EP2 - MAC0422 - Sistemas Operacionais

Nome: Erick Rodrigues de Santana NUSP: 11222008
Nome: Vinicius Pereira Ximenes Frota NUSP: 11221967

Arquitetura do programa

Controle de acesso à pista

No programa, a pista é representada por uma matriz de 10 linhas e d colunas. A orientação da corrida é da esquerda para a direita, ou seja, as posições dos ciclistas são atualizadas de 0 até d-1 na matriz.

O controle de acesso é feito através de uma matriz de mutex de mesmo tamanho da pista, ou seja, um mutex para cada posição (i, j) da matriz.

Estruturas de dados utilizadas

Após obter os parâmetros d e n, o EP cria um vetor de tamanho n de um struct chamado cyclist (ver cyclist.h), que contém as seguintes informações:

- Linha e coluna que o ciclista se encontra na pista (linePosition and columnPosition)
- Velocidade com que o ciclista está andando (1 para 30km/h, 2 para 60km/h e 3 90km/h) (velocity)
- Quantidade de voltas do ciclista completou (laps)
- Quanto tempo (de relógio) ele pedalou (runningTime)
- Três flags: se ele quebrou (broken), se ele foi eliminado (eliminated), ou se ele terminou todas as voltas que deveria pedalar (finished)

Estruturas de dados utilizadas

Para manter a classificação volta a volta dos ciclistas, foi utilizado um vetor de pilhas (implementada em stack.c e stack.h) de tamanho 2 * n (quantidade máxima de voltas). Assim, quando um ciclista completa uma volta, ele se coloca na pilha dessa volta. Logo, quando completamos uma volta de eliminação, olhamos para o primeiro da pilha que ainda está rodando (ou seja, não quebrou e não foi eliminado) e está na linha de chegada e eliminamos ele.

Fluxo do programa

A cada intervalo de tempo (60ms ou 20ms), as threads dos ciclistas executam operações e atualizam suas posições na matriz. Após isso, elas ficam presas em uma barreira e uma função judge() é executada.

Essa função judge() tem, como o próprio nome diz, o papel de juiz da corrida. É ela quem vai dar o usleep do intervalo de tempo, para depois permitir os ciclistas pedalarem (se o intervalo de tempo percorrido é suficiente para o ciclista mover 1m). Ela também tem como responsabilidade verificar quebras, eliminações e etc. Ela é executada na main thread do programa.

Após restar somente um ciclista na pista (logo quando o segundo é eliminado), finalizamos a corrida. Nesse ponto, mostramos a classificação final, que é calculada no arquivo sort.c, que ordena o vetor de ciclistas com base na quantidade de voltas percorridas e, caso houver empate, pelo tempo pedalado.

Ultrapassagem

Para realizar a ultrapassagem, foi adotada a seguinte estratégia: se os ciclistas conseguirem avançar para frente, eles avançam. Caso contrário, eles verificam se existem duas posições consecutivas livres nas faixas mais externas as que ele está. Se isso for possível, então ele avança. Senão, ele “pedala” na mesma velocidade que o da frente. No nosso programa, consideramos que pedalar na mesma velocidade do ciclista da frente seria o mesmo que “esperar” o ciclista da frente avançar.

Barreiras

Utilizamos as barreiras encontradas em pthread.h (pthread_barrier). Também utilizamos um vetor chamado canContinue, de n posições, que indica se, após a execução da função judge(), cada ciclista pode ou não continuar a corrida.

Mais uso de mutex

Além da matriz de Mutex para o controle de acesso à pista, criamos também um mutex para cada posição do vetor de pilhas (assim, dois ciclistas não dão um push simultaneamente e as informações são mantidas). Há um mutex que protege a função rand(), assim a aleatoriedade é mantida.

Algumas decisões de projeto

Após um ciclista completar todas as voltas que deveria pedalar, deixamos sua thread em execução, mas não alteramos sua posição na pista.

No programa, consideramos que a primeira volta é a 0 e calculamos a probabilidade de um ciclista quebrar assim que ele chega em uma volta múltipla de 6. Por exemplo, quando o ciclista chega na volta 6 e ele quebra, ele aparece no ranking da volta 5 e no ranking da 6, mas ele deixa de aparecer nos próximos rankings.

Resultados

Resultados

Foram feitos testes com as seguintes entradas:

Número de ciclistas: 10, 50 e 100 (poucos, médio e muitos, respectivamente)

Tamanho da pista: 250, 350 e 450 (pequena, média e grande, respectivamente).

Escolhemos esses valores pois não tínhamos tempo suficiente para realizar testes com valores maiores e julgamos esses valores como suficientes para realizar os testes necessários.

Foram 30 execuções para cada teste e foram gerados 4 gráficos analisando o tempo de execução para cada par (n, d) de entradas, além do uso de memória do programa.

Para o cálculo do intervalo de confiança com nível de confiança de 95%, supomos que os dados coletados formam uma distribuição normal.

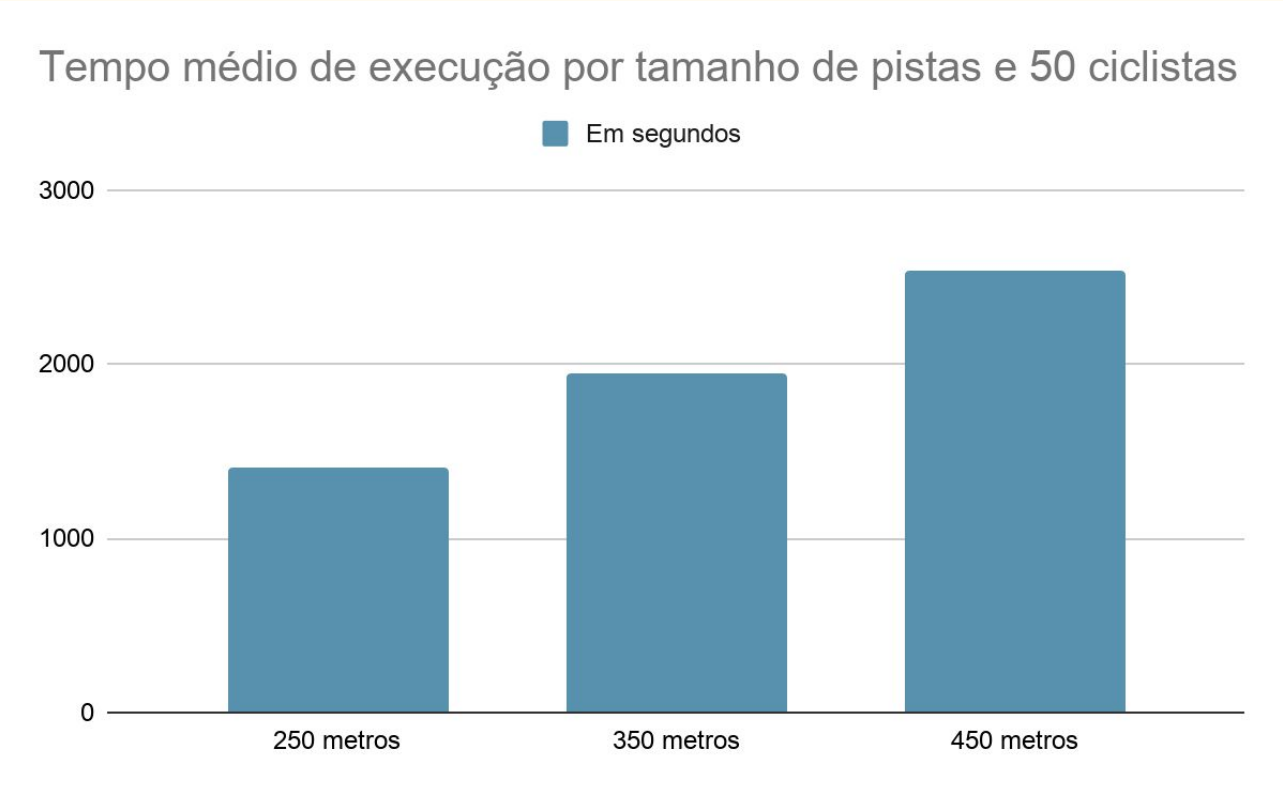
Resultados

Os dados obtidos foram coletados pela função generateOutput(). Ela cria um arquivo contendo o tempo de relógio de execução do programa e calcula o uso de memória a partir do /proc/self/status. Desse arquivo, pegamos o dado correspondente a RSSAnon (resident set size), pois, ao monitorar o uso de memória do EP por um software de interface gráfica, julgamos que esse valor é o mais próximo do real.

Para os testes com 10 ciclistas, rodamos 30 processos paralelamente. Para os testes com 50 ciclistas, primeiro rodamos 10 processos paralelamente, depois 20 processos paralelamente. Por fim, para os testes com 100 ciclistas, rodamos 10 processos paralelamente 3 vezes.

Todos os testes de tempo e de memória foram feitas na rede Linux.

Número médio de tempo de execução para pistas



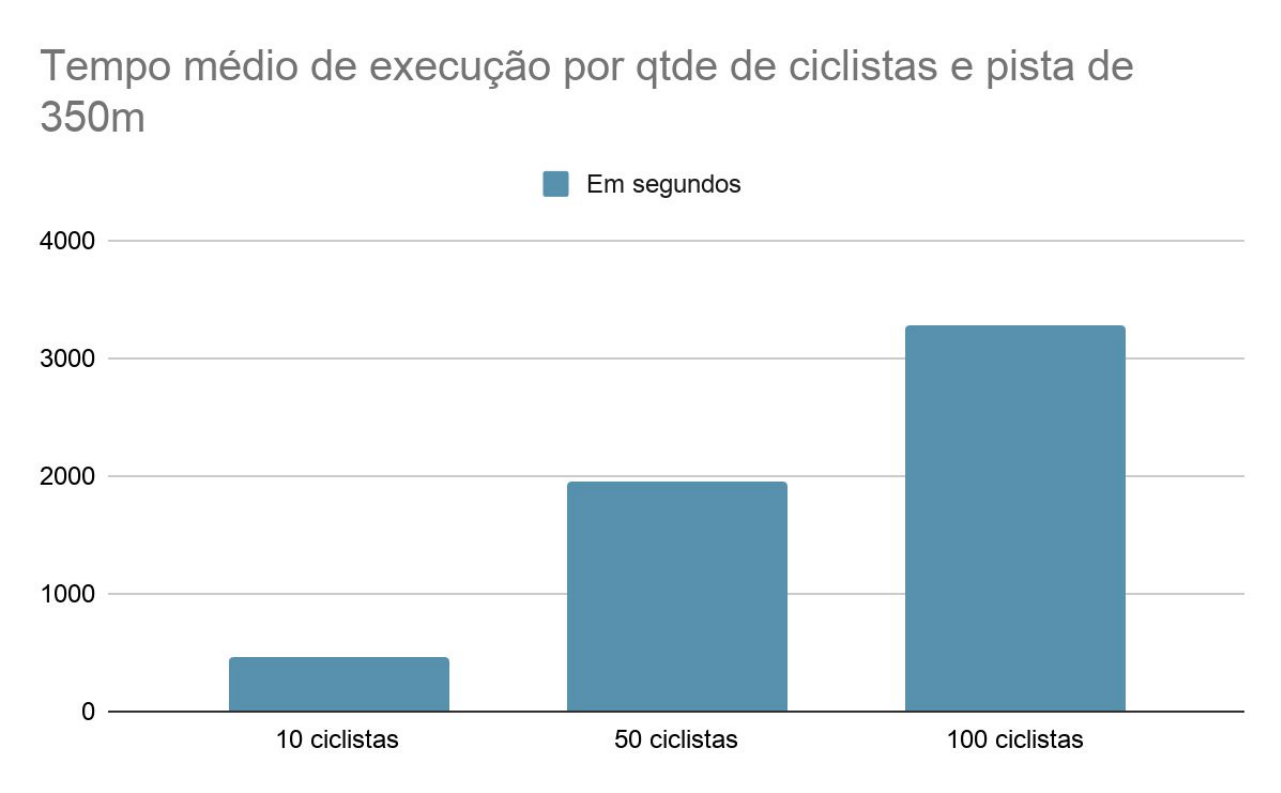
Intervalo de confiança para o tempo médio de execuções por tamanho de pista e 50 ciclistas

IC da média amostral para pista de 250m: [1370.867073, 1449.001780]

IC da média amostral para pista de 350m: [1897.376131, 2002.608585]

IC da média amostral para pista de 450m: [2457.082575, 2616.038612]

Número médio de tempo de execução por quantidade ciclistas



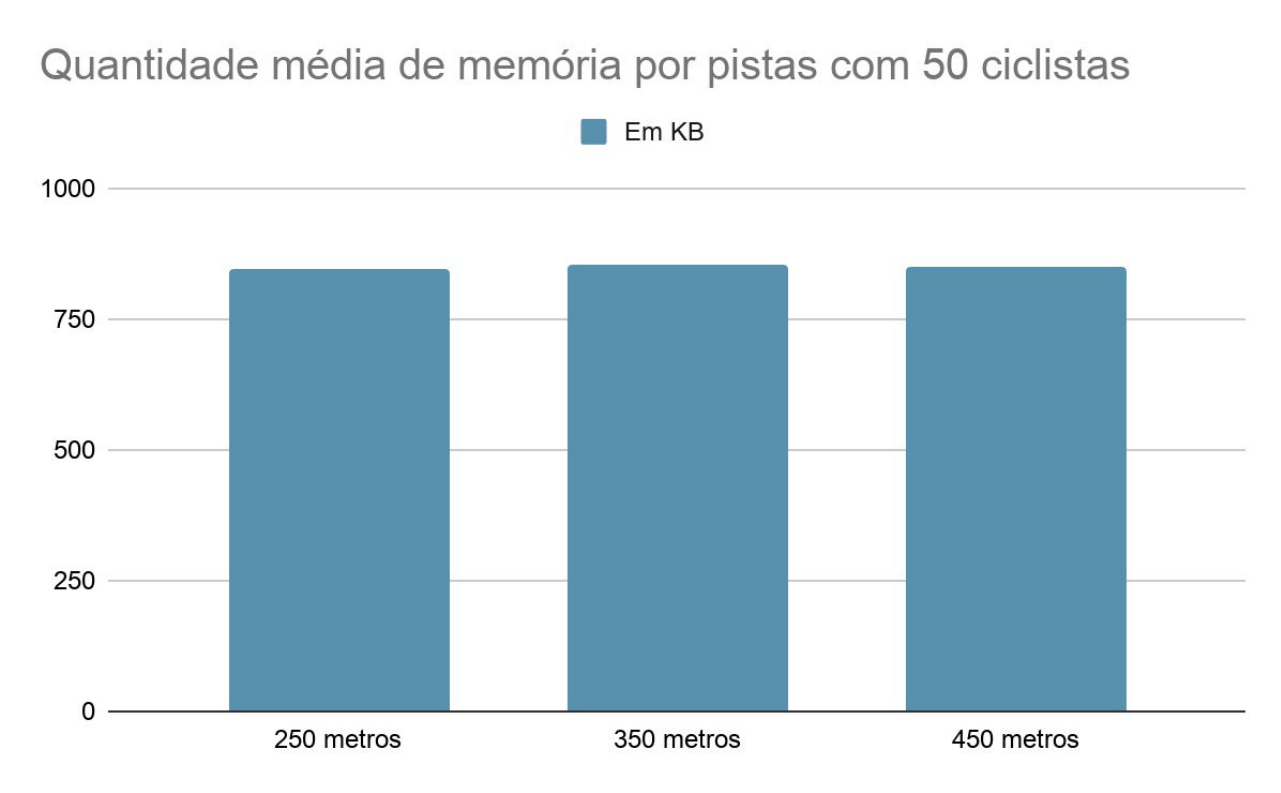
Intervalo de confiança para o tempo médio de execuções por quantidade de ciclistas e pista de 350m

IC da média amostral para 10 ciclistas: [441.307287, 469.870777]

IC da média amostral para 50 ciclistas: [1897.376131, 2002.608585]

IC da média amostral para 100 ciclistas: [3176.529642, 3365.040842]

Quantidade média de uso de memória por tamanho de pistas



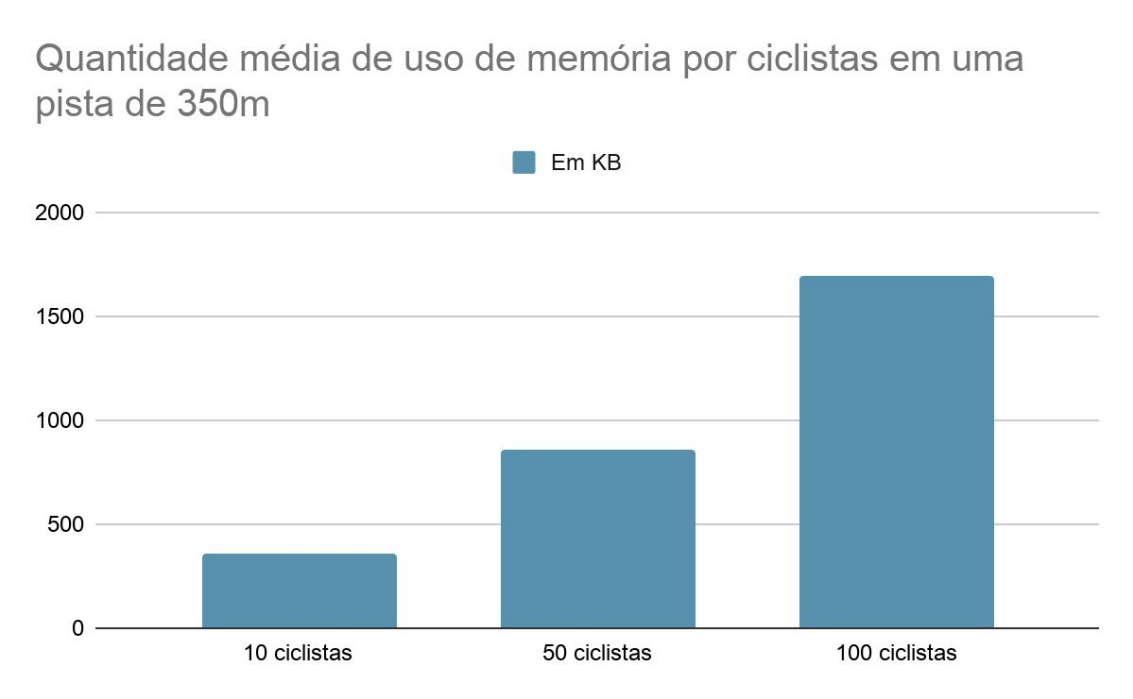
Intervalo de confiança para a quantidade média de memória por tamanho de pista e 50 ciclistas

IC da média amostral para 250m: [841.046680, 852.553320]

IC da média amostral para 350m: [848.145243, 860.121423]

IC da média amostral para 450m: [846.078683, 853.654651]

Quantidade média de uso de memória por número de ciclistas



Intervalo de confiança para a quantidade média de uso de memória por ciclistas e pista de 350m

IC da média amostral para 10 ciclistas: [354.942804, 360.257196]

IC da média amostral para 50 ciclistas: [848.145243, 860.121423]

IC da média amostral para 100 ciclistas: [1688.224654, 1706.442012]

Considerações Finais

Considerações finais

Os resultados dos experimentos foram, em partes, esperados. Percebemos que, conforme aumenta-se o número de ciclistas, aumenta-se também o uso de memória. Isso ocorre porque, no nosso programa, a maioria das alocações que fazemos depende do número de ciclistas.

No caso de diferentes tamanhos de pistas, a diferença de uso de memória foi mínima, já que, no nosso programa, somente a matriz que representa a pista e a matriz de semáforos dependem do tamanho da pista. Assim, não houve tanta diferença para diferentes tamanhos de pista com a mesma quantidade de ciclistas.

Considerações finais

Com relação ao tempo de execução do programa, os experimentos mostraram que o tamanho da pista e a quantidade de ciclistas influenciam diretamente no tempo. Nos testes para diferentes quantidades de ciclistas, houve uma discrepância maior do que nos testes para diferentes tamanhos de pistas. Isso ocorreu pois consideramos, nos nossos experimentos, ordens de grandezas diferentes para pistas e ciclistas.

Como os testes com muitos ciclistas são 10 vezes maiores do que com pequenos ciclistas, é esperado que o programa rode 10 vezes mais lento, só que os valores obtidos foram menor que isso. Isso se deve aos ciclistas quebrados e os outros eventos aleatórios da corrida (velocidade, decisão do escalonador, etc).