UNIVERSIDADE FEDERAL DE VIÇOSA
INF310 – PROGRAMAÇÃO CONCORRENTE E DISTRIBUÍDA
Lista de Exercícios 2

1. Considere uma festa onde, devido a uma falha de organização, 3 convidados sentados em uma mesma mesa devem compartilhar um mesmo copo. Quando o copo é enchido por um garçom, um dos convidados pega o copo e bebe todo o seu conteúdo, deixando-o vazio para que o garçom encha novamente após dar uma volta.
   Apenas 3 tipos de bebidas são oferecidas na festa e cada um dos 3 convidados em uma mesa têm preferências distintas:
   - O convidado 1, mais "chato", bebe apenas a bebida do tipo 1.
   - O convidado 2, mais "de boa", aceita bebidas do tipo 1 e do tipo 2.
   - O convidado 3, "pé de cana", aceita qualquer tipo de bebida.

   Implemente um monitor que permita a sincronização entre uma thread representando o garçom e 3 threads representando os clientes de uma mesa. O monitor deve obedecer aos seguintes critérios:
   - O garçom traz uma bebida aleatória e, se o copo estiver vazio, enche o copo e vai embora. Caso contrário, ele espera que o conteúdo do mesmo seja consumido para servir a dose seguinte.
   - Nenhum convidado tem a preferência, ou seja, se a bebida 1 é trazida, beberá o que pegar o copo primeiro.
   - Não pode haver desperdício, assim, todas as doses servidas devem ser consumidas.
   - As funções servir e beber devem retornar apenas quando seu objetivo for cumprido.

   Um esboço para as threads garçom e convidado é exibido abaixo:

```
garçom:
    loop {
        int b=rand()%3+1; //busca uma bebida aleatória: 1, 2 ou 3
        bebida.serve(b);  //serve
    }

convidado c:
    loop {
        bebida.bebe(c);   //parâmetro indica o número do convidado
    }
```

2. Por que um monitor é similar a uma estrutura abstrata de dados?

3. Em capítulos anteriores foi visto que duas variações para o uso de std::mutex são através de uma associação do mesmo a um std::unique_lock ou a um std::lock_guard, sendo que o std::lock_guard é uma estrutura mais simples. Por que uma std::condition_variable deve estar associada obrigatoriamente a um std::unique_lock? Faria sentido que o mesmo fosse implementado usando std::lock_guard?

4. Por que o mecanismo de monitor é considerado de nível mais alto que os semáforos?

5. As operações Down (ou wait) e Up (ou post) do semáforo devem ser atômicas? Justifique.

6. Em alguns aspectos, as instruções Up/Down realizada sobre semáforos se assemelham às instruções block/wakeup realizadas sobre threads.
   a) Destaque pelo menos 3 semelhanças.
   b) Qual a vantagem no uso de semáforos sobre diretivas do tipo block/wakeup?

7. Considerando que as threads A e B são concorrentes, atualize o código abaixo utilizando semáforos para garantir que, a qualquer momento, $0 \leq x \leq 10$.
   Obs: não é permitido inserir ou alterar as instruções que alterem o valor de x.

```
int x = 0; //variável global

void thread_A() {
    while(true) {
        x = x+1;
    }
}

void thread_B() {
    while(true) {
        x = x-1;
    }
}
```

8. 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 pega 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 (6 latas de cerveja ou 10 latas de refrigerante) e coloca as latinhas no balcão. Implemente as threads que descrevem o comportamento do atendente e dos dois tipos de jovens utilizando semárofos para a sincronização das mesmas.