

UNIVERSIDADE FEDERAL DE VIÇOSA
 DEPARTAMENTO DE INFORMÁTICA
 INF310 - Programação Concorrente e Distribuída
 Terceiro Teste - 23/11/2015 - Prof. Carlos Goulart.

Nome: Fernando Ferreira Passo

Matr.: 17459

1) Considere um problema no qual N processos lançadores interagem com um processo Saco. Cada um dos N processos lançador pega uma única bolinha e a lança para dentro do Saco, que tem capacidade para armazenar até K bolinhas. Quando a capacidade de bolinhas (K) do Saco é alcançada, o processo Saco envia as K bolinhas de uma só vez para um processo de nome Estoque e o saco volta a ficar vazio (sem bolinhas). Suponha que inicialmente o Saco está vazio. Implemente uma solução para a comunicação do processo Saco com os N processos lançadores e com o processo Estoque usando troca de mensagens assíncronas (suponha que a primitiva de envio (send) é não bloqueante e que a primitiva de recepção (receive) é bloqueante).

Considere que $N > K$ e que N é múltiplo de K (ex: $P = \frac{N}{K}$).

A solução deve ser livre de deadlock.

A sua solução do teste deve ser enviada por e-mail para goulart@dpi.ufv.br até às 15h de hoje (23/11). Pode enviar a imagem da solução feita à mão, se for mais conveniente. Vou postar uma possível solução na página do PVANet no final da tarde.

process lançador:

bola: integer initial 1
 count: integer initial N

loop

send (saco, bola++);
 ifte when bola == count;

end loop;

else {
 send (estoque, buffer)

m := m + 1;

ifte when m = N

end loop;

process saco:

buffer: array[1...K] of integer;
 i: integer initial 1;
 m: integer initial 1;

loop

if i < K then {

receive (lançador, bola);

buffer[(i mod K)] = bola;

i := i + 1;

process estoque:

m: integer initial 1

loop:

receive (saco, buffer)

"Transfere as K bolinhas"

m := m + 1

ifte when m = P

end loop;

Nome: Fernando Ferreira Passe Matr.: 119459

1) a) Por que as técnicas de sincronização de processos baseado em troca de mensagens são mais genéricas do que as técnicas de monitores ou semáforos?

b) No modelo de comunicação visto em sala de aula, foi considerado que nenhuma mensagem seria perdida. Entretanto, em uma aplicação real na Internet é possível ocorrer a perda de mensagens. Qual o impacto que a perda de uma mensagem pode trazer para uma aplicação?

c) Considerando o que foi dito no item "b", o que pode ser feito no código da aplicação para eliminar (ou minimizar) o impacto da perda de uma mensagem?

2) Considere a implementação de um serviço de acesso a páginas web. Este tipo de serviço é implementado no estilo cliente-servidor, no qual o cliente envia uma requisição e recebe a página solicitada (ou alguma mensagem de erro). O servidor, por sua vez, executa um loop infinito em que o comando principal é a execução de uma primitiva de recepção. Ao receber uma requisição ele responde ao cliente enviando a página solicitada, quando a página não existe). Responda e justifique os itens a seguir:

ATENÇÃO: essa questão NÃO é de múltipla escolha... ☺

- a) Que tipo de comunicação (síncrona ou assíncrona) deve ser usada entre cliente e servidor?
- b) Qual esquema de nomeação deve ser utilizada pelo cliente no envio?
- c) Qual esquema de nomeação deve ser utilizada pelo cliente na recepção?
- d) Qual esquema de nomeação deve ser utilizada pelo servidor no envio?
- e) Qual esquema de nomeação deve ser utilizada pelo servidor na recepção?

3) Considere uma tribo indígena onde N índios compartilham um jantar comunitário de um grande pote que comporta M pedaços de carne, com $M < N$. Quando um índio quer comer, ele vai ao pote e serve-se com 1 único pedaço de carne. Quando o pote esvazia, o índio cozinheiro enche novamente o pote com outros M pedaços de carne. Implemente uma solução para o problema descrito usando comunicação (com envio não bloqueante e recepção bloqueante) que atenda às seguintes condições:

- i. os processos índio e o processo cozinheiro vão executar em loop infinito;
- ii. deverá existir um processo Pote que implementará o local de armazenamento dos pedaços de carne;
- iii. seja livre de deadlock;
- iv. a notificação para o cozinheiro só deve ser enviada quando um índio encontrar o pote vazio (não conseguir pegar um pedaço de carne);
- v. O índio que não conseguiu pegar o pedaço de carne deve aguardar o pote encher.

4) Considere o corpo das tasks **esq** e **dir** mostrados a seguir.

Task body esq() is x : integer ... x := "gera um valor aleatório" meio.put(x); ...	Task body dir() y : integer ... y := "gera um valor aleatório" meio.put(y); ...
--	---

a) Esboce o código (cabecalho e corpo) da task **meio** detalhando os comandos de comunicação síncronos (**accept**).

A solução apresentada para a task **meio** deve atender aos seguintes requisitos:

- permitir a recepção de um único valor de cada processo (**esq** e **dir**);
- a ordem de recepção deve ser qualquer;
- com os 2 valores recebidos o processo **meio** deve enviá-los, em ordem não decrescente e apenas um valor de cada, para uma outra task de nome **prox**, usando a chamada para a operação de nome **next**.
- NÃO é necessário mostrar o código do processo **prox**.

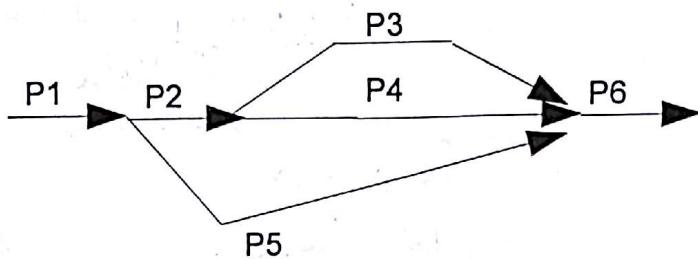
b) Esboce o código do item "a" com as alterações que devem ser feitas nas tasks **meio**, **dir** e **esq**, no caso das tasks **esq** e **dir** serem transformadas em loops para a geração de 10 números aleatórios (em cada uma), considerando que:

- não pode haver deadlock e que os três processos terminarão;
- em cada iteração na task **meio** deve ser recebido um único valor de **dir** e um único valor de **esq** (também em ordem não determinística). E os dois valores deverão ser repassados em ordem não decrescente para **prox** (que também será um loop com 10 interações – e que não precisa ser descrito).

Boa Prova!

Nome: Fernando Ferreira Passe Matr.: 77459

- 1) a) Considere uma tarefa que é implementada por 6 processos, cuja ordem de execução e/ou concorrência estão mostradas na figura abaixo:



Esboce o código dos processos P1..P6, detalhando a definição das variáveis do tipo semáforo e os comandos P e V para a sincronização baseada em semáforos. Considere que cada processo é um processo seqüencial e que executa apenas seu código apenas 1 vez.

- b) Suponha que a tarefa fosse executada em 10 passos, ou seja, cada processo deve ser executado em um loop de 10 iterações, porém mantendo a precedência da figura acima: a segunda execução de P1 só deve ocorrer após terminar a primeira execução de P6, e assim, sucessivamente. Mostre que alterações seriam necessárias na solução do item "a" para atingir este objetivo.
-

- 2) a) Faça uma comparação das técnicas de sincronização de processos usando Semáforos e Regiões Críticas Condicionais (RCCs). Quais as vantagens e desvantagens de cada uma?

- b) Explique o que é uma sincronização de barreiras. Use o grafo da questão 1 para ilustrar a sua explicação.

- c) Qual a diferença, no tocante ao acesso com exclusão mútua a uma variável compartilhada, quando se utiliza semáforos ou Monitores.

3) Suponha que existam N passageiros e um carro da montanha russa. Os passageiros repetidamente esperar para andar no carro, que pode conter no máximo C passageiros, onde $C < N$. Porém, o carro só pode iniciar uma volta quando estiver cheio. Depois de terminar uma volta, cada passageiro sai do carro e passeia pelo parque de diversões antes de retornar à montanha-russa para uma outra volta. Por razões de segurança, o carro só dá V volta, quando pára para manutenção.

Suponha que o carro e os passageiros estão representados por processos, que devem cumprir os seguintes requisitos obrigatórios:

1. O carro sempre inicia uma volta com exatamente C passageiros;
2. Nenhum passageiro vai pular fora do carro enquanto o carro está dando uma volta;
3. Nenhum passageiro saltará para dentro do carro enquanto o carro está dando uma volta;
4. Nenhum passageiro vai permanecer no carro após o término de uma volta (tentar dar duas voltas pagando apenas 1 bilhete).

Os códigos dos processos Carro e Passageiros são esboçados a seguir:

Carro:	Passageiro(<i>i</i> := 1 to <i>N</i>):	
<pre> loop ... MR.espera_carro_encher(); "dando volta" MR.espera_carro_esvaziar(); endloop </pre>	<pre> loop ... "passeia no parque" MR.entra_no_carro(<i>i</i>); "dando volta" MR.sai_do_carro(<i>i</i>); ... endloop </pre>	

Implemente uma solução usando Monitor, detalhando:

- a) as estruturas de dados para sincronizar os processos.
- b) os procedimentos usados pelos processos Carro e Passageiros.

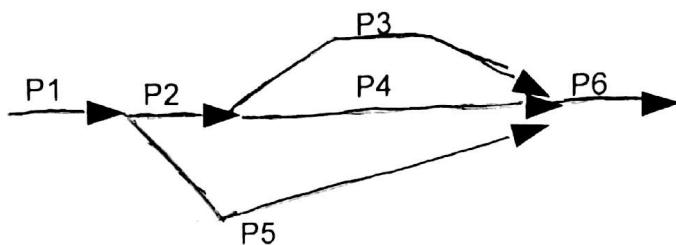
4) Considere uma tribo indígena onde N índios compartilham um jantar comunitário de um grande pote que comporta M pedaços de carne, com $M < N$. Quando um índio quer comer, ele vai ao pote e serve-se com 1 único pedaço de carne. Quando o pote esvazia, o índio cozinheiro enche novamente o pote com outros M pedaços de carne. Implemente uma solução para o problema descrito usando Regiões Críticas Condicionais (RCC).

Detalhe os dados compartilhados da RCC e os processos Cozinheiro e os N processos Índio, supondo que cada índio irá pegar K pedaços de carne.

Boa Prova!

Nome: Armando Fernandes Matr.: 68519

- 1) a) Considere uma tarefa que é implementada por 6 processos, cuja ordem de execução e/ou concorrência estão mostradas na figura abaixo:



Esboce o código dos processos P1..P6, usando a sincronização baseada em semáforos. Considere que cada processo é um processo sequencial e que executa apenas seu código apenas 1 vez.

- b) Suponha que a tarefa passasse a ter que ser executada em 10 passos, ou seja, cada processo deve ser executado em um loop de 10 iterações, porém mantendo a precedência da figura acima: a segunda execução de P1 só deve ocorrer após terminar a primeira execução de P6, e assim, sucessivamente. Mostre que alterações seriam necessárias na solução do item "a" para atingir este objetivo.
-

- 2) a) Faça uma comparação das técnicas de sincronização de processos usando Semáforos e Regiões Críticas Condicionais (RCCs). Qual delas é a melhor? Quais as vantagens e desvantagens de cada uma?

- b) Qual a diferença, no tocante ao acesso com exclusão mútua a uma variável compartilhada, quando se utiliza semáforos, RCC ou Monitores.
-

3) Suponha que existam N passageiros e um carro da montanha russa. Os passageiros repetidamente esperar para andar no carro, que pode conter no máximo C passageiros, onde $C < N$. Porém, o carro só pode iniciar uma volta quando estiver cheio. Depois de terminar uma volta, cada passageiro sai do carro e passeia pelo parque de diversões antes de retornar à montanha-russa para uma outra volta. Por razões de segurança, o carro só dá V volta, quando pára para manutenção.

Suponha que o carro e os passageiros estão representados por processos, que devem cumprir os seguintes requisitos obrigatórios:

1. O carro sempre inicia uma volta com exatamente C passageiros;
2. Nenhum passageiro vai pular fora do carro enquanto o carro está dando uma volta;
3. Nenhum passageiro saltará para dentro do carro enquanto o carro está dando uma volta;
4. Nenhum passageiro vai permanecer no carro após o término de uma volta (tentar dar duas voltas pagando apenas 1 bilhete).

Os códigos dos processos Carro e Passageiros são esboçados a seguir:

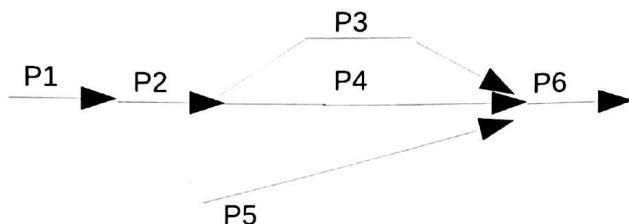
Carro: c : integer init 0; loop ... MR.espera_carro_encher(); "dando volta" MR.espera_carro_esvaziar(); c := c + 1 exit when c = V endloop	Passageiro(i := 1 to N): loop ... "passeia no parque" MR.entra_no_carro(i); "dando volta" MR.sai_do_carro(i); ... endloop
--	--

Implemente uma solução usando Monitor, detalhando:

- a) as estruturas de dados para sincronizar os processos.
- b) as operações usadas pelos processos Carro e Passageiros.

Nome: _____ Matr.: _____

- 1) a) Considere uma tarefa que é implementada por 6 processos, cuja ordem de execução e/ou concorrência estão mostradas na figura abaixo:



Esboce o código dos processos P1..P6, detalhando a definição das variáveis compartilhadas e os comandos de região para a sincronização baseada em RCCs. Considere que cada processo é um processo sequencial e que executa apenas seu código apenas 1 vez.

- b) Suponha que a tarefa fosse executada em 10 passos, ou seja, cada processo deve ser executado em um loop de 10 iterações, porém mantendo a precedência da figura acima: a segunda execução de P1 só deve ocorrer após terminar a primeira execução de P6, e assim, sucessivamente. Mostre que alterações seriam necessárias na solução do item "a" para atingir este objetivo.
-

- 2) a) Faça uma comparação das técnicas de sincronização de processos usando Semáforos e Regiões Críticas Condicionais (RCCs). Qual delas é a melhor? Quais as vantagens e desvantagens de cada uma?

- b) Qual a vantagem principal da sincronização usando semáforos sobre a técnica de sincronização baseada em espera ocupada? Explique como esta vantagem é obtida.

- c) Qual a diferença, no tocante ao acesso com exclusão mútua a uma variável compartilhada, quando se utiliza semáforos, RCC ou Monitores.

3) Considere o problema dos jantar dos selvagens descrito a seguir:

"Uma tribo de selvagens janta em conjunto, retirando pedaços de missionários assados de um grande pote que comporta até M pedaços. Quando um selvagem deseja comer ele se serve do pote, a menos que o pote esteja vazio. Se o pote estiver vazio, o selvagem acorda um cozinheiro e espera até que este tenha assado mais M pedaços de missionários e enchedo o pote. Após encher o pote o cozinheiro vai dormir"

a) Implemente uma solução baseada em semáforos para resolver este problema.

b) Analise sua solução e mostre que ela está correta.

4) Suponha que existam N passageiros e um carro da montanha russa. Os passageiros repetidamente esperar para andar no carro, que pode conter no máximo C passageiros, onde $C < N$. Porém, o carro só pode iniciar uma volta quando estiver cheio. Depois de terminar uma volta, cada passageiro sai do carro e passeia pelo parque de diversões antes de retornar à montanha-russa para uma outra volta. Por razões de segurança, o carro só dá V volta, quando pára para manutenção.

Suponha que o carro e os passageiros estão representados por processos, que devem cumprir os seguintes requisitos obrigatórios:

1. O carro sempre inicia uma volta com exatamente C passageiros;
2. Nenhum passageiro vai pular fora do carro enquanto o carro está dando uma volta;
3. Nenhum passageiro saltará para dentro do carro enquanto o carro está dando uma volta;
4. Nenhum passageiro vai permanecer no carro após o término de uma volta (tentar dar duas voltas pagando apenas 1 bilhete).

Os códigos dos processos Carro e Passageiros são esboçados a seguir:

Carro: c : integer init 0; loop ... MR.espera_carro_encher(); "dando volta" MR.espera_carro_esvaziar(); c := c + 1 exit when c = V endloop	Passageiro(i := 1 to N): loop ... "passeia no parque" MR.enta_no_carro(i); "dando volta" MR.sai_do_carro(i); ... endloop
--	---

Implemente uma solução usando Monitor, detalhando:

a) as estruturas de dados para sincronizar os processos.

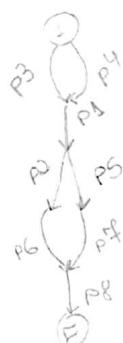
- a) as estruturas de dados para sincronizar os processos.
- b) as operações usadas pelos processos Carro e Passageiros.

UNIVERSIDADE FEDERAL DE VIÇOSA
 CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
 DEPARTAMENTO DE INFORMÁTICA
 INF 310 – Programação Concorrente e Distribuída
 Substitutiva da Segunda Prova

Nome: _____ Matrícula: _____

- 1) a) Qual(is) a(s) vantagem(ns) da utilização da técnica de semáforos sobre o uso de Monitores?
- b) Qual(is) a(s) vantagem(ns) da utilização da técnica de Monitores sobre o uso da técnica de espera ocupada (usando apenas variáveis compartilhadas)? *nova geração de espelhos de memória pode ocorrer starvation, o que não ocorre com monitores*
- c) Qual(is) a(s) vantagem(ns) da utilização da técnica de Monitores sobre o uso da técnica de Semáforos? *Simplicidade de programação, pois não é preciso controlar um grande nº de semáforos, evitando o deadlock*

- 2) Considere o código mostrado a seguir.



a, b, c, d, e: semaphore initial 0;

P1: ... P(a); P(a) “código de P1” V(b); V(b);	P2: ... P(b); “código de P2” V(c);	P3: ... “código de P3” V(a);	P4: ... “código de P4” V(a);
P5: ... P(b); “código de P5” V(d);	P6: ... P(c); “código de P6” V(e);	P7: ... P(d); “código de P7” V(e);	P8: ... P(e); P(e) “código de P8” V(f); V(f)

- a) Faça um desenho do grafo de precedência das tarefas P1 a P8 definidas pelo código acima.
- b) Suponha que os processos sejam executados de forma cíclica. Ou seja, após o término do último processo do grafo que você desenhou, a mesma sequência de processos seria executada novamente e de forma repetitiva (loop infinito). Quantos novos semáforos seriam necessários para tornar esta modificação possível. Qual o valor inicial de cada novo semáforo? *1 semáforo por item*
- c) Mostre a modificação no código dos processos P1 a P8 que seria necessária implementar para a execução na forma definida no item “b”.

- 3) Considere o problema da cervejada, descrito a seguir.

“Jovens em uma cervejada passam o tempo todo bebendo e conversando. Quando um jovem sente sede ele vai até o balcão de um quiosque e entra em uma fila indiana para pegar uma (e apenas 1) lata de bebida. Existem dois tipos de jovens: os que tomam cerveja ou refrigerante e os que tomam apenas refrigerante. Os que tomam cerveja, só tomam refrigerante quando não encontram cerveja. Quando um jovem chega ao balcão e não tem mais a bebida que ele pode pegar, ele chama o atendente do quiosque, que pega um lote da bebida preferida do jovem (48 latas de cerveja ou 24 latas de refrigerante) e coloca as latinhas no balcão. No início da cervejada o atendente coloca o lote inicial de 48 latas de cerveja e 24 latas de refrigerante no balcão.”

· Implemente uma solução para o problema da cervejada usando Monitores.

```
process Jovem_Cerveja(i : 1 to 10)
k : boolean;           /* quando k é true o jovem conseguiu pegar um a cerveja */
loop
...
Balcao.pega_cerveja(k);      /* solicita uma cerveja, mas pode receber cerveja ou refrigerante */
if k then "bebendo cerveja"; /* bebendo cerveja junto com outros jovens */
else "bebendo refrigerante"; /* bebendo refrigerante junto com outros jovens */
...
endloop

process Jovem_Refri(j : 1 to 5)
loop
...
Balcao.pega_refri();        /* solicita um rrefrigerante */
"bebendo";                  /* bebendo refrigerante junto com outros jovens */
...
endloop

process Atendente;
loop
...
Balcao.repoe_bebida; /* o tipo de bebida a ser reposta será definida por variável
                     interna do Monitor */
...
endloop;
```

4) Em um parque de diversões um dos brinquedos mais populares é a montanha russa. Os clientes (passageiros) que conseguem dar uma volta, sempre tentam dar mais uma voltinha. O carro da montanha russa tem capacidade para **C** passageiros e uma volta só pode ser iniciada quando o carro estiver com os **C** passageiros. Existem **N** passageiros tentando dar voltas, com **N > C**. Um passageiro entra no carro e espera que o carro inicie a volta (os **C** passageiros que conseguiram entrar no carro iniciarão a volta ao “mesmo tempo” - concorrentemente). Ao terminar a volta, todos os passageiros tem que sair do carro e para dar outra volta eles têm que voltar para o final da fila de entrada da montanha russa. Novos passageiros só podem entrar no carro, após todos os passageiros que deram a última volta saírem do carro.

Implemente uma solução usando Regiões Críticas Condicionais para resolver a sincronização entre os processos passageiros e o processo carro.

Boa Prova! Boas Festas! Boas Férias!

Nome: _____ Matrícula: _____

- 1) Considere o seguinte algoritmo para resolver o problema de exclusão mútua entre 2 processos P1 e P2.

Variáveis globais:

quer : array[2] of boolean initial false;
vez : integer initial 1;

P1: ...

```
quer[1] := true;
vez := 1;
loop
    exit when not (quer[2] and vez = 2);
endloop;
REGIÃO CRÍTICA;
quer[1] := false;
...
```

P2: ...

```
quer[2] := true;
vez := 2;
loop
    exit when not (quer[1] and vez = 1);
endloop;
REGIÃO CRÍTICA;
quer[2] := false;
...
```

- a) a solução proposta garante exclusão mútua? Justifique.
- b) a solução proposta garante o progresso dos processos (ausência de bloqueio)? Justifique.
- c) Caso alguma das duas propriedades não sejam satisfeitas, proponha modificações no protocolo de entrada para corrigir o algoritmo.

Nome: Amanda Fernandes Senna Matrícula: 68549

1) a) Explique o que é condição de corrida (*race condition*) e que tipo de problema essa condição pode provocar.

b) Como um possível problema provocado por uma condição de corrida pode ser evitado?

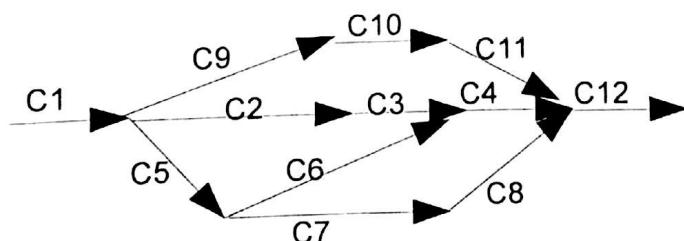
c) Explique o que é região crítica. Dê um exemplo computacional onde o conceito de região crítica necessita ser implementado.

2) a) Desenhe o grafo definido pelos comandos Serial (S) e Paralelo (P) abaixo:

$S(P(S(p1, P(p2, S(p3, P(p4, p5)))), S(P(p7, p8), p9)), p10)$

Este grafo é propriamente aninhado? Justifique.

b) Considere o grafo abaixo mostrando a precedência de execução dos comandos C_i . Escreva o trecho de código usando os comandos *fork/join/quit* para implementar a ordem de execução definida no grafo, criando o MENOR número de processos possível. Cada processo pode conter um ou mais comandos C_i .



3) Considere o seguinte algoritmo para resolver o problema de exclusão mútua entre 2 processos P1 e P2, que executam em loop infinito, mostrado no verso dessa folha.

Variáveis globais: <code>quer : array[2] of boolean initial false;</code>	<code>vez : integer initial 1;</code>
P1: ... <code>quer[1] := true;</code> <code>vez := 2;</code> <code>loop</code> <code> exit when $\neg quer[2]$ or <code>vez = 2</code>;</code> <code>endloop;</code> <code>REGIÃO CRÍTICA;</code> <code>quer[1] := false;</code> <code>...</code>	P2: ... <code>quer[2] := true;</code> <code>vez := 1;</code> <code>loop</code> <code> exit when $\neg quer[1]$ or <code>vez = 1</code>;</code> <code>endloop;</code> <code>REGIÃO CRÍTICA;</code> <code>quer[2] := false;</code> <code>...</code>

- a) a solução proposta garante exclusão mútua? Justifique.
 b) a solução proposta garante o progresso dos processos (ausência de bloqueio)? Justifique.

4) Considere o algoritmo de Dijkstra, reproduzido a seguir, com algumas linhas numeradas:

variáveis globais

`quer, dentro : array[n] of boolean;`
`vez : integer;`

código do processo i

```

1   ...
2   quer[i] := true;
3 inicio: loop
4     dentro[i] := false;
5     if  $\neg quer[vez]$  then vez := i;
6     exit when vez = i;
7   endloop;
8   dentro[i] := true;
9   for k:=1 to n st k ≠ i
10    if dentro[k] then goto inicio;
11  endfor;
12  REGIÃO CRÍTICA;
13  quer[i] := false;
14  dentro[i] := false;

```

- a) Mostre uma história onde dois processos conseguem passar pelo loop definido nas linhas 3 a 7 do protocolo de entrada?
- b) Explique como o algoritmo de Dijkstra garante a exclusão mútua quando dois (ou mais) processos passam do loop citado no item anterior?
- c) É possível, para um processo que está iniciando agora o protocolo de entrada, passar na frente e entrar na Região Crítica, antes de um dos dois processos já passaram pelo loop citado no item "a"? Justifique.

Boa Prova!

Nome: Fernando Ferreira Páse Matrícula: 17459

- ✓ 1) a) Explique a razão de poder ocorrer um resultado inesperado em um programa com dois processos concorrentes que compartilham variáveis globais.
- ✓ b) O que pode ser feito para evitar a ocorrência de um resultado inesperado em um programa com dois processos concorrentes que compartilham variáveis globais?
- ✓ c) Considere o trecho de código, com dois processos concorrente P1 e P2, mostrado a seguir:

variáveis globais

x : integer initial 1; y : integer 2; z : integer 3;

P1: ...

x := y + z;

...

P2: ...

y := 5;

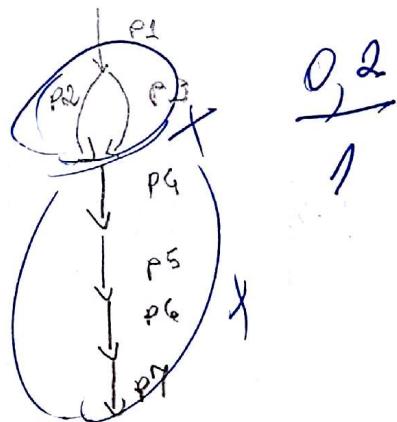
x := x + 1;

...

Liste todos os possíveis valores finais para as variáveis globais x, y e z.

- ✓ 2) a) Considere o trecho de código mostrado a seguir:

```
p1;  
a := fork();  
if (myself) then { p2; p3; quit }  
else { p4;  
b := fork();  
if (myself) then { p5; quit }  
else { p6; quit }  
join(b);  
};  
join(a);  
p7;  
...
```



Faça um desenho do grafo de precedência que representa o código acima.

- ✓ b) Caso seja possível, escreva a sequência de comandos S e P que representa o grafo de precedência obtido em "a". Caso não seja, justifique.

$S(p_1), S(S(p_2, p_3)), S(p_4), S(S(p_5, S(p_6, p_7))) \Big)$ 1

- 3 4) Considere o seguinte algoritmo para resolver o problema de exclusão mútua entre 2 processos P1 e P2, que executam em loop infinito, mostrado no verso dessa folha.

Variáveis globais:

quer : array[2] of boolean initial false; vez : integer initial 1;

P1: ...

quer[1] := true;
loop
 exit when not quer[2] or vez = 1;
endloop;
REGIÃO CRÍTICA;
quer[1] := false;
vez := 1;
...

P2: ...

quer[2] := true;
loop
 exit when not quer[1] or vez = 2;
endloop;
REGIÃO CRÍTICA;
quer[2] := false;
vez := 2;
...

- a) a solução proposta garante exclusão mútua? Justifique.
b) a solução proposta garante o progresso dos processos (ausência de bloqueio)? Justifique.
c) a solução possui a possibilidade de postergação indefinida (*starvation*)?

- 5 4) Considere o algoritmo de Dijkstra, reproduzido a seguir, com algumas linhas numeradas:

variáveis globais

quer, dentro : array[n] of boolean, vez : integer;

código do processo i

1 ...
2 *quer[i] := true;*
3 *inicio:* loop
4 *dentro[i] := false;*
5 if not *quer[vez]* then *vez := i;*
6 exit when *vez = i;*
7 endloop;
8 *dentro[i] := true;*
9 for *k:=1 to n st k ≠ i*
10 if *dentro[k]* then goto *inicio*;
11 endfor;
12 REGIÃO CRÍTICA;
13 *quer[i] := false;*
14 *dentro[i] := false;*

- a) Mostre uma história onde dois processos conseguem passar pelo loop definido nas linhas 3 a 7 do protocolo de entrada?
b) Explique como o algoritmo de Dijkstra garante a exclusão mútua quando dois (ou mais) processos passam do loop citado no item anterior?
c) É possível, para um processo que está iniciando agora o protocolo de entrada, passar na frente e entrar na Região Crítica, antes de um dos dois processos já passaram pelo loop citado no item "a"? Justifique.

Boa Prova!