Lab 3 – Semáforos: O Barbeiro Dorminhoco

• Se não há clientes, o barbeiro adormece;

• Se a cadeira do barbeiro estiver livre, um cliente pode ser atendido imediatamente;

• O cliente espera pelo barbeiro se houver uma cadeira de espera vazia.

• Se não tiver onde sentar, o cliente vai embora ...

Ver em http://www.ic.unicamp.br/~islene/mc514/barbeiro/barbeiro.pdf

Outra descrição: O Problema do Barbeiro Dorminhoco (com threads)

http://ces33.blogspot.com.br/2009/05/o-problema-do-barbeiro-dorminhoco-com_07.html

Em ciência da computação, o problema do barbeiro dorminhoco é um problema clássico de comunicação inter-threads e sincronização entre múltiplas threads.

O problema é análogo a manter o barbeiro ocupado enquanto há clientes, e descansando quando não há nenhum (fazendo isso de uma maneira ordenada).

O barbeiro e seus clientes correspondem as threads mencionadas acima.

O problema: Na barbearia há um barbeiro, uma cadeira de barbeiro e *n* cadeiras para eventuais clientes esperarem a vez. Quando não há clientes, o barbeiro senta-se na cadeira de barbeiro e cai no sono. Quando chega um cliente, ele precisa acordar o barbeiro. Se outros clientes chegarem enquanto o barbeiro estiver cortando o cabelo de um cliente, eles se sentarão (se houver cadeiras vazias) ou sairão da barbearia (se todas as cadeiras estiverem ocupadas).

O problema é programar o barbeiro e os clientes sem cair em **condições de disputa** (condições de corrida) (*race conditions*). Condições de disputa são situações em que duas ou mais threads (ou processos) estão trabalhando juntas e podem compartilhar algum armazenamento comum que cada uma pode ler e gravar. O resultado final depende de quem executa precisamente quando. Os resultados da maioria dos programas são bons, mas, de vez em quando, acontece algo estranho e inconsistente.

Esse problema é semelhante a situações com várias filas, como uma mesa de atendimento de telemarketing com diversos atendentes e com um sistema computadorizado de chamadas em espera, atendendo a um número limitado de chamadas que chegam.
Uma solução usa três semáforos: *customers*, que conta os clientes à espera de atendimento (exceto o cliente que está na cadeira de barbeiro, que não está esperando);

barbers, o número de barbeiros (0 ou 1) que estão ociosos à espera de clientes, e *mutex*, que é usado para exclusão mútua. Precisamos ainda de uma variável, *waiting*, que também conta os clientes à espera de atendimento. É essencialmente uma cópia de *customers*. A razão de se ter *waiting* é que não há uma maneira de ler o valor atual do semáforo. Nessa solução, um cliente que entra na barbearia deve contar o número de clientes à espera de atendimento. Se este for menor que o número de cadeiras, ele ficará; do contrário, ele sairá.

Nesta solução, quando chega de manhã para trabalhar, o barbeiro executa o método *barber*, que o leva a bloquear sobre o semáforo *customers*, que inicialmente está em 0. O barbeiro então vai dormir, e permanece dormindo até que o primeiro cliente apareça. Quando chega, o cliente executa *customer* e inicia obtendo *mutex* para entrar em uma região crítica. Se um outro cliente chega logo em seguida, o segundo nada pode fazer até que o primeiro libere o *mutex*. O cliente então verifica se o número de clientes à espera é menor que o número de cadeiras. Se não for, ele liberará o *mutex* e sairá sem cortar o cabelo. Se houver uma cadeira disponível, o cliente incrementará a variável inteira *waiting*. Ele faz então um up no semáforo *customers*, que acorda o barbeiro. Nesse ponto, o cliente e o barbeiro estão ambos acordados. Quando o cliente libera *mutex*, o barbeiro o pega, faz alguma limpeza e começa a cortar o cabelo. Quando termina o corte de cabelo, o cliente sai do procedimento e deixa a barbearia. Diferente de nossos exemplos anteriores, não há um laço para o cliente porque para cada um deles terá apenas um corte de cabelo. O barbeiro, contudo, contém um laço para tentar obter o próximo cliente. Se houver algum outro cliente, um novo corte de cabelo será iniciado. Do contrário, o barbeiro cairá no sono.

Implemente em Java, usando **semáforos**.