Ciência da Computação **Lista de Exercícios**
Escola Politécnica da PUCPR **Monitores**
Prof. Alcides Calsavara **2016–2019**

Questão 1 .....

Considere um conjunto de processos $P_i, 1 \leq i \leq n$, que compartilham duas impressoras idênticas. Antes de usar uma impressora, P_i chama o procedimento `reserve(&name)`, o qual retorna o *nome* da impressora que foi reservada; enquanto não houver uma impressora disponível, não ocorre o retorno da chamada do procedimento. Depois de usar a impressora que lhe foi reservada, P_i deve chamar o procedimento `free(name)`, fornecendo o *nome* da impressora como parâmetro. Escreva o pseudo-código do monitor `PrinterPool` que implementa os procedimentos `reserve` e `free`.

Questão 2 .....

Considere o *Problema do Jantar dos Selvagens* de forma que o cozinheiro e cada um dos selvagens sejam processos definidos pelo pseudo-código abaixo.

```
processo Selvagem
{
   repita
   {
      sirva uma porção do pote no seu prato;
      coma;
   }
}

processo Cozinheiro
{
   repita
   {
      encha o pote com M porções de comida;
   }
}
```

Escreva o pseudo-código de um monitor denominado `Pote` com os procedimentos `servir` e `encher`, chamados pelos processos `Selvagem` e `Cozinheiro`, respectivamente. Deve-se observar que o cozinheiro só deve ser acordado quando o pote está completamente vazio.

Questão 3 .....

Um armazém tem capacidade para estocar uma quantidade limitada de maçãs. O armazém é acessado concorrentemente por muitos processos produtores e consumidores que, respectivamente, inserem e retiram maçãs do armazém. Quando um produtor deseja inserir uma quantidade de maçãs superior ao espaço disponível, deve aguardar até que o espaço necessário seja criado. Da mesma forma, quando um consumidor deseja remover mais maçãs do que há em estoque, deve aguardar até que o estoque atinja a quantidade desejada. O monitor `Armazem` definido pelo pseudo-código abaixo implementa essa semântica. No entanto, da forma como está, não garante a política FCFS (*First Come, First Served*). Por exemplo, suponha que o estoque é de 200 maçãs e um consumidor está esperando para retirar 300 maçãs. Se um segundo consumidor chega e deseja retirar uma quantidade inferior ou igual a 200 maçãs, esse é atendido, isto é, passa na frente do primeiro consumidor. Modifique o pseudo-código para que a política FCFS seja assegurada.

```
01   monitor Armazem {
02      int capacidade = 1000;
03      int estoque = 0;
04      cond insercao_efetuada;
05      cond retirada_efetuada;
06
07      procedimento inserir(int quantidade)
09      {
10         enquanto (quantidade > capacidade - estoque) wait( retirada_efetuada );
11         estoque = estoque + quantidade;
12         signalAll( insercao_efetuada );
13      }
14
15      procedimento retirar(int quantidade)
16      {
17         enquanto (estoque < quantidade) wait( insercao_efetuada );
18         estoque = estoque - quantidade;
19         signalAll( retirada_efetuada );
20      }
21   }
```

Questão 4 .....

Uma conta bancária é acessada simultaneamente por várias pessoas, ou seja, por processos concorrentes. Cada pessoa pode *depositar* ou *retirar* dinheiro da conta. O saldo atual da conta corresponde à diferença entre a soma de todos os depósitos e a soma de todas as retiradas até o momento atual. O saldo nunca pode ficar negativo. Uma operação de depósito nunca é suspensa, exceto por requisito de exclusão mútua. Mas, uma operação de retirada pode ter que aguardar até que haja saldo suficiente.

- Escreva o pseudo-código de um monitor para uma conta bancária com os procedimentos `depositar(valor)` e `retirar(valor)`, sem precisar garantir a política FCFS (*First Come, First Served*).
- Modifique a sua resposta tal que as chamadas de `retirar` garantam a política FCFS (*First Come, First Served*). Por exemplo, suponha que o saldo é de 200 reais e um cliente está esperando para retirar 300 reais. Se outro cliente chega, deve esperar, mesmo se desejar retirar 200 reais ou menos.

Questão 5

Considere as threads `A`, `B` e `C` definidas pelo pseudocódigo abaixo. As threads compartilham a variável `x` do tipo inteiro e os semáforos `mutex`, `i`, `j`, `d` e `e`. Inicialmente, o valor de `x` é 0, o semáforo `mutex` tem o valor 1, enquanto os demais semáforos têm o valor 0. A função `dormir(r,s)` suspende a execução da thread por um tempo qualquer entre `r` e `s` segundos.

```
Thread A(x, mutex, i, j, d, e):
    loop:
        i.liberar()
        j.esperar()
        dormir(1, 4)
        d.liberar()
        e.esperar()
        dormir(1, 4)
```

```
Thread B(x, mutex, i, j):
    loop:
        i.esperar()
        mutex.esperar()
        x++
        mutex.liberar()
        j.liberar()
        dormir(1, 10)
```

```
Thread C(x, mutex, d, e):
    loop:
        d.esperar()
        mutex.esperar()
        x--
        mutex.liberar()
        e.liberar()
        dormir(1, 10)
```

Escreva o pseudocódigo de um monitor `M` que faça a sincronização equivalente à implementada por semáforos nas threads `A`, `B` e `C`. O monitor M deverá encapsular a variável compartilhada `x` e ter os seguintes procedimentos:

- `alterar(booleano b)` : chamado pela thread `A`
- `incrementar()` : chamado pela thread `B`
- `decrementar()` : chamado pela thread `C`

O parâmetro b é usado para evitar que o monitor alterne a vez de B para C (ou de C para B) antes que B (ou C) tenha tempo de chamar o procedimento incrementar (ou decrementar). Dessa forma, as threads passam a compartilhar o monitor M e ficam definidas pelo seguinte pseudocódigo:

```
Thread A(M):
    b = falso
    loop:
        b = !b
        M.alternar(b)
        dormir(1, 4)
```

```
Thread B(M):
    loop:
        M.incrementar()
        dormir(1, 10)
```

```
Thread C(M):
    loop:
        M.decrementar()
        dormir(1, 10)
```

```
Monitor M {
   Variáveis permanentes:
       Inteiro x = 0




   procedimento alternar(booleano b)
   {









   }

   procedimento incrementar()
   {









   }

   procedimento decrementar()
   {









   }
}
```

Questão 6

Considere uma estrada de mão dupla que, em certo ponto, tem uma ponte estreita, de forma que não caibam dois carros lado a lado. Carros que estejam indo no mesmo sentido (norte ou sul) podem usar a ponte ao mesmo tempo, mas carros que estejam em sentidos opostos não, ou seja, um carro deve aguardar para entrar na ponte enquanto essa estiver ocupada com carros vindo em sentido oposto. Além disso, a cada momento, o número de carros passando sobre a ponte é, no máximo, 5 (cinco). Supondo que cada carro é um processo (ou thread), conforme definido pelo pseudocódigo abaixo, defina o pseudocódigo do monitor `Ponte` que implemente essa semântica.

```
Thread CarroNorte:
   Ponte.entrar_pelo_norte()
   # atravessa a ponte
   Ponte.sair_pelo_sul()

Thread CarroSul:
   Ponte.entrar_pelo_sul()
   # atravessa a ponte
   Ponte.sair_pelo_norte()
```

O monitor deverá conter os seguintes procedimentos para serem chamados pelos carros:

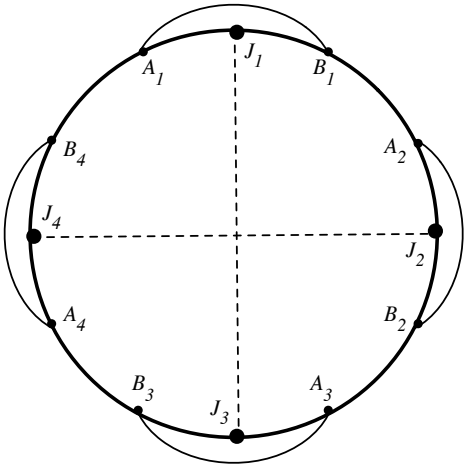
- `entrar_pelo_norte()`: chamado por um carro vindo do norte quando deseja entrar na ponte
- `entrar_pelo_sul()`: chamado por um carro vindo do sul quando deseja entrar na ponte
- `sair_pelo_norte()`: chamado por um carro vindo do sul quando termina de atravessar a ponte
- `sair_pelo_sul()`: chamado por um carro vindo do norte quando termina de atravessar a ponte

Questão 7 .....

Em uma ferrovia circular, movimentam-se dois trens em sentidos opostos: o trem X movimenta-se no sentido horário, enquanto o trem Y movimenta-se no sentido anti-horário. A ferrovia possui apenas um trilho, logo é necessário um mecanismo adequado para evitar a colisão entre os trens.

A figura abaixo representa a estrutura da ferrovia.

- A ferrovia é composta por quatro segmentos de igual comprimento. Os pontos J_1, J_2, J_3 e J_4 correspondem aos pontos de junção de segmentos. Assim, os segmentos são denominados (J_1, J_2), (J_2, J_3), (J_3, J_4) e (J_4, J_1).
- Existe um desvio (um trilho paralelo ao principal) para cada ponto de junção. Por exemplo, para o ponto J_1, existe o desvio (A_1, B_1). Um desvio tem comprimento suficiente para estacionar um trem, enquanto o outro passa no sentido oposto pelo trilho principal.



Um segmento pode ser ocupado por apenas um trem por vez, isto é, há exclusão mútua no acesso a cada segmento para que não ocorra colisão. Assim, quando um trem chega ao ponto de início de um desvio, tenta reservar o próximo segmento antes de prosseguir. Por exemplo, quando o trem X em movimento pelo segmento (J_4, J_1) chega ao ponto A_1, tenta reservar o segmento (J_1, J_2). Se esse segmento estiver livre, passa a ficar reservado para o trem X, o qual mantém o seu movimento pela trilha principal e, ao passar pelo ponto J_1, libera o segmento (J_4, J_1) e prossegue para o segmento reservado, tornando-o ocupado. Caso contrário, o trem X libera o segmento (J_4, J_1) e aguarda no desvio (A_1, B_1) até que o trem Y passe pelo ponto J_1; depois disso, o trem X reinicia o seu movimento, entrando no segmento (J_1, J_2) a partir do ponto B_1, tornando-o ocupado.

Implemente um monitor para gerenciar o acesso concorrente aos segmentos da ferrovia. Para fins de testes, pode-se assumir os seguintes parâmetros:

- Tempo para um trem percorrer um trecho completo de trilho principal onde não há desvio (por exemplo, de B_1 a A_2): de 4 a 12 segundos.
- Tempo para um trem percorrer um trecho de trilho principal entre um ponto de junção e o próximo ponto de desvio (por exemplo, de J_2 a B_2), ou vice-versa (por exemplo, de B_2 a J_2): de 1 a 4 segundos.
- Tempo para um trem estacionado num desvio reiniciar seu movimento e reentrar no trilho principal: de 2 a 6 segundos.
- Os pontos de início dos trens são B_1 para X e A_1 para Y.