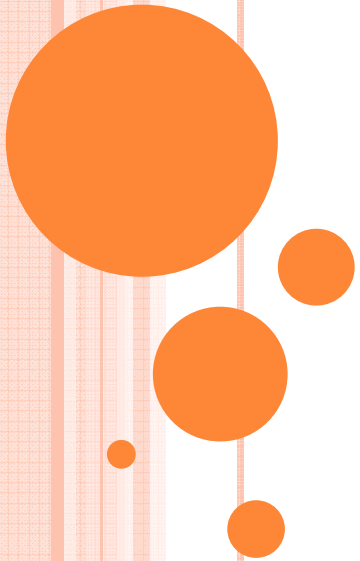

# O Problema dos Leitores e Escritores

## LABORATORIO 1

# O Problema dos Leitores e Escritores

- O problema dos leitores e escritores é o próximo problema abstrato em programação concorrente que resolveremos.
- É similar ao problema da exclusão mútua em que diversos processos estão competindo para acessar uma seção crítica.
- Neste problema, contudo, dividimos os processos em duas classes:

# O Problema dos Leitores e Escritores

- **Leitores**:  Processos, os quais não são requeridos excluir uns aos outros (entre eles).

- **Escritores**:  Processos, os quais são requeridos excluir todos os outros processos, **leitores** e **escritores**.

# O Problema dos Leitores e Escritores

- Este problema é uma **abstração do acesso à base de dados**, onde não existe o perigo em termos de diversos processos lendo concorrentemente, mas escrevendo ou mudando os dados deve ser feito sob exclusão mútua para garantir consistência.

# O Problema dos Leitores e Escritores

- O conceito de base de dados **deve ser entendido no sentido mais amplo possível**.

- Mesmo a mais simples tabela descrevendo o *status* de um **drive de disco** ou um conjunto **de janelas** sobre uma tela de terminal podem ser considerados **exemplos de uma base de dados**, que pode necessitar ser protegida por **alguma solução para o problema dos leitores e escritores**.

# O Problema dos Leitores e Escritores

- O processo que deseja **ler / escrever** chama um procedimento adequado do monitor: `Start_Read` ou `Start_Write`.

- No retorno do procedimento, o processo **lê** / **escreve** e, então, chama um outro procedimento do monitor `End_Read` or `End_Write,` para indicar ao monitor que ele terminou.

# O Problema dos Leitores e Escritores

- Mesmo antes de vermos o monitor, deve estar claro que os procedimentos `Start` podem causar um processo leitor ou escritor, **suspender sua execução** (bloquear), mas que os procedimentos `End` somente retomam a execução (desbloqueiam).

# O Problema dos Leitores e Escritores

- **Processo Reader**

- Process Reader is

```
begin
      loop
         Start-Read;
         Read_the_Data;
         End_Read;
      end loop;
end Reader;
```

# O Problema dos Leitores e Escritores

- **Processo Writer**

- Process Writer is

```
begin
      loop
         Start_Write;
         Write_the_Data;
         End_Write;
      end loop;
end Writer;
```

# O Monitor

- O monitor tem duas variáveis de status:
- `Readers`: um contador do número de leitores que tem passado bem sucedidos em Start_Read e estão correntemente lendo.
- `Writing`: Um *`flag`* que é **verdadeiro** quando um processo **escritor** está escrevendo.

# O MONITOR

- Existem duas **variáveis de condição** (Condition):
- `Ok_to_Read`: para **suspender leitores** (leitores são colocados numa fila).
- `Ok_to_Write`: para **suspender escritores** (escritores são colocados numa fila).

# O MONITOR

- A forma geral do monitor não é difícil para seguir.
- As variáveis `Readers` e `Writing` são incrementadas nos procedimentos de `Start` e decrementadas nos procedimentos de `End`.
- No início dos procedimentos `Start`, uma variável *booleana* é testada para **ver se alguma condição deve ser sinalizada**. Estas expressões determinam o comportamento do monitor.

# O Monitor

- Um **leitor** é suspenso se algum processo **escritor** está correntemente escrevendo (`Writing`) ou algum processo está esperando para escrever (`Non_Empty (Ok_to_Write)`).

- A primeira condição é obviamente requerida pela declaração do problema.

- A segunda condição é uma decisão para dar ao primeiro **writer** suspenso, prioridade sobre **leitores** esperando.

# O Monitor

- Por outro lado, um **escritor** é suspenso somente se existem processos correntemente lendo (Readers $\neq 0$) ou escrevendo (`Writing`).
- `Start_Write` não verifica as filas de condição.
- `End_Read` executa `Signal(Ok_to_Write)` se não mais existem **leitores**.
- Se existem **escritores** suspensos, um deles será acordado e permitido completar `Start_Write.`

# O Monitor

- Por outro lado, a operação não faz nada e retornamos ao estado inicial.
- `End-Write` dá prioridade ao primeiro leitor suspenso, se existe algum. De outra maneira, ele acorda escritores suspensos, se existem.
- Finalmente, qual é a função de `Signal(Ok_to_Read)` em `Start_Read` ?

# O MONITOR

- Esta declaração realiza um ***cascaded wakeup*** dos leitores suspensos. Ao término de um escritor, damos prioridade para acordar um leitor suspenso, sobre um escritor suspenso.

# O Monitor

- Contudo, se um processo é permitido **ler**, poderíamos por bem acordar todos.

- Quando o primeiro leitor completa `Start_to_Read,` ele assinalará (`signal`) ao próximo **leitor**, e assim por diante, até que esta cascata de `signals` acorde todos leitores correntemente suspensos.

# O Monitor

- O que se sabe sobre **leitores** que tentam iniciar leitura durante o *cascaded wake-up* (a cascata de ativação) ?

- Eles terão prioridade sobre **escritores** suspensos ?

- Pela exigência de retomada imediata, o *cascaded wake-up* será executada até seu término, antes de qualquer novo leitor seja permitido começar a execução de um procedimento do monitor.

# O Monitor

- Quando o último `Signal(Ok_to_Read)` for executado (e nada é feito porque a fila de leitores está vazia), o monitor será liberado e um novo leitor pode entrar.

- Contudo, ele (o leitor) está sujeito à verificação usual que causará ele, leitor, suspender, se existem escritores esperando.

# O MONITOR

- Para sumarizar:
  - Se existem **escritores** suspensos, um novo **leitor** é requerido esperar até o término do (ao menos) primeiro **escritor**.
  - Se existem **leitores** suspensos, eles serão (todos) liberados, antes do próximo **escritor**.

# VARIÁVEIS DE CONDIÇÃO

- Uma variável de condição (Condition) **C** tem três operações definidas:

  - **`Wait(C)`** O processo que chamou o procedimento do monitor, satisfazendo a condição **C** é suspenso e colocado numa fila associada a **C**. **`Wait(C)`** pode ser lida: "Estou esperando para C ocorrer".

  - **`Signal(C)`** Se a fila para a condição C é não vazia, então acorde o processo na cabeça da fila. **`Signal (C)`** pode ser lida: "Estou sinalizando que C ocorreu".

  - **`Non_Empty (C)`** Uma função booleana que retorna ***true*** se a fila para **C** é não vazia.

# O MONITOR

```
monitor  Reader_Write_Monitor  is
   Readers: Integer := 0;
   Wrinting: Boolean := false;
   OK_to_Read, Ok_to_Write: Condition;

 procedure Start_Read is
   begin
     if Writing or
        Non_Empty(Ok_to_Write) then
        Wait(ok_to_Read);
      Readers := Readers + 1;
      Signal(Ok_to_Read);
    end Start_Read;
```

# O MONITOR

```
procedure End_Read is
begin
   Readers := Readers - 1;
    if Readers = 0 then
       Signal(Ok_to_Write);
end End_Reader;
```

# O MONITOR

```
procedure Start_Write is
  begin
   if Readers /= 0 or Writing then
      wait(Ok_to_Write);
   Writing := true;
end Start_Writer;
```

# O MONITOR

```
procedure End_Write is
  begin
   Writing := false;
   if Non_Empty(Ok_to_Reader) then
      signal(Ok_to_Read);
   else
      signal(Ok_to_Write);
end End_Writer;

end Read_Writer_Monitor;
```