MAC0438 – Programação concorrente – 1s2015
EP1
Data de entrega: 22/4/2015

Prof. Daniel Macêdo Batista

1 Problema

Uma das várias modalidades de ciclismo realizada em velódromos é a *Omnium*, uma competição na qual os ciclistas vão conquistando pontos em diversas sub-modalidades de provas. Uma dessas sub-modalidades é a corrida por eliminação. O objetivo deste EP será simular a sub-modalidade de corrida por eliminação no modo *miss-and-out*. Vamos considerar um cenário em que essa sub-modalidade é a única prova que vai definir os vencedores de medalhas de ouro, prata e bronze em uma competição.

Na corrida por eliminação *miss-and-out* tradicional, os ciclistas iniciam a prova enfileirados e, a partir da largada, a cada duas voltas concluídas no velódromo, o último ciclista que passar pela linha de chegada é desclassificado. Assim, a partir do momento em que houverem 3 ciclistas, o primeiro destes a ser eliminado ganha a medalha de bronze, o segundo deles a ser eliminado ganha a medalha de prata e o último ciclista que sobrar na pista, aquele que não foi eliminado nenhuma vez, ganha a medalha de ouro.

A sua tarefa neste EP é simular a corrida. A simulação deve considerar que a corrida é em um velódromo com d metros e que n ciclistas começam a prova ($d > 249$ e $n > 3$). A posição inicial de cada ciclista deve ser definida de forma aleatória. A qualquer momento, no máximo, apenas 4 ciclistas podem estar lado a lado em cada ponto da pista. Considere que cada ciclista ocupa exatamente 1 metro da pista e que a quantidade de ciclistas que começa a prova é no máximo $\lceil d/2 \rceil$.

2 Requisitos

2.1 Linguagem

O simulador deve ser escrito em C e toda a gerência de threads deve ser feita utilizando POSIX threads (pthreads). Programas escritos em outra linguagem ou utilizando alguma biblioteca extra para gerenciar as threads terão nota zero.

Informações sobre como programar utilizando pthreads podem ser encontradas na seção 4.6 do livro do Andrews (basta ler até a subseção 4.6.1 inteira), na página da wikipedia em `http://en.wikipedia.org/wiki/POSIX_Threads` ou no tutorial da IBM disponível em `http://www.ibm.com/developerworks/library/l-posix1/index.html`.

2.2 Detalhes do código

Seu simulador deve criar n threads "`ciclista`". Seu código terá duas opções de execução. Na primeira opção todos os ciclistas conseguem pedalar exatamente na mesma velocidade de 50Km/h (1m a cada

72ms) durante toda a prova. Na segunda opção, todos os ciclistas fazem a primeira volta a 25Km/h mas a partir da segunda volta cada um dos ciclistas define suas velocidades aleatoriamente, para realizar a volta seguinte, como sendo 25 ou 50Km/h. Desconsidere a aceleração necessária para mudar a velocidade.

Seu código deve possuir um vetor compartilhado "`pista`" que tem um tamanho igual a d. Cada posição do vetor corresponde portanto a 1 metro da pista. Em um dado instante de tempo, a posição i da `pista` deve possuir os identificadores de todos os ciclistas que estão naquele trecho. A simulação do seu código deve simular a corrida em intervalos de 72ms. Cada thread `ciclista` tem a obrigação de escrever seu identificador na posição correta do vetor `pista` a cada momento em que ele entra em um novo trecho de 1m, e de remover seu identificador da posição referente ao trecho que ele acabou de sair. Como é possível perceber, cada posição do vetor corresponde a uma variável compartilhada que deve ter seu acesso controlado. Note que apesar de ter sorteado a velocidade de 50Km/h para a próxima volta, pode ser que um ciclista não consiga de fato pedalar a essa velocidade, por exemplo, caso hajam 4 ciclistas na frente dele pedalando a 25Km/h.

Assim como no mundo real, ciclistas podem "quebrar" durante a prova e desistirem. Considere que a cada 4 voltas, há a chance de 1% de que um ciclista quebre. Qual ciclista, pode ser definido de forma aleatória. Caso algum ciclista quebre, essa informação deve ser exibida na tela no momento exato em que ele quebrou. A volta em que ele estava, a posição em que ele estava nessa volta e o identificador dele deve ser informado. Entretanto, quando houverem apenas 3 ciclistas na pista, a probabilidade de quebra deixa de existir.

Toda vez que um ciclista for eliminado ou quebrar, a thread dele deve ser destruída.

A saída do seu programa deve ser um relatório informando a cada volta completada, os 3 **últimos** colocados, ou seja, aquele que foi eliminado e os outros 2 que ficaram imediatamente antes dele. Ao término da corrida (depois que todos os ciclistas passarem pela linha de chegada), a classificação final deve ser informada, destacando os vencedores das medalhas. Ciclistas que quebrarem devem ser identificados como tendo quebrado e a posição deles depende da volta em que eles quebraram (Ciclistas que completam mais voltas ficam na frente de ciclistas que completam menos voltas). Se dois ou mais ciclistas empatarem na linha de chegada, o desempate deve ser feito de forma aleatória. Seu programa deve ainda permitir uma opção de *debug* na qual a cada 14,4 segundos deve ser exibido na tela a volta em que cada ciclista está e a posição dele naquela volta atual.

Não há um formato padrão para a saída do seu programa. Basta que ela informe tudo que foi solicitado no parágrafo anterior.

Com relação à entrada, seu simulador deve receber como argumentos nesta ordem:

```
d n [v|u]
```

O `v` deve ser usado para definir simulações com velocidades aleatórias a cada volta. O `u` deve ser usado para definir simulações com velocidades uniformes de 50Km/h.

Não há necessidade de validar a entrada.

Lembre que seu programa é um simulador. Ou seja, a simulação não precisa levar o mesmo tempo que uma corrida de verdade levaria.

3 Sobre a entrega

Você deve entregar um arquivo .tar.gz contendo os seguintes itens:

- fonte, Makefile (ou similar), arquivo LEIAME;

- relatório em .pdf mostrando a saída do seu código para pelo menos 2 exemplos de entrada que você tenha criado, um com velocidade uniforme e um com velocidade variável.

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep1-membros_da_equipe. Por exemplo: ep1-joao-maria.

A entrega do .tar.gz deve ser feita através do PACA.

O EP pode ser feito individualmente ou em dupla.