MAC0422 – Sistemas Operacionais – 2s2016
EP2
Data de entrega: 17/10/2016 até 8:00

Prof. Daniel Macêdo Batista

1 Problema

Uma das várias modalidades de ciclismo realizada em velódromos é a perseguição por equipe [1]. O objetivo deste EP será simular essa modalidade.

Na perseguição por equipe, cada equipe, contendo 4 ciclistas, inicia a prova em lados opostos do velódromo. Os 4 ciclistas de cada equipe ficam lado-a-lado na largada e logo depois, para obter uma vantagem aerodinâmica, ficam enfileirados com mudanças regulares das posições de modo que o primeiro ciclista mude de tempos em tempos e os demais, que ficam atrás, possam economizar energia. A equipe que vence a prova é aquela cujo terceiro ciclista, após 4Km (16 voltas num velódromo de 250 metros) cruzar primeiro a linha de chegada (que é relativa à linha de largada da equipe). A prova também termina se o terceiro ciclista de uma equipe ultrapassar o terceiro ciclista da outra equipe.

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 em cada equipe ($d > 249$ e $n > 4$). A qualquer momento, com exceção do instante da largada, no máximo, apenas 2 ciclistas podem estar lado a lado em cada ponto da pista, independente das suas equipes. Considere que cada ciclista ocupa exatamente 1 metro da pista e que a quantidade de ciclistas de cada equipe é no máximo $\lceil d/4 \rceil$. A quantidade de voltas será sempre 16 independente do tamanho do velódromo.

2 Requisitos

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`.

Seu simulador deve criar $2 \times n$ threads "`ciclista`". Seu código terá duas opções de execução. Na primeira opção todos os ciclistas conseguem pedalar exatamente na mesma velocidade de 60Km/h (1m a cada 60ms) durante toda a prova. Na segunda opção, todos os ciclistas fazem a primeira volta a 30Km/h mas a partir da segunda volta cada um dos ciclistas define suas velocidades aleatoriamente, para

[1] `https://www.youtube.com/watch?v=TD2Vg_S0gzg` `https://en.wikipedia.org/wiki/Team_pursuit`

realizar a volta seguinte, como sendo 30 ou 60Km/h, com 50% de chance de escolher 60Km/h e 50% de chance de escolher 30Km/h. Se a velocidade sorteada para um ciclista for de 30Km/h, **todos** os ciclistas companheiros de equipe que estiverem atrás dele, naquela volta, devem pedalar a 30Km/h, independente do valor que foi sorteado para eles. Desconsidere a aceleração necessária para mudar a velocidade. Na volta seguinte, se o ciclista lento continuar lento, ele pode ser ultrapassado pelos seus companheiros de equipe caso haja algum ciclista atrás dele com velocidade de 60Km/h.

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 60ms. 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 60Km/h para a próxima volta, pode ser que um ciclista não consiga de fato pedalar a essa velocidade, por exemplo, caso hajam 2 ciclistas na frente dele pedalando a 30Km/h.

Assim como no mundo real, ciclistas podem "quebrar" durante a prova e desistirem. Considere que a cada 4 voltas, há a chance de 10% 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 em uma equipe, a probabilidade de quebra para todos os ciclistas daquela equipe deixa de existir.

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

A saída do seu programa deve ser um relatório informando a cada volta completada pelo terceiro ciclista de uma equipe, todos os 3 primeiros ciclistas daquela equipe, o número da volta, e o instante de tempo que esse terceiro ciclista passou pela linha de chegada (considere que a simulação começa no instante de tempo 0). Ao término da corrida (depois que todos os ciclistas passarem pela linha de chegada), deve ser informada qual equipe chegou em primeiro e qual chegou em segundo, ou se houve empate. A ordem final de chegada de todos os ciclistas, considerando a ordenação da sua equipe e o instante de tempo que cada ciclista cruzou a linha de chegada também deve ser impresso na saída. Ciclistas que quebrarem devem ser identificados como tendo quebrado e, ao invés de mostrar as suas posições, deve ser informada a volta em que eles quebraram. Seu programa deve ainda permitir uma opção de *debug* que informa a cada 60ms o status de cada posição da pista, ou seja, o identificador do(s) ciclista(s) naquela posição ou a informação de que não há nenhum ciclista ali.

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 60Km/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

Deve ser entregue um arquivo .tar.gz contendo os itens listados abaixo. EPs que não contenham **todos** os itens abaixo **exatamente como pedido** terão nota ZERO e não serão corrigidos. **A depender da qualidade do conteúdo entregue**, mesmo que o EP seja entregue, **ele pode ser considerado como não entregue, o que mudará o cálculo da média final**:

- código-fonte em C;
- arquivo LEIAME **em formato texto puro** explicando como compilar e executar o simulador;
- Makefile ou similar para facilitar a compilação do código-fonte;
- apresentação **em .pdf** para ser apresentada em no máximo 15 minutos resumindo os resultados obtidos com diversos experimentos e explicando detalhes da implementação do controle de acesso à pista.

Os resultados devem ser exibidos com gráficos que facilitem observar qual foi o impacto no uso de memória e no tempo de execução do programa ao aumentar tanto o tempo de simulação (com o aumento da pista) quanto o número de threads (com o aumento do número de ciclistas). Considere 3 tamanhos de pista (pequena, média e grande) e 3 quantidades de ciclistas (poucos, normal e muitos). Apresente os resultados obtidos com gráficos em barra. Cada valor a ser apresentado nos gráficos deve possuir média e intervalo de confiança de 30 medições com nível de 95%. Os resultados dos experimentos valem 3,0 pontos.

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep2-membros_da_equipe. Por exemplo: ep2-joao-maria. EPs que não gerem um diretório ou que gerem o diretório com o nome errado perderão 1,0 ponto.

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

O EP pode ser feito individualmente ou em dupla.

Obs.: não inclua no .tar.gz itens que não foram pedidos neste enunciado. Relatórios, saídas para diversas execuções e entradas usadas para testar o programa não devem ser entregues. No máximo um resumo sobre as entradas usadas pode ser colocado na apresentação, caso isso não ocupe muito espaço.