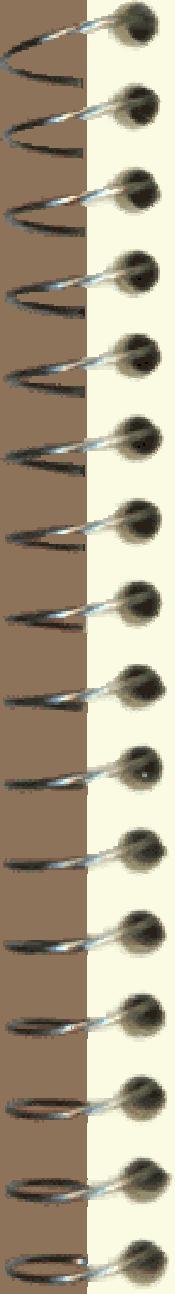


Sistemas Operacionais

Sincronização: Semáforos

Problema dos Leitores/Escritores



Autoria

Autores

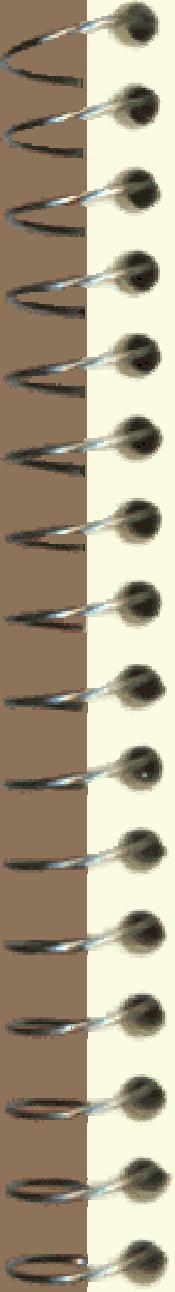
- Eduardo André Mallmann 0905/96-9
- João Paulo Reginatto 1530/96-0

Local

- Instituto de Informática
- UFRGS
- disciplina: Sistemas Operacionais II

Versão

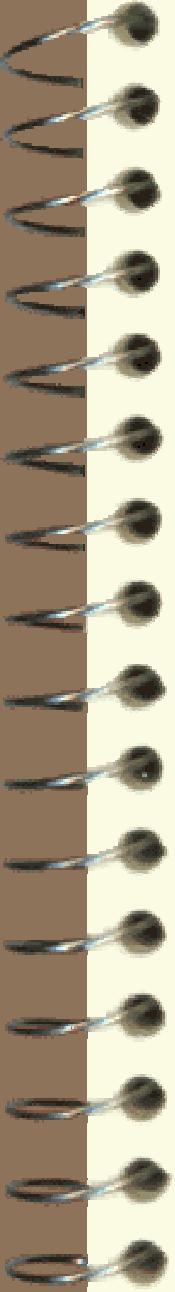
- V7, 2007-2



Autoria

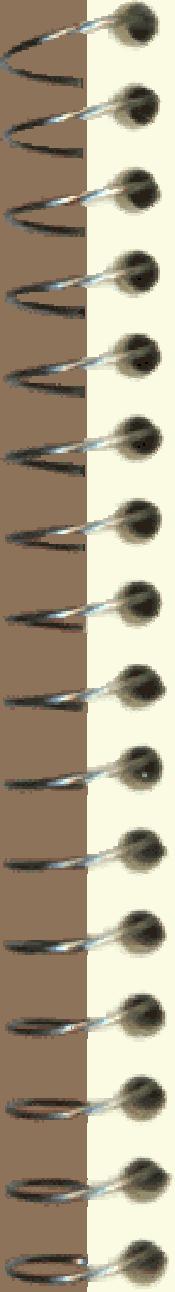
Versões

- janeiro de 2002
 - por C. Geyer
- setembro 2005
 - por C. Geyer



Baseado em que ??

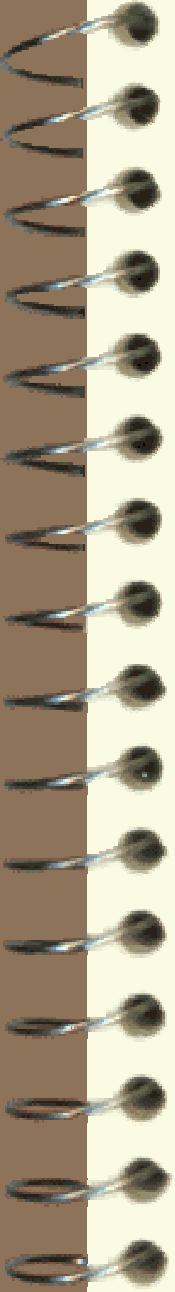
- Essas transparências são baseadas na apostila do Prof. Simão Sirineo Toscani - Introdução aos Sistemas Operacionais - 1996



Especificação do Problema

📋 O problema de sincronização do tipo leitores/escritores surge quando

- vários processos acessam para leitura ou (exclusivo) escrita uma mesma estrutura de dados.



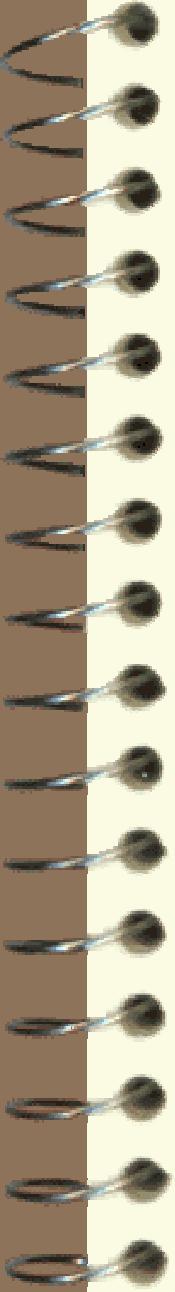
Processo Leitor

Leitor

- é aquele processo que acessa mas não altera a informação.

 Dessa forma notamos que, podemos ter vários processos leitores utilizando ao mesmo tempo a informação compartilhada

- uma vez que nenhum desses processos realizará alterações na mesma.



Processo Escritor



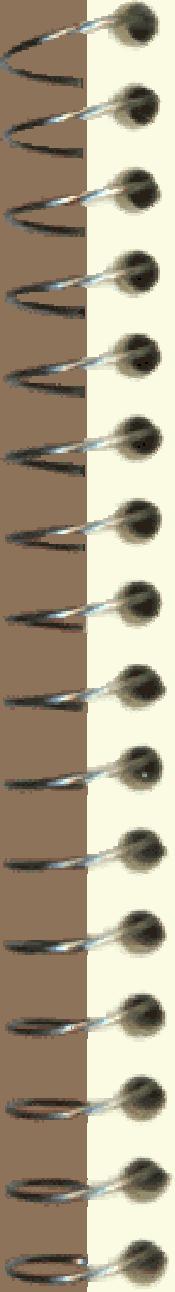
Por sua vez o processo escritor

- é aquele que acessa a informação e faz alterações na mesma.



Este acesso deverá ser exclusivo a este processo

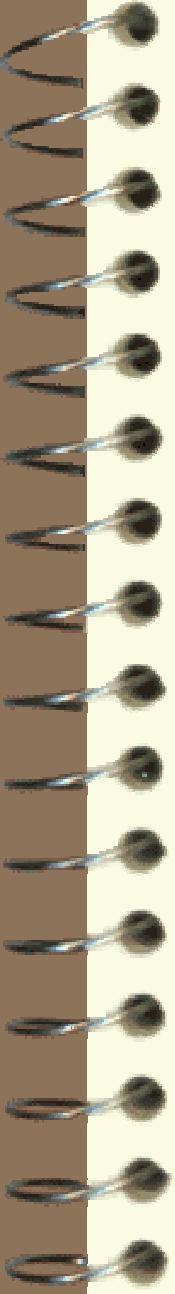
- já que o dado será alterado.



O exemplo

■ No exemplo que será mostrado a seguir

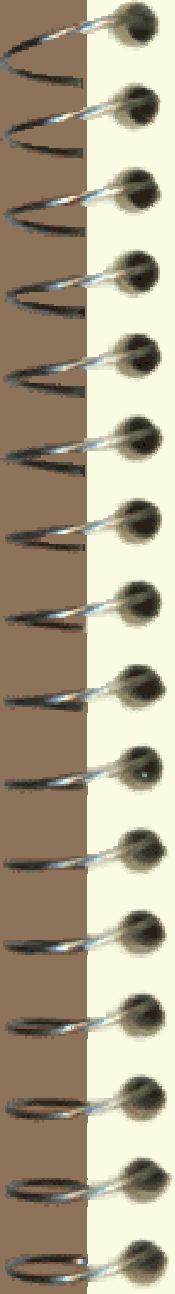
- usaremos como informação a ser compartilhada um arquivo global, acessado por vários processos.



Sincronização

Então a sincronização a ser implementada no acesso ao arquivo global

- deverá permitir que qualquer número de leitores tenham acesso a ele simultaneamente.
- Entretanto, qualquer processo escritor alterando o arquivo deverá ter acesso exclusivo ao mesmo.

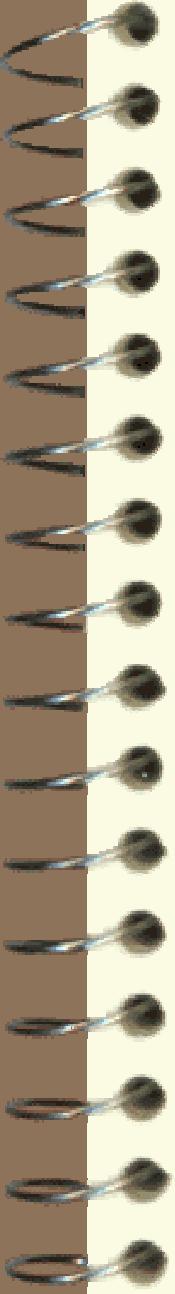


Soluções



Serão mostradas duas soluções para o problema

- A diferença entre elas é que a primeira dá prioridade aos leitores
- enquanto a segunda dá prioridade de acesso ao arquivo para os escritores.
- “dar prioridade” significa
 - que em certas situações especiais de concorrência,
 - somente os processos com prioridade conseguirão acessar o arquivo

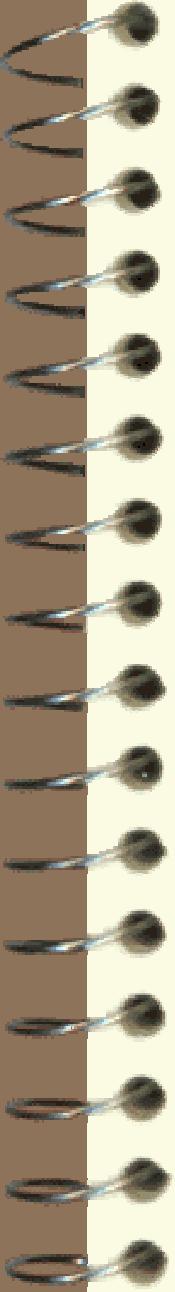


Prioridade aos leitores

■ A primeira solução garante a prioridade de acesso aos leitores.

■ Primeiro subproblema:

- como manter acesso exclusivo ao arquivo entre
 - os leitores e qualquer escritor?
 - entre os escritores?



Prioridade aos leitores

■ Primeiro subproblema:

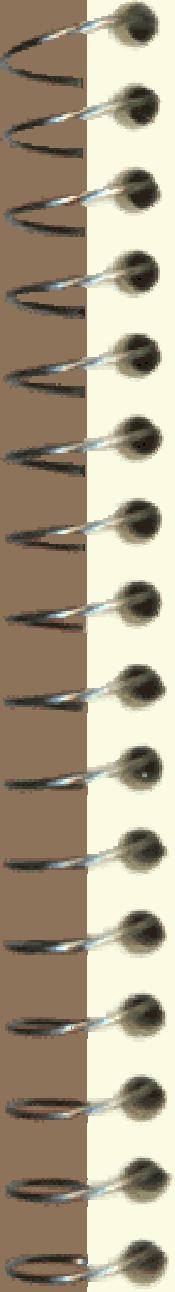
- como manter acesso exclusivo ao arquivo entre
 - os leitores e qualquer escritor?
 - entre os escritores?

■ Solução:

- Um semáforo do tipo mutex

■ Segundo subproblema:

- Como, tendo um mutex entre leitores e escritores, garantir que vários leitores possam acessar o arquivo concorrente?
- Ao menos nas situações usuais



Prioridade aos leitores

■ Segundo subproblema:

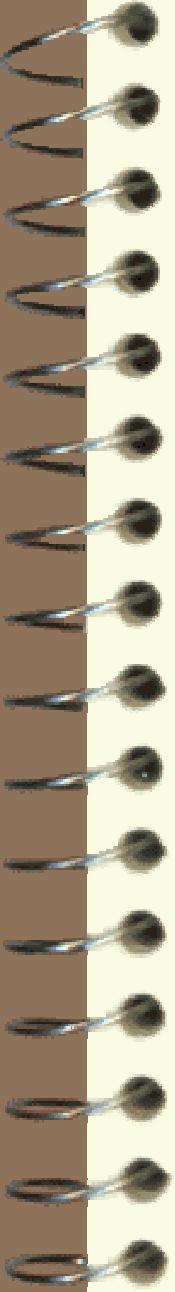
- Como, tendo um mutex entre leitores e escritores, garantir que vários leitores possam acessar o arquivo concorrente?

■ Solução (idéia):

- Somente um leitor bloqueia os escritores (e eventualmente bloqueia os outros leitores)

■ Novo subproblema

- Qual leitor?



Prioridade aos leitores

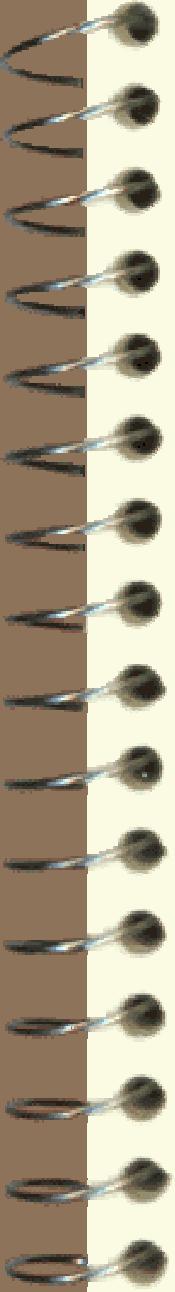
Novo subproblema

- Qual leitor?

Solução

- O primeiro

Como implementar o controle do primeiro
leitor?



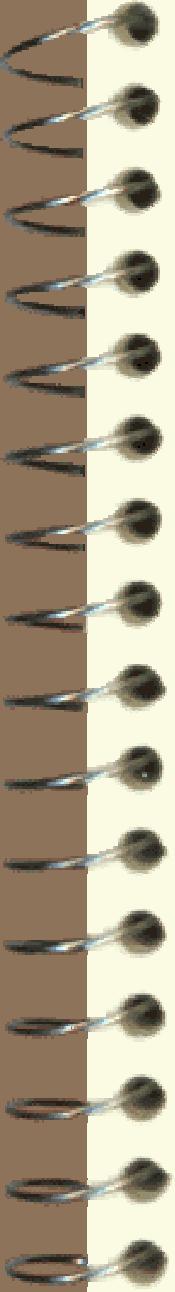
Prioridade aos leitores

�� Como implementar o controle do primeiro leitor?

�� Solução:

- Através de um contador de leitores
- O primeiro leitor executa a primitiva P(mutex)
 - Compete com os escritores
- Os outros não executam P(mutex)
 - Não competem

�� Como e quando liberar o acesso aos escritores?



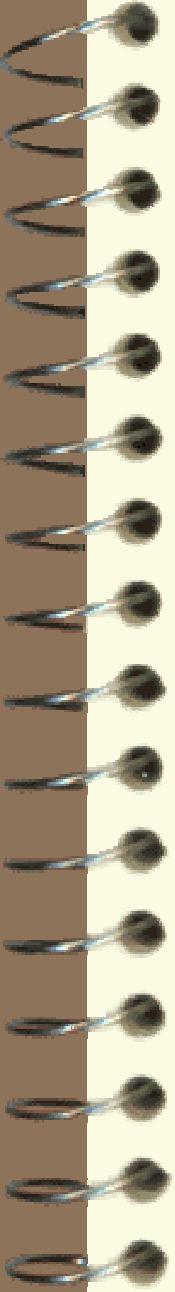
Prioridade aos leitores

�� Como e quando liberar o acesso aos escritores?

�� Solução:

- O último leitor libera o acesso aos escritores
- Executando a primitiva V(mutex)

�� Se não houver último leitor (sempre há um novo leitor concorrente)?



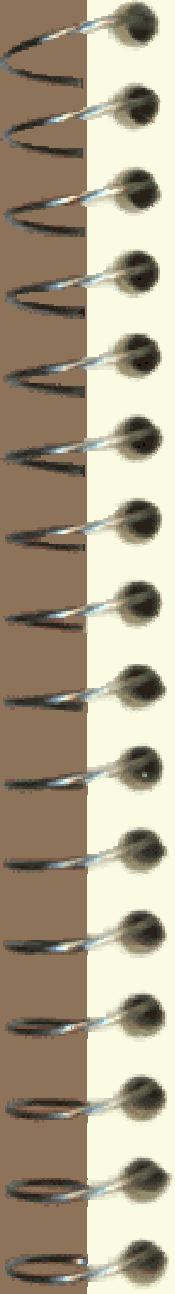
Prioridade aos leitores

■ Se não houver último leitor (sempre há um novo leitor concorrente)?

■ Solução

– Os escritores podem morrer de fome ↓

■ As operações sobre o contador de leitores necessitam de sincronização?



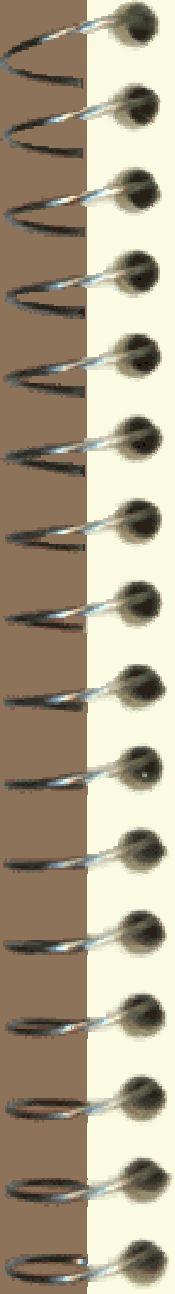
Prioridade aos leitores

As operações sobre o contador de leitores necessitam de sincronização?

– Resposta:

- sim;
- mais um semáforo do tipo mutex; seção crítica somente entre leitores

Quando um escritor estiver escrevendo, o que deve ocorrer com novos leitores que pedem leitura?

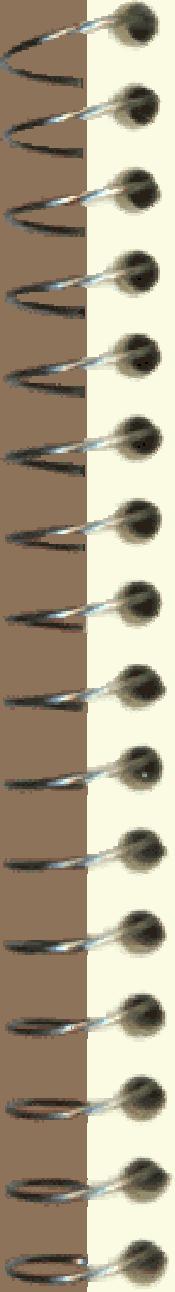


Prioridade aos leitores

■ Quando um escritor estiver escrevendo, o que deve ocorrer com novos leitores que pedem leitura?

– Resposta: devem ficar bloqueados

■ Onde? Mutex entre leitores e escritores? Ou mutex do contador de leitores? Ou um novo mutex?



Prioridade aos leitores

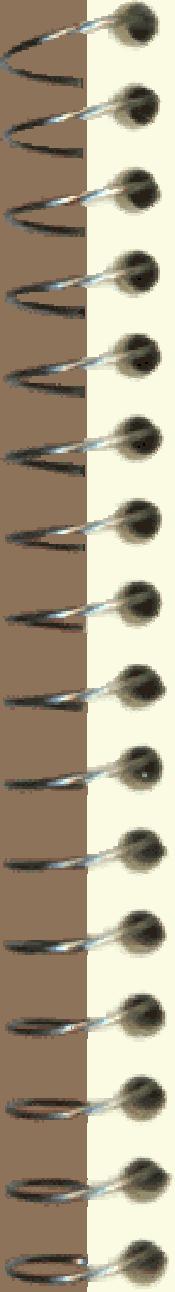
■ Onde? Mutex entre leitores e escritores? Ou mutex do contador de leitores? Ou um novo mutex?

– Resposta:

- mutex do contador de leitores

– Observação:

- O “primeiro” leitor está em ...?



Prioridade aos leitores

■ Serão utilizadas as seguintes variáveis:

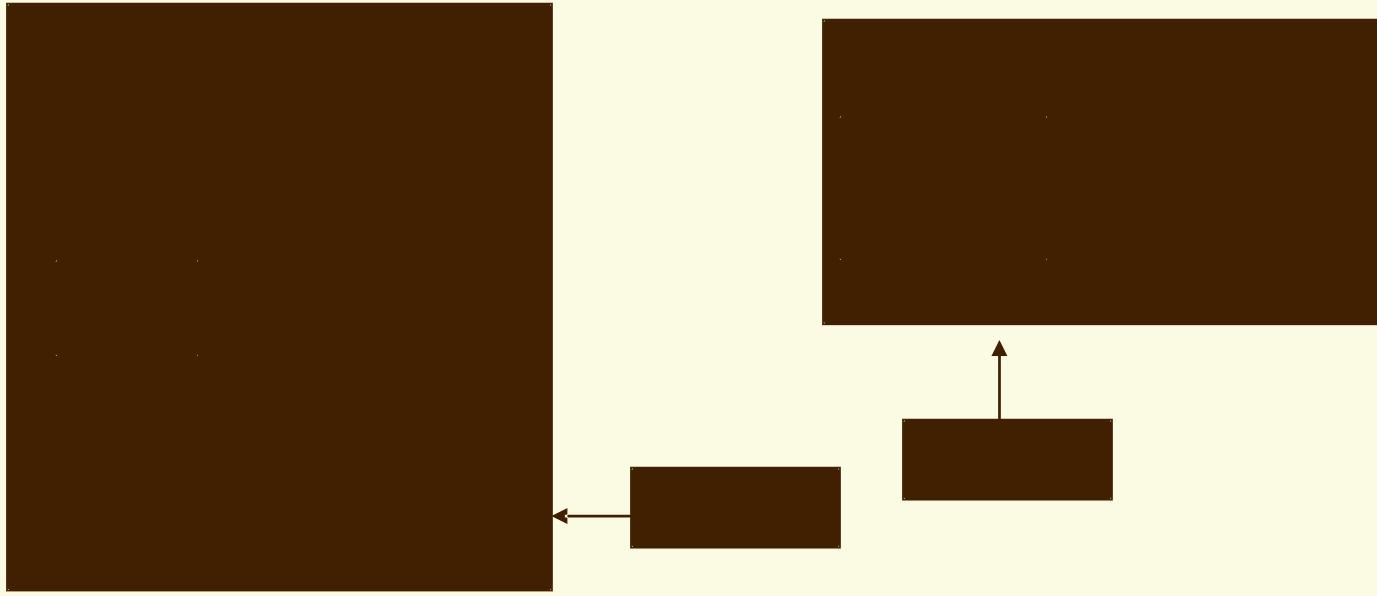
- duas variáveis semáforas: *mutex* e *e*, ambas inicializadas em 1.
 - exclusão mútua
- uma variável inteira *nl*, inicializada em 0, que contará o número de processos leitores acessando o arquivo.

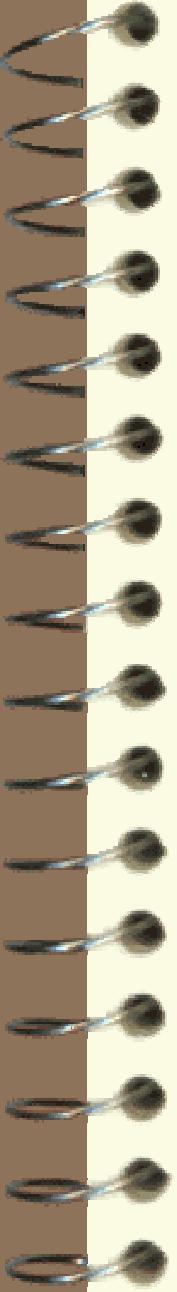
Implementação

■ Abaixo mostramos o código que cada processo (leitor ou escritor) deve executar .

Semáforo mutex, e initial (1, 1)

Integer nl initial 0





Comentários sobre o código

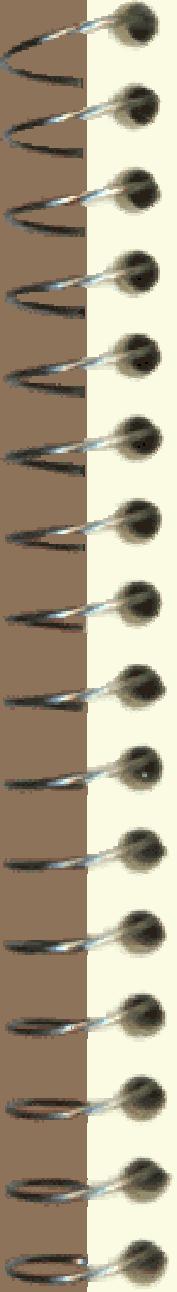
Agora o código apresentado será comentado:

Procedimento do Leitor:

- P(mutex) *esta chamada faz com que apenas um leitor altere a variável nl*
- nl := nl+1 *incrementa nl, número de leitores ativos*
- if nl = 1 then P(e) *se for o primeiro leitor a ter acesso, bloqueia o acesso por escritores*
- V(mutex) *libera o acesso de leitores a variável nl*

Depois de executar estas instruções o leitor tem acesso ao arquivo global.

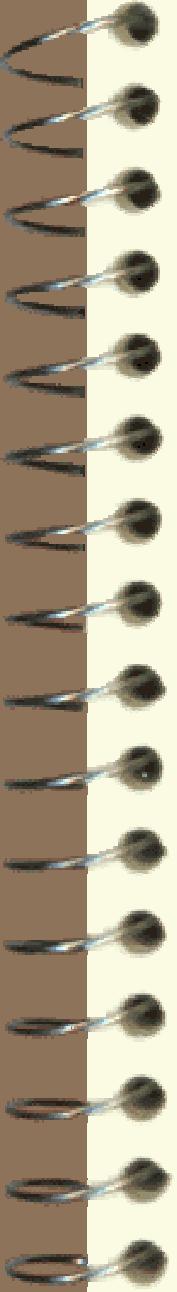
Observação: P(mutex) bloqueia novos leitores quando ...



Comentários sobre o código

Após ler o arquivo o leitor executa estas instruções:

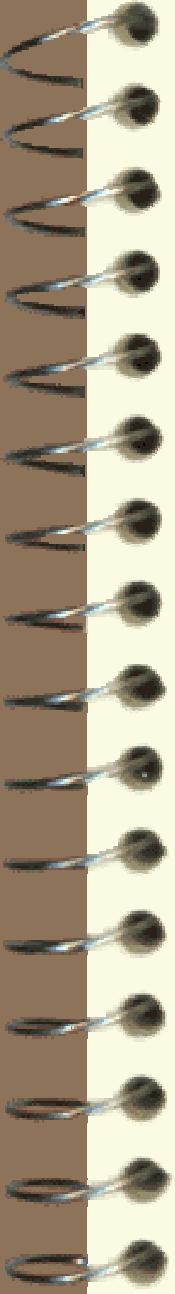
- P(mutex) *novamente faz com que apenas um leitor altere a variável nl*
- nl := nl-1 *decrementa nl, número de leitores ativos*
- if nl = 0 then V(e) *se for o último leitor a deixar o arquivo, libera o acesso por escritores*
- V(mutex) *libera o acesso de leitores a variável nl*



Comentários sobre o código

■ No processo escritor são executadas as seguintes instruções

- P(e) *faz com que o escritor tenha acesso exclusivo ao arquivo*
- Escreve no arquivo
- V(e) *libera o acesso ao arquivo*



Exercícios

Questão A)

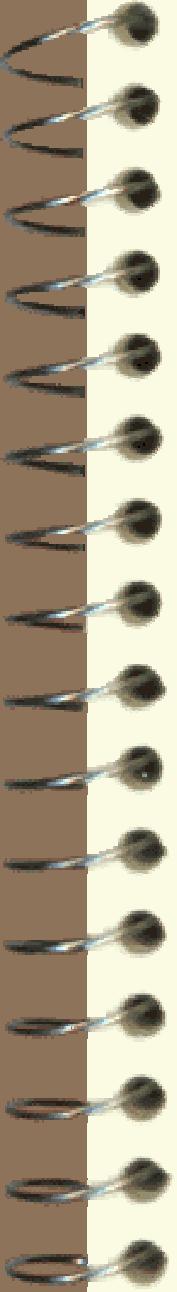
- na situação de 1 ou mais leitores lendo, onde escritores ficam bloqueados?

Questão B)

- na situação de 1 escritor escrevendo, onde os leitores ficam bloqueados (considere 2 ou mais leitores)?

Questão C)

- quando 1 escritor termina sua escrita e leitores e escritores estão esperando, quem prossegue?



Solução 2

■ A segunda solução garante a prioridade de acesso aos escritores.

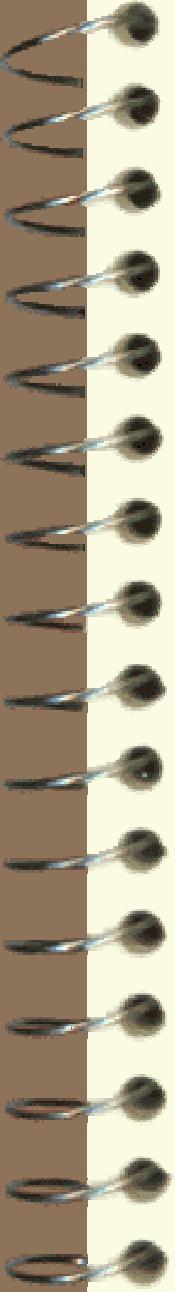
■ Requisitos

– Anteriores

- Múltiplos leitores concorrentes
- 1 único escritor escrevendo

– Novo

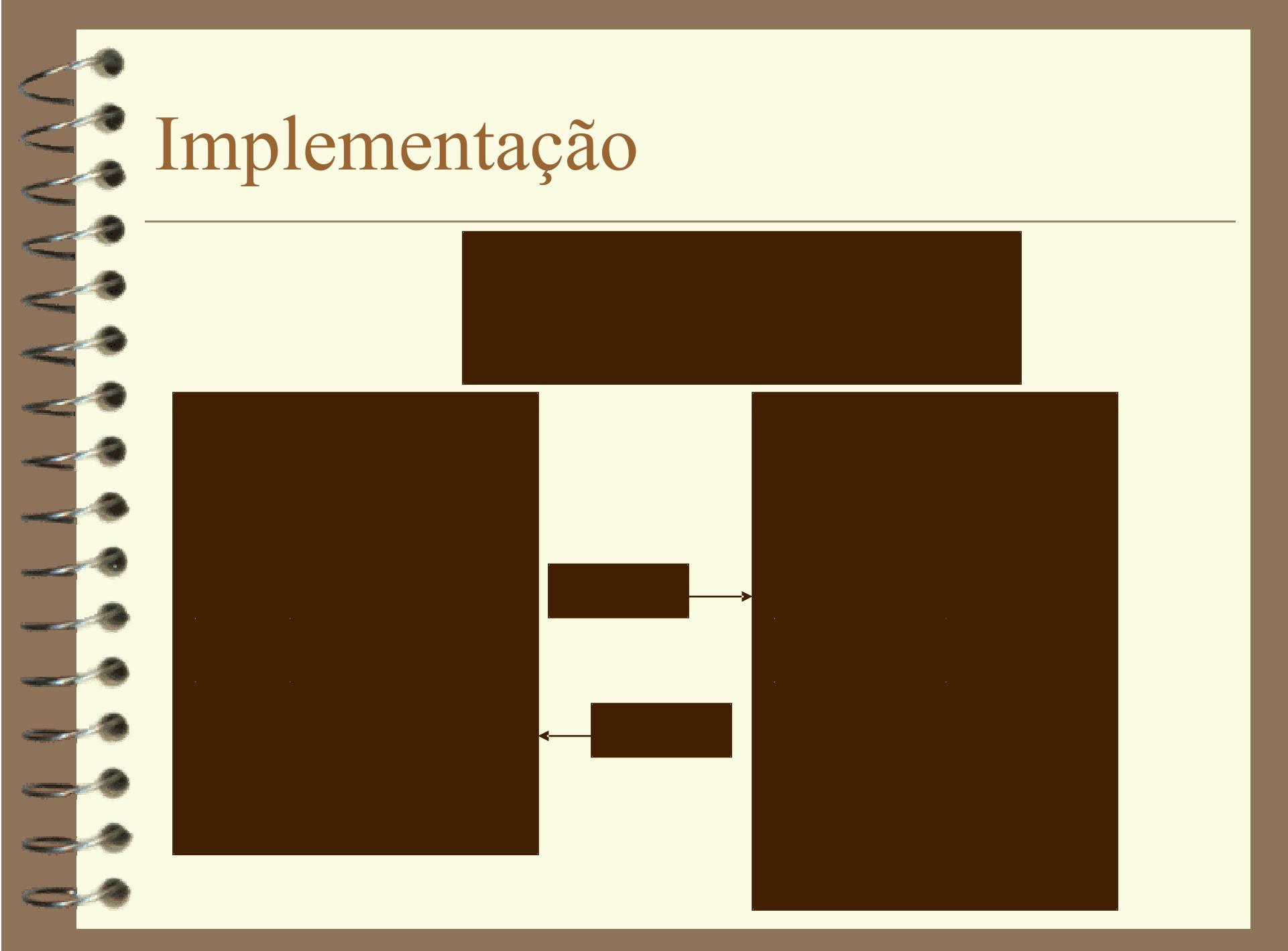
- Enquanto houver escritores em um lote de escritores “concorrentes”: leitores ficam bloqueados



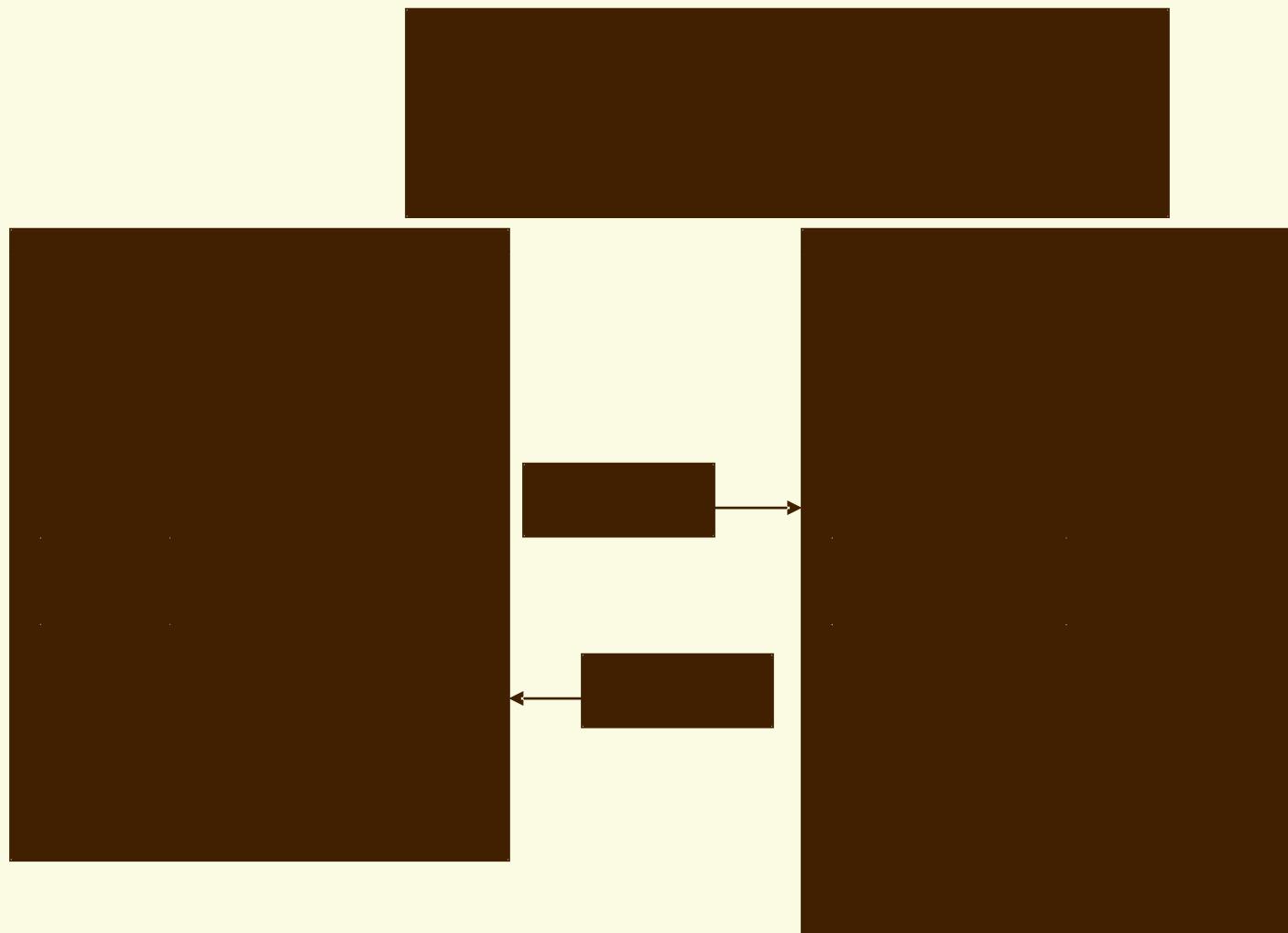
Solução 2

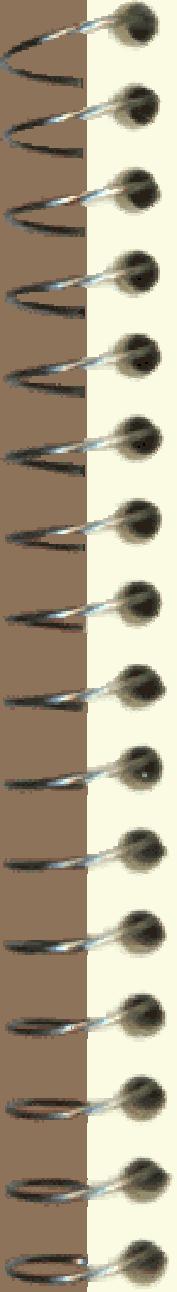
■ Serão utilizadas as seguintes variáveis:

- uma variável inteira ne , inicializada em zero, para contar o número de processos que esperam para escrever no arquivo
 - $ne = 1$ significa que um processo está escrevendo e nenhum está esperando
- duas variáveis semáforas mx e l , inicializadas em 1.



Implementação

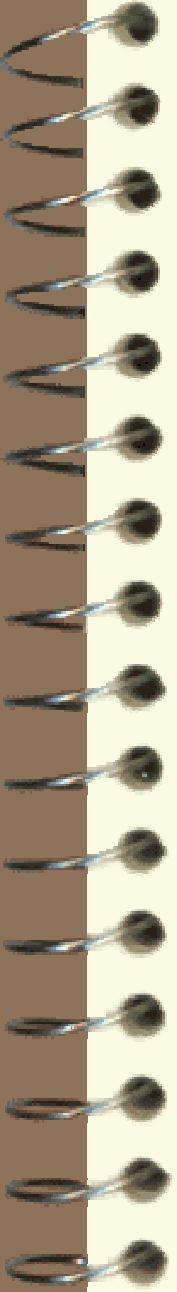




Comentários sobre o código

Leitor (1):

- P(l) *tenta obter permissão de leitor*
 - P(mutex) *restringe acesso à variável nl*
 - nl := nl+1 *incrementa número de leitores ativos*
 - if nl = 1 then P(e) *se for primeiro leitor ativo bloqueia escritores*
 - V(mutex) *libera acesso a nl*
 - V(l) *libera semáforo leitor*
-
- Depois de executar estas instruções o processo lê o arquivo



Comentários sobre o código

■ Leitor (2):

■ Depois da leitura do arquivo são estas instruções que o processo executa:

- P(mutex) *faz com que apenas um leitor altere a variável nl*
 - nl := nl-1 *decrementa nl, número de leitores ativos*
 - if nl = 0 then V(e) *se for o último leitor a terminar a leitura, libera o acesso por escritores*
 - V(mutex) *libera o acesso de leitores a variável nl*
- Como vemos esta parte do código é idêntica ao outro algoritmo.

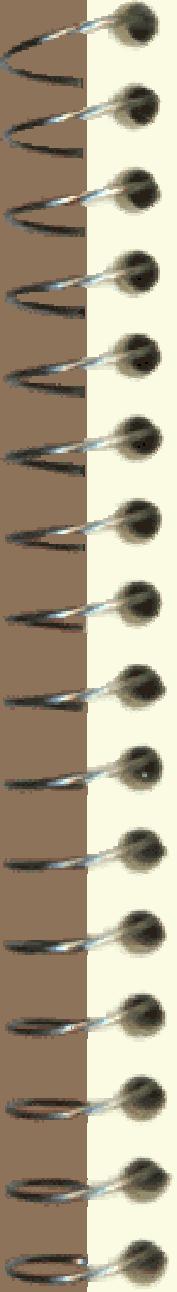
Comentários sobre o código

Escritor :

-  P(mx) *faz com que apenas um escritor altere a variável ne*
-  ne := ne+1 *incrementa número de escritores na fila de espera*
-  if ne = 1 then P(l) *se for o primeiro escritor, fecha semáforo de leitores*
-  V(mx) *libera semáforo da variável ne*
-  P(e) *fecha semáforo de escritor*

 “Altera arquivo”

-  V(e) *abre semáforo de escrita*
-  P(mx) *fecha semáforo para alteração de ne*
-  ne := ne-1 *decrementa número de escritores na fila*
-  if ne = 0 then V(l) *se processo estava escrevendo, libera leitura*
-  V(mx) *libera acesso a ne*



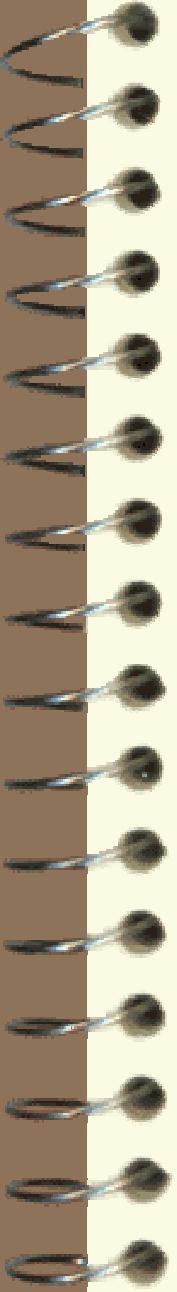
Comentários

�� Como podemos ver o raciocínio utilizado nesta solução

- é o mesmo usado na solução anterior,
- agora tratando de bloquear os processos leitores.

�� Como evitar que leitores continuamente concorrentes
bloqueiem todos os escritores?

- Solução:
 - Um novo semáforo l , inicializado em 1 (aberto)
 - Fechado ($P(l)$) pelo “primeiro escritor” de um lote de escritores concorrentes
 - Aberto ($V(l)$) pelo último escritor do lote
 - Esse semáforo, quando fechado, bloqueia todos os leitores



Comentários

Note ainda

- que esta solução não garante prioridade absoluta para os escritores.

Consideremos a situação

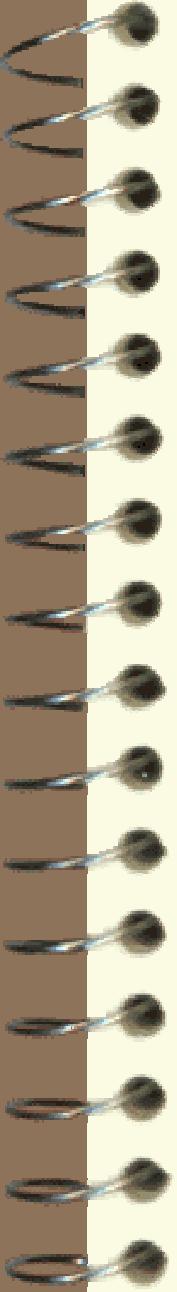
- de $ne=1$: lote de um (1) único escritor
- com 5 leitores na fila do semáforo l .

Após a escrita

- o último escritor do lote muda ne para 0,
- e então surge um novo escritor (novo lote)

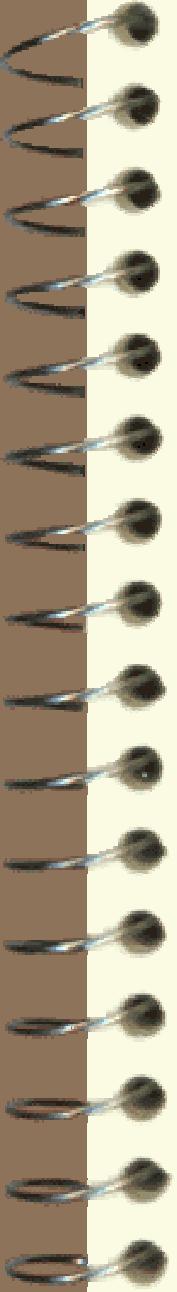
Neste caso, o novo escritor

- deverá esperar todos os leitores acabarem a leitura para poder acessar o arquivo.



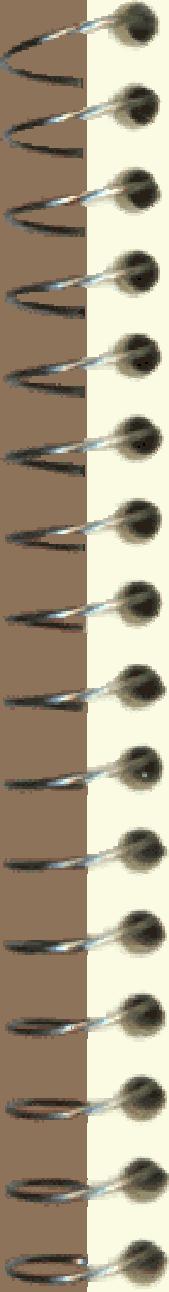
Observação

- Então, para os escritores obterem prioridade total para os escritores poderíamos criar um semáforo m_z inicializado em 1
- e colocar operações $P(m_z)$ e $V(m_z)$, respectivamente antes de $P(r)$ e depois de $V(r)$, no código dos leitores.
- Desta maneira a fila do semáforo r passará a ter no máximo um processo leitor.



Conclusão

- Aqui acaba este mini-curso.
- Nele foi apresentado o problema dos leitores/escritores assim como duas soluções para este problema utilizando o conceito de semáforos.
- Esperamos que esses conceitos lhe tenham sido úteis.



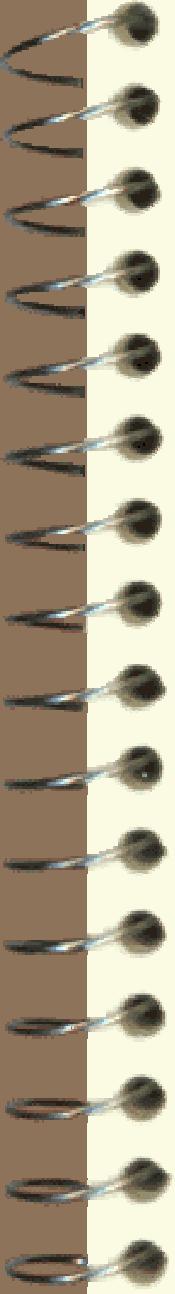
Exercícios

Exercício A)

- Escreva o código da 3a solução sugerida nos últimos slides (mais o semáforo mz)
- Analise seu comportamento; por exemplo, é possível haver vários leitores concorrentes? Justifique.

Exercício B)

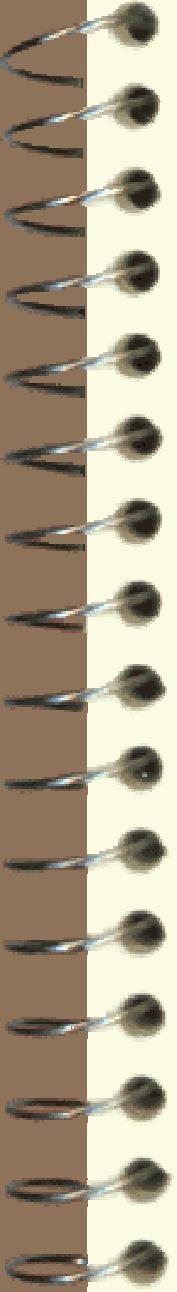
- Porque é necessário manter o semáforo *e* sobre os escritores com contador associado (nl) na solução prioridade aos escritores?



Revisão

Problema dos leitores/escritores

- Caracterize os leitores
- Caracterize os escritores
- Qual a diferença na questão de acesso a recurso comum com relação ao problema dos produtores/consumidores?
- Quais as diferenças nas questões de exceção com relação ao problema dos produtores/consumidores?



Revisão

- Como resolver o problema de acesso concorrente entre escritores?
- Como garantir acesso concorrente entre leitores mas evitar entre os leitores e os escritores?
- Algum outro semáforo é necessário? Tipo? Motivo?
- Explique a prioridade aos leitores

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.