MAC0422 – Sistemas Operacionais – 1s2025 EP2 (Individual)

Data de entrega: 13/5/2025 até 13:00:00

Prof. Daniel Macêdo Batista

1 Problema

Uma das várias modalidades de ciclismo realizada em velódromos é a corrida por eliminação ¹. O objetivo deste EP será simular essa modalidade considerando o critério *Miss and out*. Todo o código deve ser escrito em C para ser executado no GNU/Linux.

Na corrida por eliminação *Miss and out*, ciclistas iniciam a prova ao mesmo tempo no mesmo lado do velódromo. A cada 2 voltas, o ciclista que completar a última volta na última posição sai da corrida e é eliminado. A prova termina quando sobrar apenas um ciclista, que é o campeão.

A simulação deve considerar que a corrida é em um velódromo com d metros e que k ciclistas começam a prova ($100 \le d \le 2500$ e $5 \le k \le 5 \times d$)). 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 do comprimento da pista.

2 Requisitos

Toda a gerência de threads no simulador deve ser feita utilizando POSIX threads (pthreads) na linguagem C. Programas escritos em outra linguagem ou utilizando alguma biblioteca extra para gerenciar as threads terão nota ZERO.

Seu simulador deve criar k threads <code>ciclista</code> iguais. Os ciclistas largam em fila ordenados aleatoriamente com no máximo 5 ciclistas lado a lado em cada posição. 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 75% de chance de escolher 60Km/h e 25% de chance de escolher 30Km/h. Caso a volta anterior tenha sido feita a 60Km/h, o sorteio é feito com 45% de chance de escolher 60Km/h e 55% de chance de escolher 30Km/h. Os sorteios das velocidades deve ser feito de forma autônoma por cada thread <code>ciclista</code> e cada uma delas deve ser responsável por controlar quando é hora do ciclista avançar uma posição na pista. Não pode haver uma entidade central fazendo as definições das velocidades e nem controlando qual ciclista deve se mover em um dado instante de forma sequencial. Todas as threads dos ciclistas devem ser capazes de rodar em paralelo, respeitando as restrições devido às seções críticas no código. As únicas tarefas de uma entidade central são fazer as impressões na tela, controlar o relógio global e atualizar as colocações dos ciclistas considerando quem

¹https://www.youtube.com/watch?v=9SPRxRNW9xc O jogo Fall Guys também considera uma modalidade similar https://www.youtube.com/watch?v=Wj3dUvGLjNQ

ainda esteja na corrida e quem saiu da corrida. De forma bem genérica, o algoritmo a ser implementado seria este abaixo:

```
enquanto (há no máximo 2 ciclistas na pista):
faça todos os ciclistas andarem 1 passo de forma concorrente;
destrua as threads dos ciclistas que precisam ser destruídas;
avance o relógio em 60ms;
imprima as informações na tela;
```

Um algoritmo como este abaixo está incorreto pois ele não permite que os ciclistas lidem com situações que tornem necessário lidar com seções críticas:

```
# ERRADO!!!!
enquanto (há no máximo 2 ciclistas na pista):
    para (cada ciclista i)
        faça o ciclista i andar 1 passo;
    destrua as threads dos ciclistas que precisam ser destruídas;
    avance o relógio em 60ms;
    imprima as informações na tela;
# ERRADO!!!!
```

Se a velocidade sorteada 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 e desconsidere o tempo necessário para o ciclista se mover para pistas diferentes dentro de uma mesma metragem da pista. Ou seja, ele pode subir ou descer no velódromo gastando 0 milissegundos. Considere que tempo é gasto apenas quando o ciclista se move para frente.

Seu código deve possuir um vetor circular 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, ou seja, faz sentido criar uma matriz para representar a pista. 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. Não é permitido ter uma entidade central no código que faça essas movimentações dos ciclistas. Como é possível perceber, cada posição do vetor (matriz) 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 pista mais externa com um ciclista pedalando a 30Km/h imediatamente na frente.

Note que apesar da pista ser uma única variável, colocar um único semáforo para controlar o acesso a ela é uma péssima solução pois não necessariamente todos os ciclistas estarão tentando escrever na mesma posição da pista (i.e.: na mesma posição de memória) ao mesmo tempo. Mesmo entendendo que isso é uma péssima solução, você terá que implementar uma versão do seu simulador que adote essa abordagem, que vamos chamar de ingênua. Uma outra abordagem é usar mais semáforos para controlar o acesso à pista, por exemplo um semáforo para cada posição da matriz ou um semáforo para cada coluna da matriz, etc... Pense em qual seria a melhor forma em termos de desempenho sem desrespeitar o

controle de acesso às seções críticas do programa. Vamos chamar essa segunda abordagem de eficiente, que você também terá que implementar.

Assim como no mundo real, ciclistas podem "quebrar" durante a prova e desistirem. Considere que a cada vez que um ciclista completa múltiplos de 5 voltas, ele tem a chance de 10% 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 e o identificador dele devem ser informados. O sorteio para definir as quebras deve ser feito de forma autônoma por cada thread ciclista. Não pode haver uma entidade central tomando essa decisão.

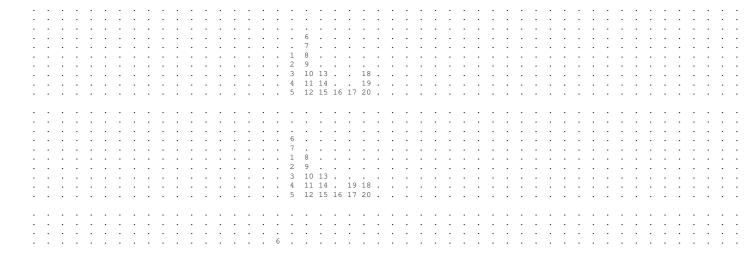
Toda vez que um ciclista quebrar, a thread dele deve ser destruída. O mesmo deve acontecer quando o ciclista terminar sua participação na prova. Em casos onde mais de um ciclista passe pela linha de chegada na última posição nas voltas pares, o ciclista a ser removido deve ser aleatoriamente sorteado dentre os que passaram ao mesmo tempo. Os desempates e a destruição das threads devem ser feitos por uma entidade central no código.

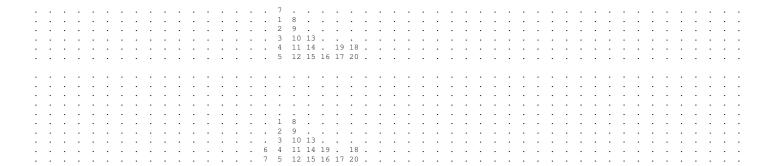
A saída do seu programa deve ser um relatório impresso na saída padrão (stdout) informando a cada volta completada, as posições de todos os ciclistas naquela volta. Ao término da corrida (depois que o último ciclista, dentre os dois finais, passar pela linha de chegada na última volta par) também devem ser impressos na saída o ranqueamento final de todos os ciclistas e o instante de tempo que cada um cruzou a linha de chegada pela última vez. 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. Não há um formato padrão para a saída do seu programa. Basta que ela informe tudo que foi solicitado aqui neste parágrafo.

Com relação à entrada, seu simulador deve receber como argumentos de linha de comando, nesta ordem, os dois números inteiros d e k sem necessidade de validá-los, seguidos da abordagem de controle de acesso à pista: i para a abordagem ingênua e e para a abordagem eficiente.

d k < i | e >

Seu programa deve ainda permitir um argumento final —debug opcional para que ele informe, na saída de erro (stderr), 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. Abaixo há um exemplo dessa saída para uma pista de 100 metros com 20 ciclistas por 4 iterações consecutivas. Para a simulação ficar mais próxima do mundo real, os ciclistas movem-se da direita para a esquerda, simulando o movimento anti-horário para um observador no centro do velódromo direcionando o seu olhar para o pelotão de ciclistas. Por limitação do tamanho da página, apenas 50 posições da pista estão sendo mostradas:





Nesse caso do debug, a saída com o relatório de cada volta não deve ser impresso. Apenas o relatório final, na saída padrão, ao término da corrida.

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 em termos de formato e de nomes 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:

- 1 único arquivo ep 2. c e 1 único arquivo ep 2. h com a implementação do simulador;
- 1 único arquivo LEIAME em formato texto puro explicando como compilar e executar o simulador;
- 1 único arquivo Makefile para gerar o executável;
- 1 único arquivo slides-ep2.pdf com no máximo 12 slides (contando todos os slides, inclusive capa, roteiro e referências, se houverem) resumindo os resultados obtidos com diversos experimentos e explicando detalhes de como foi feito o controle de acesso à pista na abordagem eficiente. Esses slides não serão apresentados. Eles devem ser preparados supondo que você teria que apresentá-los. Não coloque nos slides conteúdo que não foi pedido. Por exemplo, não precisa repetir o enunciado e nem explicar o controle de acesso na abordagem ingênua.

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) para as duas abordagens de controle de acesso à pista. 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 confiança de 95%. Discuta se os resultados saíram conforme o esperado em todos os aspectos: tamanho da pista, número de ciclistas e tipo de abordagem. Os gráficos e a discussão dos resultados dos experimentos valem 3,0 pontos. Não esqueça de informar nos slides a configuração do computador utilizado para executar os experimentos. Para garantir que as medições não sejam afetadas por outros programas do seu computador, evite rodar outros processos que consumam muita CPU e memória enquanto os experimentos estiverem sendo executados. O ideal mesmo é que você mantenha apenas os terminais com o código do EP rodando.

O desempacotamento do arquivo .tar.gz deve produzir um diretório contendo os itens. O nome do diretório deve ser ep2-seu_nome. Por exemplo: ep2-enedina_marques. Entregas que sejam tarbombs ou que, quando descompactadas, gerem o diretório com o nome errado perderão 2,0 pontos.

A entrega do .tar.gz deve ser feita através do e-Disciplinas.

O EP deve ser feito individualmente.

Obs.: não inclua no .tar.gz itens que não foram pedidos neste enunciado, como por exemplo, dotdirs como o .git, dotfiles como o .gitignore, saídas para diversas execuções, arquivos précompilados, etc.... A presença de conteúdos não solicitados no .tar.gz levarão a um desconto de 2,0 na nota final.