Processos bloqueados

Essa situação que um processo bloqueia outro por erro nos semáforos buffer é chamada de **Deadlock**

Esse problema foi levantado para mostrar o cuidado que se deve ter com os semáforos

Para facilitar a escrita correta

Uma sincronização de alto nível

Um monitor é uma coleção de procedimentos, variáveis e estruturas de dados, tudo isso agrupado em um tipo especial de módulo ou pacote.

Para facilitar a escrita correta

Uma sincronização de alto nível

Um monitor é uma coleção de procedimentos, variáveis e estruturas de dados, tudo isso agrupado em um tipo especial de módulo ou pacote.

Como funciona?

Os processo podem chamar os procedimentos em um monitor quando quiserem, mas não podem ter acesso direto às estruturas internas de dados ao monitor a partir de procedimentos declarados fora do monitor.

Para facilitar a escrita correta

Uma sincronização de alto nível

Um monitor é uma coleção de procedimentos, variáveis e estruturas de dados, tudo isso agrupado em um tipo especial de módulo ou pacote.

Como funciona?

Os processo podem chamar os procedimentos em um monitor quando quiserem, mas não podem ter acesso direto às estruturas internas de dados ao monitor a partir de procedimentos declarados fora do monitor.

Propriedades dos monitores

Somente um processo pode estar ativo em um monitor em um dado momento.

Thread Despachante

C,C++

```
1  monitor exemplo
2  integer i;
3  condition c;
4  procedure produtor();
5    .
6    .
7    .
8    end;
9  procedure consumidor();
10   .
11   .
12   .
13   end;
14 end monitor;
```

Implementação

Quem é o responsável pela implementação?

Cabe ao compilador implementar a exclusão mutua nas entradas do monitor, mas um modo comum é utilizar um **mutex** ou um semáforo binário.

Implementação

Quem é o responsável pela implementação?

Cabe ao compilador implementar a exclusão mutua nas entradas do monitor, mas um modo comum é utilizar um **mutex** ou um semáforo binário.

Exemplos fáceis

Embora monitores representem um modo fácil de fazer exclusão mútua, isso não é o bastante. É preciso também uma maneira de bloquear processos quando não tiverem uma forma de continuar.

Implementação

Quem é o responsável pela implementação?

Cabe ao compilador implementar a exclusão mutua nas entradas do monitor, mas um modo comum é utilizar um **mutex** ou um semáforo binário.

Exemplos fáceis

Embora monitores representem um modo fácil de fazer exclusão mútua, isso não é o bastante. É preciso também uma maneira de bloquear processos quando não tiverem uma forma de continuar.

Variáveis condicionais

A solução está na introdução de **variáveis condicionais**, duas operações sobre elas, *wait* e *signal*.

Produtor e consumidor de novo ??

Produtor

Quando um procedimento do monitor descobre que não pode prosseguir

Produtor e consumidor de novo ??

Produtor

Quando um procedimento do monitor descobre que não pode prosseguir

Buffer

O produtor descobre que o buffer esta cheio, emite um *wait* sobre alguma variável condicional – por exemplo a **full**

Produtor e consumidor de novo ??

Produtor

Quando um procedimento do monitor descobre que não pode prosseguir

Buffer

O produtor descobre que o buffer esta cheio, emite um *wait* sobre alguma variável condicional – por exemplo a **full**

Ação

Essa açao resulta no bloqueio do sinal que esta chamando. Ela tambem permite que o outro processo anteriomente proibido de entrar no monitor agora entre

Produtor e consumidor de novo ??

Consumidor

Esse outro processo – o consumidor – pode acordar seu parceiro adormecido a partir de um **signal** para a variavel condicional que seu parceiro esta esperando.

Produtor e consumidor de novo ??

Consumidor

Esse outro processo – o consumidor – pode acordar seu parceiro adormecido a partir de um **signal** para a variavel condicional que seu parceiro esta esperando.

Buffer

O produtor descobre que o buffer esta cheio, emite um *wait* sobre alguma variável condicional – por exemplo a **full**

Produtor – consumidor

em Pascal

```
1  monitor ProdutorConsumidor
2  integer count;
3  condition full, empty;
4  procedure insere(item: integer);
5    begin
6          if count = N then wait(full);
7          insereItem(item);
8          count := count + 1;
9          if count = 1 then signal(empty)
10   end;
11 function remove: integer;
12   begin
13         if count = 0 then wait(empty);
14         remove = removeItem;
15         count := count - 1;
16         if count = N - 1 then signal(full)
17   end;   count := 0; end monitor;
```

Produtor – consumidor

em Pascal

```
1
2  procedure produtor;
3    begin
4          while true do
5          begin
6            item = produzItem;
7            ProdutorConsumidor.insert(item)
8          end
9    end;
10 procedure consumidor;
11   begin
12         while true do
13         begin
14           item = ProdutorConsumidor.remove;
15           consumeItem(item);
16         end
17   end;
```