MAC0438 – Programação Concorrente

Daniel Macêdo Batista

IME - USP, 28 de Maio de 2012

Sintaxe e semântica

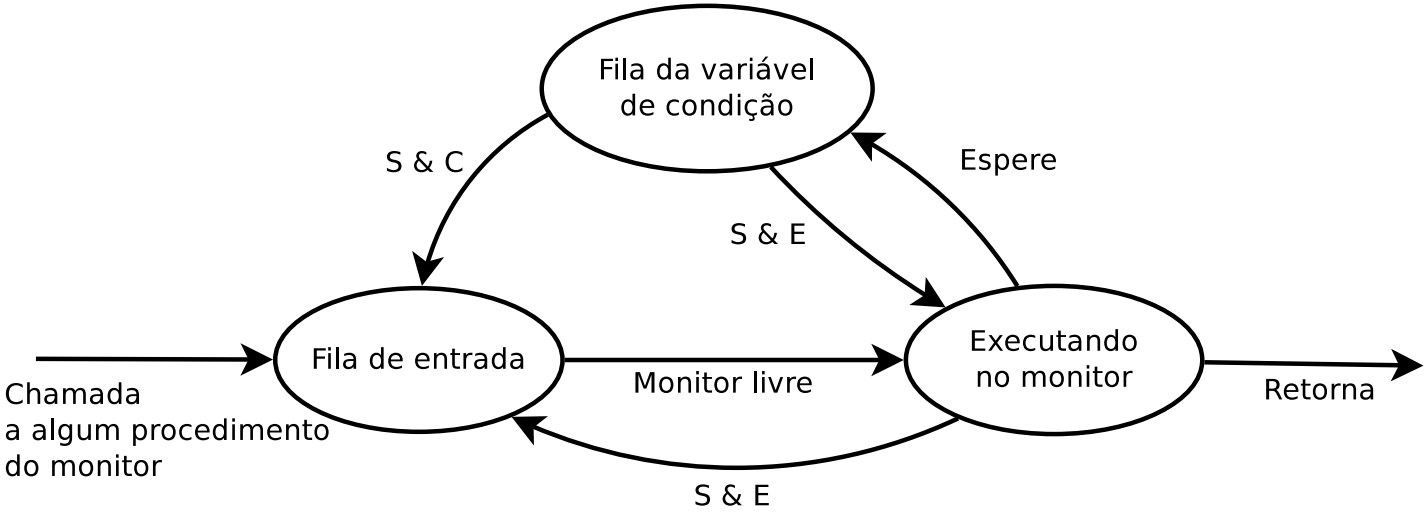
Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor nomedomonitor {
   variaveis permanentes;
   comandos de inicializacao;
   demais procedimentos;
}
```

- Variáveis permanentes $\rightarrow$ Atributos privados de uma classe
- Comandos de inicialização $\rightarrow$ Construtor de uma classe
- Demais procedimentos $\rightarrow$ Métodos de uma classe

Resumo da sincronização em monitores

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador



- Além do Signal and Urgent Wait que leva o processo sinalizador para o início da fila de entrada

Implementação melhorada de semáforo com monitor

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor SemaforoMelhorado {
   int s = 0; /* Valor do semaforo (s >= 0) */
   cond pos; /* Sinaliza quando s > 0 */

   procedure Psem() {
      if (s == 0) wait(pos);
      else s = s-1;
   }

   procedure Vsem() {
      if (empty(pos)) s = s+1;
      else signal(pos);
   }
}
```

- Técnica de passagem de condição

  O sinalizador passa implicitamente, para o processo acordado, que s é positivo

Mais métodos para manipular variáveis de condição

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

```
wait(var,rank)
```

```
minrank(var)
```

```
signal_all(var)
```

Algoritmo – Monitores com semáforos

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
variaveis compartilhadas:
  sem m = 1     /* 1 semaforo por monitor */
  int cvN = 0   /* 1 contador por variavel de condicao */
  queue cvDelay /* 1 fila por variavel de condicao */
  sem private[N] /* 1 posicao por processo */

entrada do monitor:
  P(m);

saida do monitor:
  V(m);
```

Algoritmo – Monitores com semáforos

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

```
wait(cv):
  cvN++;
  insere id do processo em cvDelay;
  V(m);
  P(private[id]);
  P(m);

signal(cv):
  if (cvN > 0) {
     cvN--;
     remove id de cvDelay;
     V(private[id]);
  }
```

Buffer limitado implementado com monitor

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor BufferLimitado {
   typeT buf[n];
   int front=0;  /* Primeira posicao com mensagem */
   int rear=0;   /* Primeira posicao vazia */
   int count=0;  /* Quantidade de posicoes com mensagens */
   /* Obs.: rear == (front+count)%n */
   cond notfull; /* Fila sinalizada quando count < n */
   cond notempty; /* Fila sinalizada quando count > 0 */
}
```

Buffer limitado implementado com monitor

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
procedure armazena (typeT data) {
   while (count == n) wait(notfull);
   buf[rear] = data; rear = (rear+1)%n; count++;
   signal(notempty);
}

procedure recupera (typeT &result) {
   while (count == 0) wait(notempty);
   result = buf[front]; front=(front+1)%n; count--;
   signal(notfull);
}
```

Roteiro

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

Continuação da implementação do buffer limitado

Verificando a implementação do buffer limitado

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
...
while (count == n) wait(notfull);
...
signal(notempty);
```

```
...
while (count == 0) wait(notempty);
...
signal(notfull);
```

- Os `while`'s garantem que quando os processos forem escalonados, as condições deles ainda estão válidas (S&C, múltiplos produtores e múltiplos consumidores)

Verificando a implementação do buffer limitado

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
...
while (count == n) wait(notfull);
...
signal(notempty);
```

```
...
while (count == 0) wait(notempty);
...
signal(notfull);
```

- Não precisa testar nenhuma condição para chamar os `signal` porque acabou de colocar ou remover uma mensagem
- Os `signal` poderiam estar em qualquer parte do procedimento porque a condição em cada processo é verificada de novo pelo `while`

Sobre o `signal` executado de forma incondicional

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- De modo geral (não só para o problema dos produtores e consumidores)

  Pode afetar a eficiência de um algoritmo caso a condição do `while` seja falsa muitas vezes antes do processo poder continuar

  O ideal é que o `signal` seja executado apenas quando há uma grande probabilidade de que o processo que estava dormindo vai poder executar

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

Leitores e escritores

Relembrando o problema

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Leitores leem uma base de dados
- Escritores alteram a base de dados
- Leitores podem acessar a base de forma concorrente
- Escritores requerem acesso exclusivo

- A base de dados é o recurso compartilhado. Por que ela não pode ser implementada como um monitor?

Como resolver o problema dos leitores e escritores com monitor?

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- A base de dados não pode ser um monitor porque leitores não poderiam ler de forma concorrente
- Vamos ter um monitor que controla o acesso à base de dados
- O monitor terá quatro procedimentos:

  requisitaLeitura: chamado pelo leitor antes de ler

  liberaLeitura: chamado pelo leitor após ler

  requisitaEscrita: chamado pelo escritor antes de escrever

  liberaEscrita: chamado pelo escritor após escrever

Como resolver o problema dos leitores e escritores com monitor?

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Condição que deve ser respeitada
- **Evitar**: `(nr > 0 && nw > 0) || nw > 1`
- é **garantir**: `(nr == 0 || nw == 0) && nw <= 1`

  `nr` é o número de leitores e `nw` é o número de escritores
- `nr` e `nw` serão incrementados nos procedimentos `requisita*` e decrementados nos procedimentos de `libera*`

Como resolver o problema dos leitores e escritores com monitor?

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

- O monitor terá duas variáveis de condição

  Uma vai sinalizar que é possível ler (quando `nw==0`)

  Uma vai sinalizar que é possível escrever (quando `nr==0 && nw==0`)

Acesso à base de dados implementado com monitor

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor Acesso_RW {
    int nr=0, nw=0;
    cond oktoread; /* Fila sinalizada quando nw == 0 */
    cond oktowrite; /* Fila sinalizada quando nr == 0
                       e nw == 0 */

    procedure requisitaLeitura() {
       while (nw>0) wait(oktoread);
       nr = nr + 1;
    }

    procedure liberaLeitura() {
       nr = nr - 1;
       if (nr == 0) signal(oktowrite);
    }
```

Acesso à base de dados implementado com monitor

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

```
    procedure requisitaEscrita() {
       while (nr > 0 || nw > 0) wait(oktowrite);
       nw = nw + 1;
    }

    procedure liberaEscrita() {
       nw = nw - 1;
       signal(oktowrite);
       signal_all(oktoread);
    }
}
```

Continuação da implementação do buffer limitado

Leitores e escritores

▷ Escalonamento de processos

Temporizador

Escalonamento de processos

Problema

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Um processo quer acessar um recurso:

  Chama `request();`

- Um processo terminou de acessar o recurso:

  Chama `release();`

- Resolvemos com semáforos

  Um vetor de n semáforos

  Uma fila ordenada pelo tempo de utilização do recurso

  Um semáforo para cada um dos n processos

- Usamos passagem de bastão

Usando monitores

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Ideias para resolver com monitores? (O escalonador de processos vai ser um monitor – Similar ao problema dos leitores e escritores)
  1. Quais métodos?
  2. Quantas variáveis de condição?
  3. Quantas variáveis convencionais?
  4. Como implementar a fila que ordena pelo tempo de execução?

Métodos

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

- `request`:

  Recebe o tempo como parâmetro

  Espera até o recurso ficar livre; ou

  É alocado para o recurso

- `release`:

  Libera o recurso; e

  Acorda o processo com tempo de finalização mais curto

Variáveis de condição

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

- 1 variável de condição: `turn`
  - Controla se o recurso está sendo usado
  - Processos vão para a fila sempre que chegarem

Variáveis convencionais

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- 1 variável booleana: `free`

  Vai marcar se o recurso está sendo usado por outro processo

Implementando a fila de processos

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Quanto menor o tempo de execução, mais próximo do início da fila o processo precisa estar
- Podemos usar o `wait` com prioridade para isso: `wait(var,rank)`
- O `rank` nesse caso vai ser o tempo de execução do processo

Juntando tudo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor TarefaMaisCurtaPrimeiro {
   bool free=true;
   cond turn;

   procedure request (int tempo) {
      while (!free) wait(turn, tempo);
      free=false;
   }

   procedure release () {
      free=true;
      signal(turn);
   }
}
```

- Resolve o problema? (Lembrando que a política de sinalização é S&C)

Juntando tudo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Usando a técnica de passagem de condição

```
monitor TarefaMaisCurtaPrimeiro {
   bool free=true;
   cond turn;

   procedure request (int tempo) {
      if (!free) wait(turn, tempo);
      else free=false;
   }

   procedure release () {
      if (empty(turn)) free=true;
      else signal(turn);
   }
}
```

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
▷ Temporizador

Temporizador

Problema

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

- Um processo deve dormir por um determinado intervalo de tempo
- Comum em sistemas operacionais para permitir que o usuário programe processos para serem executados periodicamente

  similar ao `sleep`, `usleep`, etc...

  pode servir de base para o `cron`, `at`, etc...

Usando monitores

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Ideias para resolver com monitores?
  1. Quem vai ser o recurso compartilhado implementado como um monitor?
  2. Quais métodos?
  3. Quantas variáveis de condição?
  4. Quantas variáveis convencionais?
  5. Como implementar a fila que ordena qual processo vai ser acordado primeiro?

Primeira solução – Métodos

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- O recurso será um relógio lógico
- O relógio será acessado por processos que querem dormir
- O relógio terá conhecimento sempre que um `tick` passar

Primeira solução – Métodos

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- O recurso será um relógio lógico
- O relógio será acessado por processos que querem dormir (método `delay(intervalo)`)
- O relógio saberá sempre que um `tick` passar (método `tick()`)

- Quem chama o `delay`?
- Quem chama o `tick`?

Primeira solução – Métodos

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- `delay(intervalo)`:

  Chamado pelos processos do usuário

  Enviará o processo para a fila de uma variável de condição

- `tick`:

  Chamado por um processo de alta prioridade

  O processo pode ser acordado por um temporizador do hardware

Primeira solução – Variáveis de condição

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- 1 variável de condição: `check`

  Enfileirar os processos até o `intervalo` ter passado

Primeira solução – Variáveis convencionais

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- 1 variável inteira: `tod`

  sempre é ≥ 0 e sempre muda seu valor $+1$

- Quando um processo vai para a fila da variável `check`, não pode acordar antes de terem passado `intervalo` unidades de tempo

  Obs.: aceitamos um pequeno atraso no despertar do processo porque o processo de alta prioridade tem que rodar para executar o `tick`

Primeira solução – Variáveis convencionais

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Quando um processo acorda?

  Precisa calcular a hora de acordar usando o `tod` na hora que executa o `delay`

  Cada processo tem sua hora de acordar, então precisa ser uma variável local (`wake_time`)

```
wake_time = tod + intervalo;
```

Primeira solução – Resumindo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Processo de alta prioridade

  Acorda sempre que passar 1 `tick` no temporizador de hardware

  Executa o método `tick` que só faz incrementar o `tod` e acorda processos da fila da variável `check`

Primeira solução – Resumindo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Processo do usuário

  Calcula o `wake_time`

  Verifica se o `tod` já alcançou o `wake_time`. Se não, vai para a fila da variável `check`

Primeira solução – Juntando tudo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Como fazer o processo de alta prioridade acordar exatamente quem precisa acordar? (o `wake_time` é local a cada processo de usuário)
- Cada processo poderia ter uma variável de condição privada e ele iria para aquela fila
- Poderia ter um vetor permanente no monitor
- Cada processo escreveria no vetor o seu `wake_time`
- O processo de alta prioridade olharia o vetor a cada `tick` e mandaria um `signal` para as variáveis de condição dos processos que chegaram no `wake_time`

Primeira solução – Juntando tudo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- Como fazer o processo de alta prioridade acordar exatamente quem precisa acordar? (o `wake_time` é local a cada processo de usuário)

  Criar tanta variável e vetor é custoso

  Uma forma de resolver é ter uma condição booleana que cubra indiretamente todas as condições de todos os processos dormindo

Primeira solução – Juntando tudo

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- **Condição de cobertura**

  Cada incremento no `tod` pode fazer algum processo chegar no seu `wake_time`

  O incremento no `tod` pode fazer o processo de alta prioridade enviar um `signal` para todos os processos

- Usando condição de cobertura, cada processo tem que verificar se sua condição está sendo atendida

Primeira solução

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor Temporizador1 {
   int tod=0;
   cond check;

   procedure delay (int intervalo) {
      int wake_time;
      wake_time = tod + intervalo;
      while (wake_time > tod) wait(check);
   }

   procedure tick() {
      tod++;
      signal_all(check);
   }
}
```

Segunda solução

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- A solução anterior é ineficiente pois acorda todos os processos a cada `tick`!
- Muitos processos que tenham que ser acordados num futuro distante farão o computador fazer um monte de comparação desnecessária
- Obs.: condição de cobertura é apropriada apenas quando o custo dos alarmes falsos for menor do que o custo de ter variáveis de condição específicas para cada processo

Segunda solução

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- O ideal seria verificar os processos na ordem que eles precisam acordar
- Se encontra um processo que não precisa acordar, não olha os outros (eles estão ordenador pela hora de acordar)
- Como modificar a ordem com que os processos são enviados para a fila da variável de condição `check`?

Segunda solução

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

- `wait` com prioridade:

  Temos uma ordem estática entre os processos

  Processos são ordenados pelo `waking_time`

  O processo de alta prioridade tenta acordar o primeiro processo da fila até chegar algum cujo `waking_time` não tenha sido atingido ainda. A partir daí, não precisa mais tentar acordar

Segunda solução

Continuação da implementação do buffer limitado
Leitores e escritores
Escalonamento de processos
Temporizador

```
monitor Temporizador2 {
   int tod=0;
   cond check;

   procedure delay (int intervalo) {
      int wake_time;
      wake_time = tod + intervalo;
      if (wake_time > tod) wait(check,wait_time);
   }

   procedure tick() {
      tod++;
      while (!empty(check) && minrank(check) <= tod)
         signal(check);
   }
}
```

Tarefa extra (+0,5 na MF)

Continuação da implementação do buffer limitado

Leitores e escritores

Escalonamento de processos

Temporizador

- Melhorar o monitor que resolve o problema dos leitores e escritores (slides 20 e 21) de modo a evitar que os leitores monopolizem o acesso à base