MAC0422 – 2016/2
Sistemas Operacionais

Exercício-programa 2

Profº Daniel Macêdo Batista

Vítor Kei Taira Tamada -8516250

Estruturas

Pista: Vetor de structs

main(): Coordenadora

ciclista(): Thread

- Variáveis globais
- Barreiras
- Mutex

Pista

Struct com mutex próprio

Controle de mudança de espaço na pista

Vetor de 2 elementos: lane[2]

Pode haver apenas dois ciclistas um do lado do outro em qualquer dado momento

main()

Inicializa as barreiras, mutex e variáveis globais

Coordenadora das threads – uma thread além dos 2n ciclistas

Modo de debug

Impressão de resultados/relatório ao final da simulação

ciclista()

Variáveis globais: c_r, c_p, c_b, c_eor, t_l, spd30

c_r[2]: cyclists_running

c_r[*t*] recebe o número de ciclistas ainda correndo na equipe *t*

c_p[2][16][*n*]: cyclist_placement

c_p[*t*][*l*][*c*] recebe o ID do ciclista da equipe *t*, volta *l* e colocação *c* em relação à equipe

c_b[3]: cyclists_broken

c_b[*l*] recebe o ID do ciclista que quebrou na volta 4(*l*+1)

c_eor[2][2][*n*]: cyclist_end_of_race:

c_eor[0][*t*][*c*] recebe o ID do ciclista da equipe *t* que ficou em *c*-ésimo lugar em relação a sua equipe

c_eor[1][*t*][*c*] recebe o tempo que o ciclista levou para terminar a corrida

t_l[2]: team_lap

t_l[*t*] determina se a equipe *t* inteira terminou uma volta

spd30[2]: speed_is_30

spd30[*t*] é uma variável Booleana que determina se alguém da equipe *t* está a 30km/h

ciclista()

<u>Três barreiras</u>: b0, b1, b_aux

Threads chegam em **b0**. Se uma thread for destruída (ciclista quebrar ou terminar a corrida), todas, incluindo a coordenadora e a que foi destruída, param em **b_aux**

Enquanto threads estiverem em **b_aux**, coordenadora constrói **b1** para receber uma thread a menos que **b0**

Thread do ciclista que quebrou só é destruída ao passar por **b_aux**

Repetir até corrida acabar

b_n_t: barrier_number_of_threads

Variável global que recebe o número de threads que uma barreira que está para ser construída deve esperar

ciclista()

Quatro mutex: c_b_m, e_c_m, c_p_m, s_c_m

c_b_m: cyclist_break_mutex

Relacionado ao momento da quebra de um ciclista – apenas um ciclista deve quebrar a cada quatro voltas cumpridas

e_c_m: end_of_cyclist_mutex

Relacionado à finalização da corrida por um ciclista, seja por quebra ou por correr as 16 voltas – organiza os ciclistas por ordem de término da corrida

c_p_m: cyclist_placement_mutex

Relacionado à ordenação dos ciclistas de uma mesma equipe por colocação – ordena os ciclistas de uma mesma equipe por colocação

s_c_m: speed_change_mutex

Relacionado à mudança de velocidade de um ciclista – se um ciclista muda para velocidade de 30km/h, todos os que estiverem atrás deverão ter essa mesma velocidade

ciclista()

<u>Duas fases por pulso de clock</u>: quebra de ciclista e movimento de ciclista

1) Quebra de ciclista

Verifica se está em uma volta em que um ciclista quebra e age de acordo (quebrando o ciclista ou não)

2) Movimento de ciclista:

Realiza as verificações e ações que resultam no ciclista mover-se ou não (sem espaço a frente para se mover, turno em que o ciclista não se move caso esteja a 30km/h, ...)

ciclista()

<u>Saída do ciclo de pulsos de clock</u>: término da corrida ou quebra do ciclista

1) Remoção do ciclista da pista

2) Armazenamento de sua colocação bem como tempo que terminou a corrida (caso não tenha saído por quebra)

3) Ajuste das barreiras

Resultados dos testes

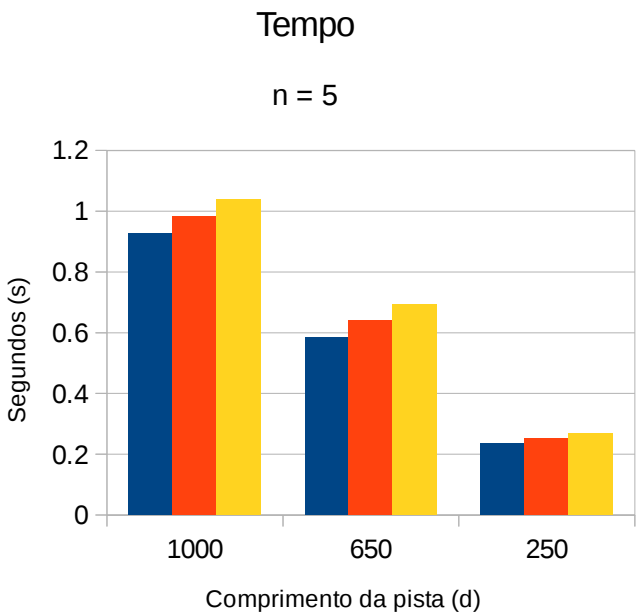
Especificações do computador utilizado para realizar os testes:

- Intel Core i7-3630QM CPU @ 2.40GHz * 8
- Ubuntu 14.04 LTS 64-bit
- 7.7 GB de RAM

Dados obtidos por meio do comando

- /usr/bin/time -v ./ep2 *d n u*
  - *d* e *n* são os argumentos que variam entre os blocos de testes
  - Testes sempre realizados com modo de velocidade uniforme (comando *u*)
- **Tempo:** Elapsed (wall clock) time (h:mm:ss or m:ss)
- **Espaço:** Maximum resident set size (kbytes)

(d, n) = ({250, 650, 1000}, 5)
Efeito do aumento de d



d	Média	I.C.
1000	0.9827	[0.9254; 1.04]
650	0.639	[0.58567; 0.69233]
250	0.252	[0.236487; 0.26751]

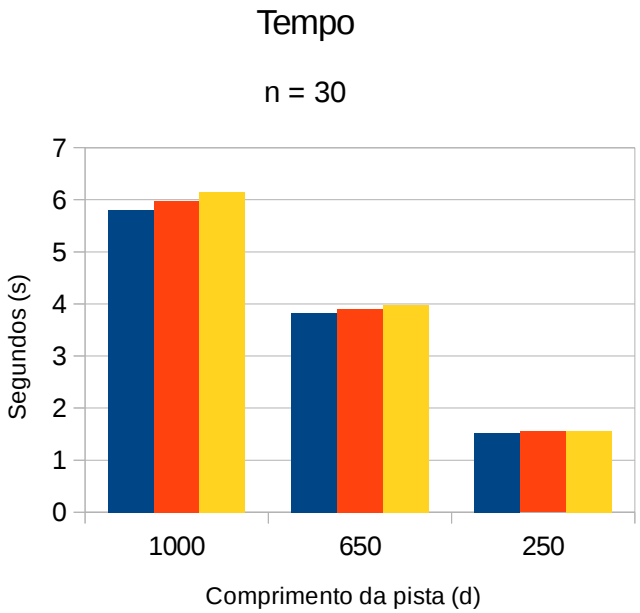
Espaço

Na maior parte dos testes para um determinado d e n, o consumo de memória era igual ao mínimo.
Entretanto, havia alguns poucos casos em que o consumo de memória era de valores muito altos, fazendo com que a variância atingisse valores absurdos. Como a média é mais próximo do valor mínimo do que do máximo, conclui-se que esses valores são outliers e, portanto, o consumo de memória deveria ser constantemente do valor mínimo.

Gráficos não construídos devido os outliers que os estragariam.

d	Média	Máximo	Mínimo
1000	923.73	4744	792
650	839.87	2692	776
250	888	4596	760

(d, n) = ({250, 650, 1000}, 30)
Efeito do aumento de d



d	Média	I.C.
1000	5.977	[5.808; 6.14523]
650	3.906	[3.831; 3.981]
250	1.544	[1.525; 1.563]

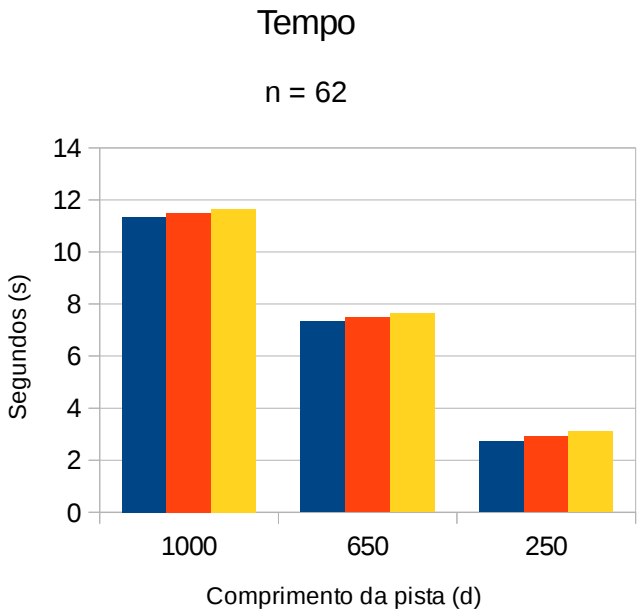
Espaço

Na maior parte dos testes para um determinado d e n, o consumo de memória era igual ao mínimo.
Entretanto, havia alguns poucos casos em que o consumo de memória era de valores muito altos, fazendo com que a variância atingisse valores absurdos. Como a média é mais próximo do valor mínimo do que do máximo, conclui-se que esses valores são outliers e, portanto, o consumo de memória deveria ser constantemente do valor mínimo.

Gráficos não construídos devido os outliers que os estragariam.

d	Média	Máximo	Mínimo
1000	1345.5	5100	1216
650	2169.6	5108	1200
250	1834.5	5112	1180

(d, n) = ({250, 650, 1000}, 62)
Efeito do aumento de d



d	Média	I.C.
1000	11.506	[11.349; 11.6632]
650	7.482	[7.33; 7.64]
250	2.9163	[2.731; 3.102]

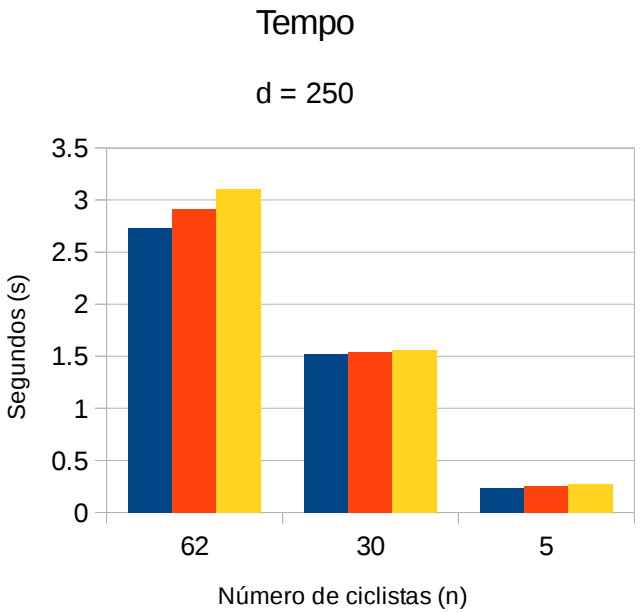
Espaço

Na maior parte dos testes para um determinado d e n, o consumo de memória era igual ao mínimo.
Entretanto, havia alguns poucos casos em que o consumo de memória era de valores muito altos, fazendo com que a variância atingisse valores absurdos. Como a média é mais próximo do valor mínimo do que do máximo, conclui-se que esses valores são outliers e, portanto, o consumo de memória deveria ser constantemente do valor mínimo.

Gráficos não construídos devido os outliers que os estragariam.

d	Média	Máximo	Mínimo
1000	2144.1	5644	1756
650	2648.5	5636	1740
250	2107.7	5636	1180

(d, n) = (250, {5, 30, 62})
Efeito do aumento de n



n	Média	I.C.
62	2.9163	[2.731; 3.102]
30	1.544	[1.525; 1.563]
5	0.252	[0.2365; 0.2675]

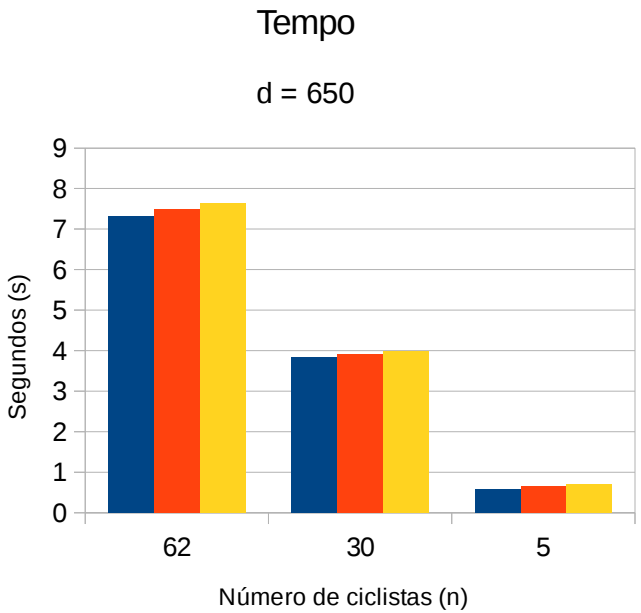
Espaço

Na maior parte dos testes para um determinado *d* e *n*, o consumo de memória era igual ao mínimo.
Entretanto, havia alguns poucos casos em que o consumo de memória era de valores muito altos, fazendo com que a variância atingisse valores absurdos. Como a média é mais próximo do valor mínimo do que do máximo, conclui-se que esses valores são outliers e, portanto, o consumo de memória deveria ser constantemente do valor mínimo.

Gráficos não construídos devido os outliers que os estragariam.

n	Média	Máximo	Mínimo
62	2107.7	5636	1720
30	1834.5	5112	1180
5	888.0	4596	760

(d, n) = (650, {5, 30, 62})
Efeito do aumento de n



n	Média	I.C.
62	7.482	[7.327; 7.637]
30	3.906	[3.831; 3.981]
5	0.639	[0.5857; 0.692]

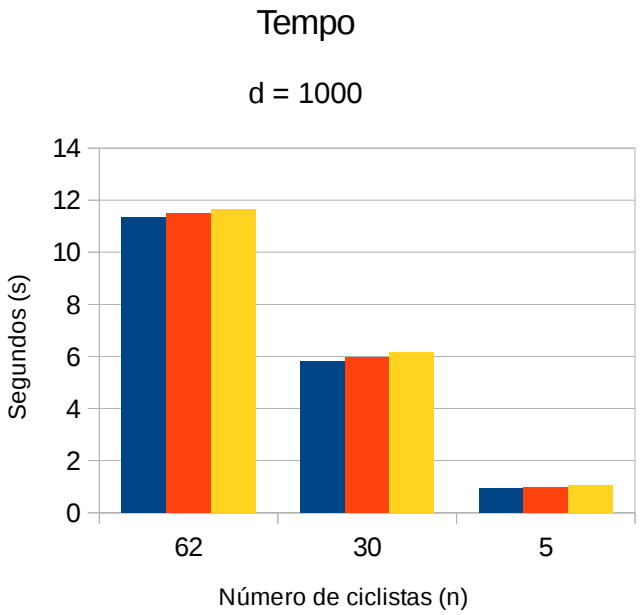
Espaço

Na maior parte dos testes para um determinado d e n, o consumo de memória era igual ao mínimo.
Entretanto, havia alguns poucos casos em que o consumo de memória era de valores muito altos, fazendo com que a variância atingisse valores absurdos. Como a média é mais próximo do valor mínimo do que do máximo, conclui-se que esses valores são outliers e, portanto, o consumo de memória deveria ser constantemente do valor mínimo.

Gráficos não construídos devido os outliers que os estragariam.

n	Média	Máximo	Mínimo
62	2648.5	5636	1740
30	2169.6	5108	1200
5	839.9	2692	776

(d, n) = (1000, {5, 30, 62})
Efeito do aumento de n



n	Média	I.C.
62	11.506	[11.349; 11.663]
30	5.977	[5.809; 6.145]
5	0.9827	[0.925; 1.04]

Espaço

Na maior parte dos testes para um determinado d e n, o consumo de memória era igual ao mínimo.
Entretanto, havia alguns poucos casos em que o consumo de memória era de valores muito altos, fazendo com que a variância atingisse valores absurdos. Como a média é mais próximo do valor mínimo do que do máximo, conclui-se que esses valores são outliers e, portanto, o consumo de memória deveria ser constantemente do valor mínimo.

Gráficos não construídos devido os outliers que os estragariam.

n	Média	Máximo	Mínimo
62	2144.1	5644	1756
30	1345.5	5100	1216
5	923.7	4744	792

FIM