# MAC0422 – Sistemas Operacionais – 2s2017
# EP2
# Data de entrega: 16/10/2017 até 8:00

Prof. Daniel Macêdo Batista

## 1 Problema

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

Na corrida por pontos, ciclistas iniciam a prova ao mesmo tempo no mesmo lado do velódromo. A prova geralmente dura 160 voltas em um velódromo de 250m, no caso da prova masculina, e 100 voltas em um velódromo de 250m, no caso da prova feminina. A pontuação é definida em "*sprints*" que acontecem a cada 10 voltas, com 5, 3, 2 e 1 ponto(s) sendo atribuídos às 4 primeiras colocações em cada sprint. Ao término da prova, a pontuação acumulada define as colocações de cada ciclista. Além da pontuação definida a cada "*sprint*", ciclistas que conseguem completar 1 volta sobre todos(as) os(as) outros(as), ganham 20 pontos.

A simulação deve considerar que a corrida é em um velódromo com $d$ metros, que $n$ ciclistas começam a prova e que $v$ voltas serão realizadas ($d > 249$, $5 < n \leq 5 \times d$ e $v$ é múltiplo de 20). A qualquer momento, no máximo, apenas 10 ciclistas podem estar lado a lado em cada ponto da pista. Considere que cada ciclista ocupa exatamente 1 metro da pista.

## 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 $n$ threads "`ciclista`" iguais. Todos os ciclistas fazem a primeira volta a 30Km/h (1m a cada 120ms) mas a partir da segunda volta cada um dos ciclistas define suas velocidades aleatoriamente, para realizar a volta atual, como sendo 30 ou 60Km/h (1m a cada 60ms). Caso a volta anterior tenha sido feita a 30Km/h, o sorteio é feito com 70% de chance de escolher 60Km/h e 30% de chance de escolher 30Km/h. Caso a volta anterior tenha sido feita a 60Km/h, o sorteio é feito com 50% de chance de escolher 30Km/h e 50% de chance de escolher 60Km/h. Se a velocidade sorteada

[1] `https://www.youtube.com/watch?v=fAGb0bWtjIQ` `https://www.youtube.com/watch?v=WUEL2okSiQ0` `https://en.wikipedia.org/wiki/Points_race`

para um ciclista for de 30Km/h, **todos** os ciclistas que estiverem imediatamente atrás dele na mesma linha que ele, devem pedalar a 30Km/h, independente do valor que foi sorteado para eles, caso não seja possível ultrapassar. Ultrapassagens podem ser realizadas caso haja espaço em alguma pista mais externa (ultrapassagens só podem ser realizadas usando as pistas externas). Desconsidere a aceleração necessária para mudar de velocidade. Considere ainda que nas 2 últimas voltas há a chance de 10% de 1 ciclista aleatório qualquer e respeitando as regras de ultrapassagem, fazer essas 2 últimas voltas a 90Km/h (1m a cada 40ms).

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 até as duas últimas voltas. A partir das duas últimas voltas, caso algum ciclista tenha sido sorteado para pedalar a 90Km/h, a simulação deve passar a simular a corrida em intervalos de 20ms. 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, pode ser que um ciclista não consiga de fato pedalar a essa velocidade, por exemplo, caso ele esteja na linha mais externa da pista com um ciclista pedalando a 30Km/h imediatamente na frente.

Assim como no mundo real, ciclistas podem "quebrar" durante a prova e desistirem. Considere que a cada vez que um ciclista completa múltiplos de 15 voltas, ele tem a chance de 1% de quebrar. 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 na classificação por pontos e o identificador dele devem ser informados. Entretanto, se houverem apenas 5 ciclistas na prova, a probabilidade de quebra para todos 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, as posições de todos os ciclistas naquela volta. Em voltas múltiplas de 10, deve também ser informada a pontuação acumulada da prova, em ordem decrescente. Ao término da corrida (depois que todos os ciclistas passarem pela linha de chegada), a pontuação final de todos os ciclistas, e o instante de tempo que cada um cruzou a linha de chegada também deve ser impresso na saída (considere que a simulação começa no instante de tempo zero). Ciclistas que quebrarem devem ser identificados nessa lista final como tendo quebrado e, ao invés de mostrar as suas colocações, deve ser informada a volta em que eles quebraram. Seu programa deve ainda permitir uma opção de *debug* que informa a cada 60ms (e 20ms nas duas últimas voltas, caso alguém pedale a 90Km/h) 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 os três números inteiros:

```
d n v
```

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 e/ou do número de voltas) quanto o número de threads (com o aumento do número de ciclistas). Considere 3 tamanhos de pista/voltas (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%. Discuta se os resultados saíram conforme o esperado. Os gráficos e a discussão dos 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 1,5 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.