

EP2 de MAC0422

Gabriel Capella e Luís Felipe de Melo

Universidade de São Paulo

October 16, 2016

Problema

Temos que simular uma das provas do ciclismo, a perseguição por equipe. No programa, temos 2 equipes, e cada uma possui n ciclistas. A pista possui d metros. Cada time começa em um lado oposto da pista. Durante a corrida, apenas dois ciclistas podem estar lado a lado no circuito. Vence a equipe cujo terceiro ciclista ultrapassa o terceiro ciclista da outra equipe ou quando o terceiro ciclista de uma equipe termina 16 voltas primeiro.

Implementação

- Cada ciclista é implementado com uma thread `ciclista`.
- Implementamos a pista como dois vetores de tamanho $2*d$ (logo, cada ciclista ocupa duas posições no vetor). Os dois vetores representam a pista interna e a pista externa.
- Fizemos essa escolha porque as velocidades podem variar, no modo `v`.
- Então, quando o ciclista está a 30 km/h, ele se move uma posição no vetor. Quando está a 60 km/h, move-se duas.
- Temos um "juiz" que fiscaliza a corrida.

Considerações

- O custo da ultrapassagem (sair da pista interna e ir para a externa) é o mesmo custo de ir para a frente.
- Cada ciclista é autônomo, ele sabe as regras e decide se ultrapassa ou não.
- A corrida começa com todos os ciclistas fora da pista e os ciclistas da equipe A são inseridos nas posições 0 e 1 (já que em nossa implementação os ciclistas ocupam duas posições de vetor) do vetor e os ciclistas da equipe B são inseridos nas posições d e $d+1$.

- Sobre o ciclista que “segura” os outros: quando um ciclista passar na linha de chegada, ele vai ver se existe algum colega de equipe com número de voltas maior do que o dele em sua frente e que tenha a velocidade de 30km/h. Se ninguém tiver ele sorteia a velocidade, se alguém tiver ele vai andar a 30 km/h.
- Quando dois ciclistas paralelos a 60 km/h se deparam com um ciclista a 30 km/h à frente, o primeiro que conseguir o mutex para ficar paralelo ao de 30 km/h vai manter a velocidade. O outro vai ficar a 30 km/h.

Probabilidade de quebra

- Temos no enunciado que a probabilidade de quebra é 10% e 4 rodadas. Vamos considerar que ela seja uniforme, ou seja, a probabilidade de quebrar em movimento será:

$$P(q) = \frac{0,1 \cdot v}{n \cdot d \cdot 8}$$

- Onde v é a velocidade, que é 1 caso 30 km/h ou 2, caso contrário.
- Exemplo: Um ciclista está andando a 60 km/h. A probabilidade de que ele quebre a cada passo da iteração em uma pista de 400 m é de 0.00125.
- Obs: O ciclista anda e quebra, não começa uma volta quebrado, mas pode terminar assim.

- Em nosso código, utilizamos o tempo como raiz da função `random`.
- Temos a seguinte fórmula:

$$P(q) = \frac{speed}{n \cdot d \cdot 80}$$

- Note que duas vezes a distância máxima é igual à máxima posição que um ciclista pode estar, portanto o 40 da nossa fórmula.

Módulos

barrier.c

- Fizemos a nossa própria biblioteca de barreiras.
- A principal razão disso é que as barreiras da `pthread.h` não funcionariam direito quando um ciclista quebra. Quando ele quebra, sua thread é destruída, e o número de threads é alterado.
- Implementamos nossa barreira como uma struct que possui dois semáforos (mutexes), uma variável condicional das pthreads o número total de threads e um contador de quantas já finalizaram.
- Existem as funções de inicializar, destruir, esperar e a remover uma thread.

Módulos

cyclist.c

- Esse módulo cuida de tudo relacionado aos ciclistas.
- Declaramos o ciclista como uma `struct` que possui os seguintes campos:
 - Número do ciclista.
 - Número do time.
 - A distância total percorrida.
 - Status do ciclista.
 - Velocidade.
 - Qual pista ele está.
 - Momento em que acaba a prova.
 - Qual a linha de chegada dele.
 - Qual volta ele está.
 - Se o inspetor validou a volta dele.

- Esse módulo possui funções públicas e privadas.
- As funções públicas são as de configurar o ambiente, inicializar o ciclista, a função para a thread do ciclista e a função de desalocar a memória.
- As funções privadas são as funções de ver como está a frente do ciclista (posição e semáforo mutex), a de colocar o ciclista na pista e a de avançar \times posições.

Módulos

inspector.c

- Esse módulo funciona como um juiz para a prova.
- É como se tivesse alguém responsável por ver se a corrida está indo de forma correta.
- Suas funções são: mostrar quem quebrou, exibir os dados do fim da corrida e dizer se o ciclista pode acelerar (de 30 km/h para 60 km/h).
- Ele possui uma função de inicialização (basicamente pegando as informações que vieram da `main`) e uma que checa as seguintes informações:
 - O número de voltas.
 - O decremento da barreira.
 - Como estão os terceiros das equipes (se terminou a corrida ou se passou na frente do outro terceiro).

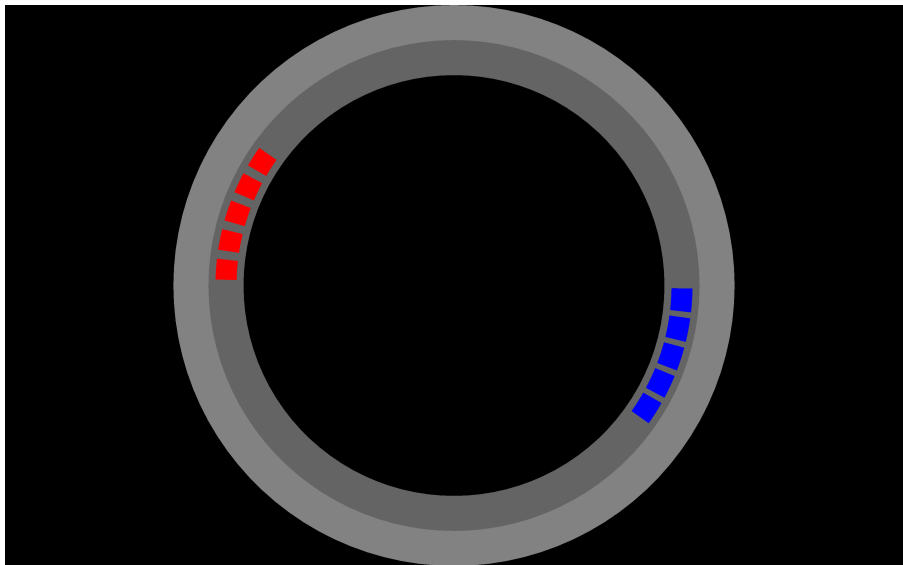
Módulos

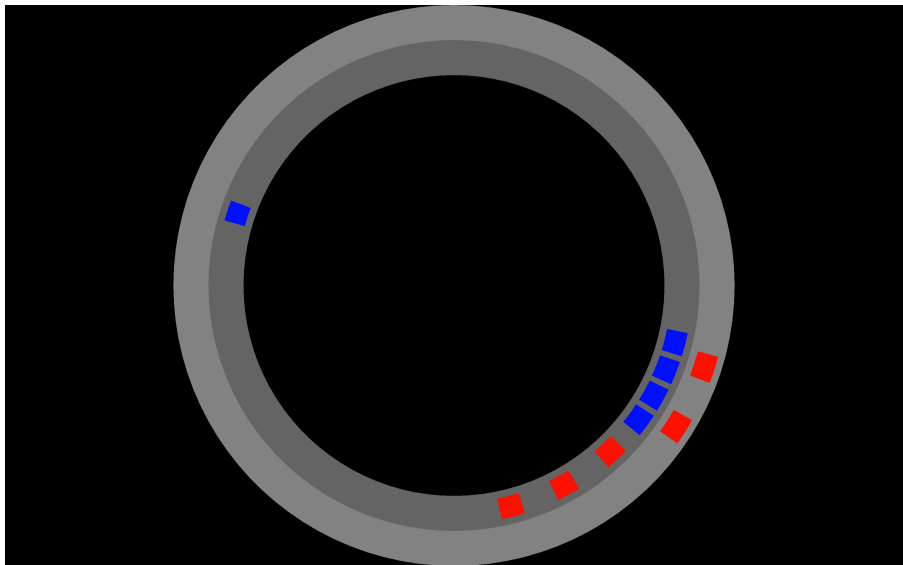
ep2.c

- Recebe as informações da entrada e assim define as modos de execução, passando essas informações para as funções dos módulos.
- Declara o espaço de trabalho, chama as funções de inicialização e então convoca as de execução para os ciclistas.
- Usa as funções do juiz para exibição de informações.

Simulador

- Para ver se nossa implementação foi feita corretamente, fizemos um simulador para averiguar se a corrida estava coerente.
- Os resultados estão a seguir:





Entrada

```
$ ./ep2 d n [v|u] [-v]
```

- d: número inteiro que representa o comprimento em metros da pista.
- n: número de ciclistas em cada equipe.
- v: usado para definir simulações com velocidades aleatórias a cada volta.
- u: define simulações com velocidades uniformes de 60 km/h.
- -v: opção para mostrar tudo (debug), utilizado para produzir saída para o terminal.

Saída

- A saída tem a seguinte forma:

```
Mode: RANDOM
30.24 s > TEAM: 0      F: 0      (252) S: 1      (251) T: 3      (250) LAP 1
30.24 s > TEAM: 1      F: 7      (254) S: 4      (251) T: 5      (250) LAP 1
60.24 s > TEAM: 0      F: 0      (504) S: 1      (501) T: 3      (500) LAP 2
60.24 s > TEAM: 1      F: 7      (752) S: 4      (501) T: 5      (500) LAP 2
90.24 s > TEAM: 0      F: 0      (1002) S: 1      (751) T: 3      (750) LAP 3
90.24 s > TEAM: 1      F: 7      (1002) S: 4      (751) T: 5      (750) LAP 3
```

- Mostrando o modo das velocidades, e a cada volta (tempo), o primeiro, o segundo e o terceiro colocado de cada time, com a sua metragem percorrida (por 16 voltas).

```
Broken: Team 0: 2 (13),
Broken: Team 1: 7 (7),
Finishing order: 4 (405.63), 0 (420.51), 5 (428.07), 1 (450.39), 6 (465.51), 3 (480.27),
Result: TEAM 1 WIN
```

- No final, mostra os ciclistas quebrados (número do ciclista e volta que quebrou), a ordem de chegada (número do ciclista e tempo) e quem venceu ou se houve empate.

- Se o resultado impresso na imagem fosse `TEAM 1 WIN*`, isso significa que o terceiro dessa equipe passou o terceiro da outra equipe durante a corrida.
- No modo debug, temos as seguintes informações (respectivamente): o número do ciclista, seu time, a distância percorrida na volta, o status, qual pista ele está (interna ou externa), a distância total percorrida e a volta.






```

C
N 0, T 0, P 250, S 2, E 1, D 8000, V 16
N 1, T 0, P 250, S 2, E 0, D 8000, V 16
N 2, T 0, P 250, S 2, E 0, D 8000, V 16
N 3, T 0, P 250, S 2, E 1, D 8000, V 16
N 4, T 1, P 0, S 2, E 1, D 8000, V 16
N 5, T 1, P 0, S 2, E 0, D 8000, V 16
N 6, T 1, P 254, S 3, E 0, D 6754, V 13
N 7, T 1, P 0, S 2, E 0, D 8000, V 16
C

```

Testes

- Para os testes, utilizamos as seguintes máquinas:

<input type="checkbox"/> Nome ^	Zona	Tipo de máquina
<input type="checkbox"/>  i1	us-central1-b	8 vCPUs, 7,2 GB
<input type="checkbox"/>  i2	asia-east1-b	8 vCPUs, 7,2 GB
<input type="checkbox"/>  i3	europa-west1-b	8 vCPUs, 7,2 GB
<input type="checkbox"/>  i4	us-east1-b	8 vCPUs, 7,2 GB
<input type="checkbox"/>  i5	us-west1-b	8 vCPUs, 7,2 GB

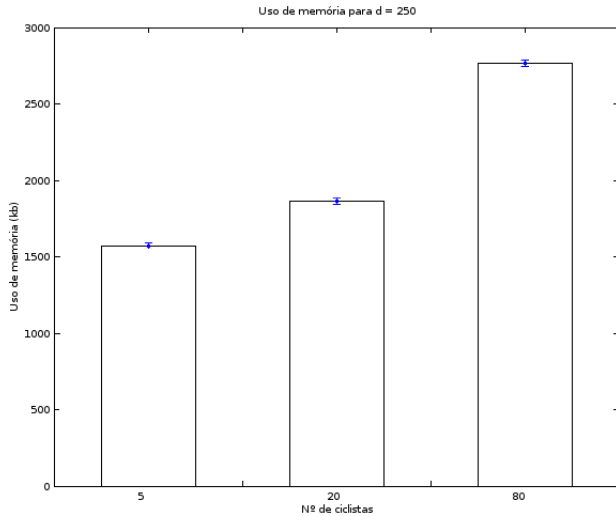
- O uso de CPU ficou assim:

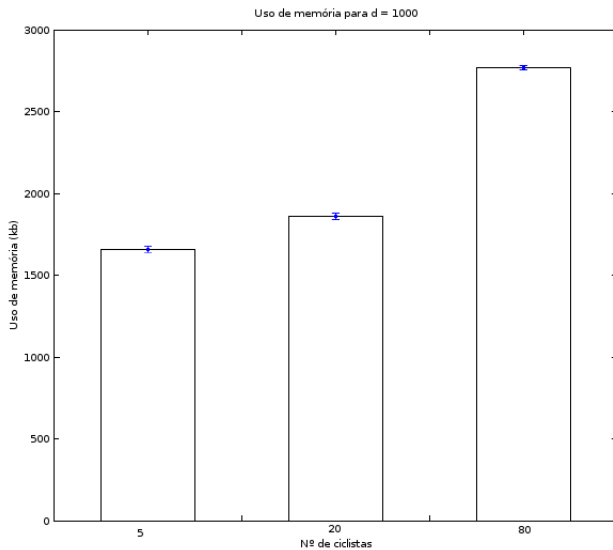


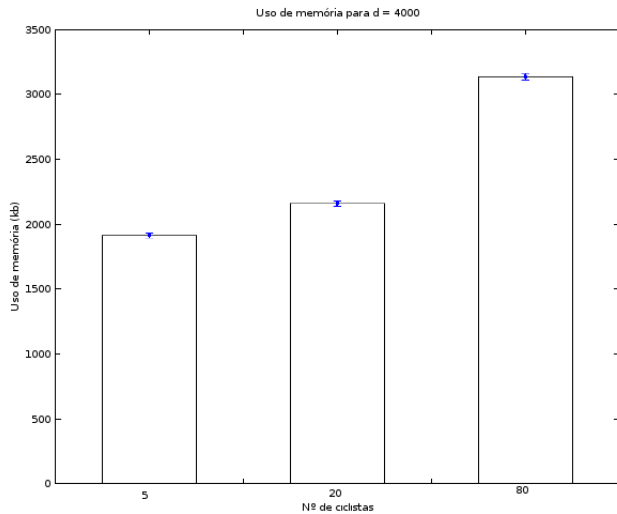
- Executamos o programa da seguinte maneira:
- `$ /usr/bin/time -v ./ep2 d n v`
- Com isso, temos uma saída desse tipo, além da saída do programa:

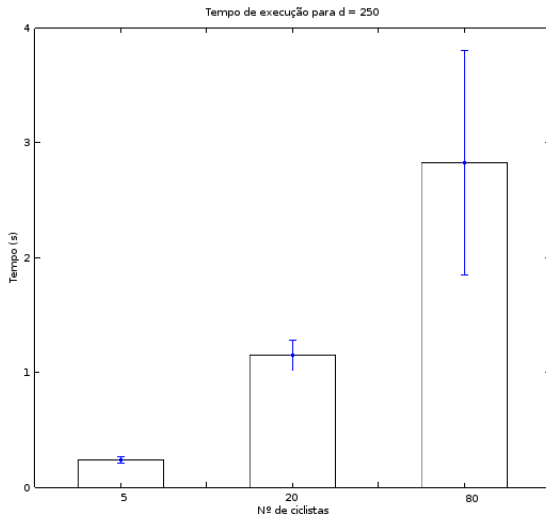
```
Command being timed: "./ep2 250 4 v"
User time (seconds): 0.30
System time (seconds): 0.68
Percent of CPU this job got: 78%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:01.26
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 1780
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 101
Voluntary context switches: 135416
Involuntary context switches: 527
Swaps: 0
File system inputs: 8
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

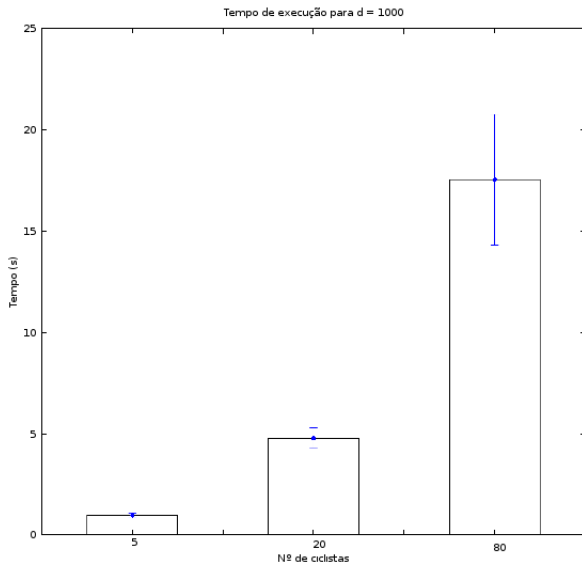
- De todos esses dados, utilizamos o Maximum resident set size para fazer as medições de uso de memória.
- Esse item representa o máximo de memória que pertenceu ao processo e ficou na RAM durante a execução, em kbytes.
- Para o tempo de execução do processo, utilizamos o User time, que é o tempo que o processo dura na visão do usuário, em segundos.
- Com isso, temos os seguintes gráficos, com poucos (5), muitos (80) e um valor intermediário de ciclistas (20) para uma pista pequena (250 m), média (1000 m) e grande (4000 m).

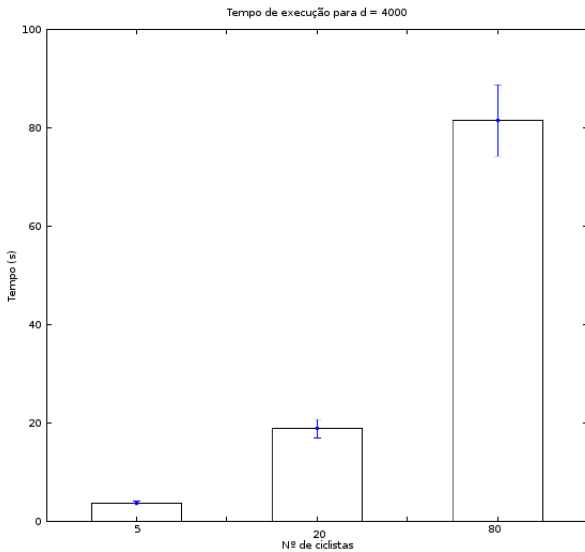












Feito por:

- Gabriel Capella, 8962078.
- Luís Felipe de Melo, 9297961.