Departamento de Computação e Matemática (DCM)
Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto (FFCLRP)
Universidade de São Paulo (USP)

Terceira Lista de Exercícios de Sistemas Operacionais

1) Implemente um semáforo contador em Java utilizando as primitivas de sincronização da linguagem. Considere que esta classe implemente os métodos abaixo. Modifique a assinatura dos métodos se necessário.

```
public void down();
public void up();
```

2) Suponha que uma dada instituição comercial tenha um único toalete para seus clientes. Esta instituição adota a seguinte política de utilização deste toalete: quando uma mulher estiver ocupando este toalete, outra mulher poderá entrar, mas um homem não. De forma análoga, quando um homem estiver ocupando o toalete, outro home poderá entrar, mas uma mulher não. Um sinal com um marcador deslizante, na porta do toalete indica em qual dos três estados o banheiro se encontra: vazio; com mulher; com homem. Inicialmente o banheiro encontra-se vazio. Implemente a classe Toalete utilizando as primitivas de sincronização de Java e/ou classes do pacote *java.util.concurrent*. Considere que esta classe implemente os métodos abaixo. Modifique a assinatura dos métodos se necessário. Implemente também um número variável de threads representando os diferentes tipos de clientes (homens e mulheres) e simule a utilização da classe Toalete. Cada cliente (homem ou mulher) deve esperar um período de tempo aleatório antes tentar entrar no toalete. De forma análoga, cada cliente que tenha entrado no toalete deve esperar um período aleatório antes de sair.

```
public void mulherQuerEntrar(String name);
public void homemQuerEntrar(String name);
public void mulherSaiToalete(String name);
public void homemSaiToalete(String name);
```

3) Um spooler de impressão é um processo responsável por gerenciar todos os *jobs* (tarefas) contendo requisições para a impressão de um arquivo. Este processo recebe as requisições de impressão encaminhas por diferentes processos, servindo-as uma por vez, isto é, imprimindo um arquivo de cada vez. Caso o spooler esteja servindo um *job*, um processo requisitante deve aguardar sua vez. Implemente uma classe Java que simule um *spooler* de impressão utilizando o semáforo desenvolvido no Exercício 1. Esta classe deve implementar os seguintes métodos:

```
public class PrintSpooler {
    // construtor classe - recebe como parâmetro o nome do arquivo de spool
    public PrintSpooler(String spoolFile);

    // método utilizado para abrir o arquivo de spool
    public boolean openPrintSpooler();

    // método utilizado para imprimir um job
    public void printJob(String jobName);

    // método utilizado para fechar o arquivo de spool
    public void closePrintSpooler();
}
```

Todas as solicitações recebidas (arquivos texto apenas), devem ser salvas ("impressas") em um mesmo arquivo de saída ("arquivo de spool") na ordem em que foram recebidas. A impressão dos *jobs* ocorre de forma simulada, mediante a cópia linha por linha do arquivo a ser impresso para o arquivo de *spool*. Implemente também um número variável de threads responsáveis pela submissão de uma requisição de impressão. Cada um destes threads deve esperar um período de tempo aleatório antes de solicitar a impressão de um *job*. Antes que estes threads possam encaminhar estas solicitações, o arquivo de *spool* deve ser aberto e após a conclusão de todas as impressões (encerramento de todos os threads), o arquivo de *spool* deve ser fechado.

4) Atualmente, está se tornando cada vez mais comum que pais permitam que filhos menores de idade tenham seus próprios cartões de crédito. Os cartões adicionais gerados para os filhos permitem que estes compartilhem o limite de crédito estabelecido para o cartão principal, que deve ser administrado com cuidado pelos pais. Escreva um programa em Java que implemente uma classe *CreditCard*, que deve possuir o limite atual dos cartões como um atributo e dois métodos para a manipulação do limite: *boolean withdraw* para lançar uma despesa no cartão,

reduzindo como consequência o limite disponível, e *double getBalance* para obter o valor atual do limite disponível. Implemente também uma classe *CardPayment* utilizando threads que irá representar a realização de um conjunto de requisições de pagamento que deverão ser realizadas junto à classe *CreditCard*. Garanta que o limite nunca fique negativo por meio das primitivas de sincronização. Esta classe de receber como parâmetros de inicialização, uma referência para a classe *CreditCard* e um array de contendo as diferentes requisições de pagamento a serem realizadas. Espere um tempo aleatório antes de fazer cada requisição. Finalmente, implemente uma classe testar as classes *CreditCard* e *CardPayment*.