MAC0422 - Sistemas Operacionais - EP1

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

Arquitetura do Shell

Introdução

O shell BCCSH consegue suportar os seguintes comandos:

1. mkdir <arquivo>
2. kill -9 <pid>
3. ln -s <arquivo1> <arquivo2>

Além disso, suporta a execução de três binários abaixo:

1. /usr/bin/du -hs .
2. /usr/bin/traceroute www.google.com.br
3. /ep1 <argumentos do EP1>

Funções e funcionamento do parser de comandos

Para obter o nome do usuário e o diretório atual, foram utilizadas as funções getenv(“USER”) e getpwd(). Além disso, utilizamos a função srtcat() para concatenar os dados e deixar no formato desejado.

Quando o usuário digitar a string que identifica o comando junto com os parâmetros, usamos a função readline() para obter a linha inteira e, então, utilizamos a função strsep(“ “) para separar o comando dos parâmetros através do espaço. Além disso, usamos a função add_history() para adicionar o comando no histórico de comandos.

Execução dos comandos

Para identificar qual comando executar, foi criada uma função chamada execute_command(). Esse função faz uma série de condições que comparam o comando digitado com os comandos suportados pelo shell. Utilizamos strcmp() para realizar tais comparações.

Para os três comandos suportados (mkdir, ls e kill), utilizamos funções em C apropriadas e que foram encontradas na manpage do linux.

Para a execução dos três binários, criamos execute_bin() e que tem como função dar fork() e rodar execvp(), passando os argumentos necessários como parâmetro.

Se o usuário digitou um comando que não é implementado com syscall, o shell executa a função execute_bin()

Implementação dos escalonadores

Implementação dos escalonadores

O EP1 recebe 4 argumentos: o número que identifica o escalonador, o nome do arquivo trace de entrada, o nome do arquivo de saída que será gerado, além de um parâmetro opcional “d” que funciona como log da execução do programa.

Os escalonadores são:

1. First-Come First-Served
2. Shortest Remaining Time Next
3. Round-Robin

Funcionamento do simulador de processos

O programa lê o arquivo trace de entrada e armazena em um vetor de struct chamado Schedule, que contém as seguintes informações sobre o processo: tempo de chegada (t0), tempo de duração (dt), tempo final (tf), deadline e a thread que o processo vai executar.

Esse struct está contido no arquivo schedule.h

Assim que lê os processos do trace e cria o vetor, o programa chama uma função específica para executar o escalonador desejado. As funções criadas são fcfs(), que executa o escalonador 1, srtn(), que executa o escalonador 2 e roundRobin() que executa o escalonador 3.

Funcionamento comum dos escalonadores

Ambas as funções feitas para os escalonadores contém certas semelhanças no seu funcionamento. No início de ambas, iniciamos as threads de cada processo e os semáforos utilizados no código. Para isso, foram feitas duas funções auxiliares, initPthreads() e initMutex() para realizar essas operações, respectivamente.

Ambas as funções também executam um while (true) e analisam a chegada de novos processos utilizando as funções das bibliotecas time.h e sys/time.h, que utilizam o tempo do sistema.

A cada segundo (exceto no Round-Robin, que usamos meio segundo) passados no relógio do sistema, verificamos se houve a chegada de novos processos no escalonador. Se chegou processos novos, incluímos na estrutura de dados utilizada pelo escalonador.

First-Come First-Served

Esse escalonador fica sempre rodando em paralelo com o processo que está executando e verifica a chegada de novos a cada segundo. Utilizamos o próprio vetor de Schedules, que contém as informações lidas, como estrutura de dados.

Utilizamos também um vetor de semáforos mutex (um para cada processo) e, dessa forma, bloqueamos as threads que não estão rodando. Para liberar o processo desejado, fazemos unlock no semáforo adequado na função do escalonador. Quando o escalonador percebe que um processo estava executando e o dt deste processo é igual a 0 ou se não há nada rodando e há processos na fila, nós liberamos o processo que está disponível (o que chegou na fila primeiro).

Shortest Remaining Time Next

Para esse escalonador, foi criada uma interface de min-heap (heap.c e heap.h), no qual inserimos os processos que chegam no escalonador. O heap funciona como uma fila de prioridades, onde o processo com menor dt é o primeiro elemento do heap.

O escalonador tem um while (true) que detecta cada mudança de segundo usando funções do time.h. Toda vez que chegamos em um segundo de diferença, ele insere os elementos que chegaram no sistema no heap. Se não há nada executando, ele remove o primeiro elemento do heap e libera a execução dele. Se o processo A está executando e dt é igual a 0 ou o processo B que está no heap possui um dt menor que o do processo A, A é reinserido nesta estrutura de dados, enquanto B é removido da mesma e entra em execução. Caso não exista B e dt de A seja maior que 0, A continua executando.

Como a cada segundo a thread que executa o processo diminui o valor do dt e precisamos verificar esse valor no escalonador, foi utilizado um vetor de semáforos empty e um vetor de semáforo full. Assim, quando a thread está alterando o valor do dt, o escalonador fica esperando essa operação acontecer para assim tomar as decisões necessárias.

Round-Robin

Para esse escalonador, analogamente ao SRTN, foi criada uma interface de fila (queue.c e queue.h). A diferença é que ela é implementada com uma lista ligada.

A detecção de um novo tempo é análogo ao SRTN, a diferença é que ele detecta a mudança de um quantum, isto é, foi acrescido um quantum no tempo atual. Outra diferença, que caracteriza o Round-Robin, é que sempre que há um elemento na nossa estrutura de dados e há execução de um processo, há preempção após um quantum passar.

Os semáforos funcionam exatamente como no SRTN (vetor de empty e full).

Utilizamos Quantum = 0.5s.

Resultados

Resultados

Os resultados foram coletados com base em um input gerado aleatoriamente e que respeita as seguintes propriedades:

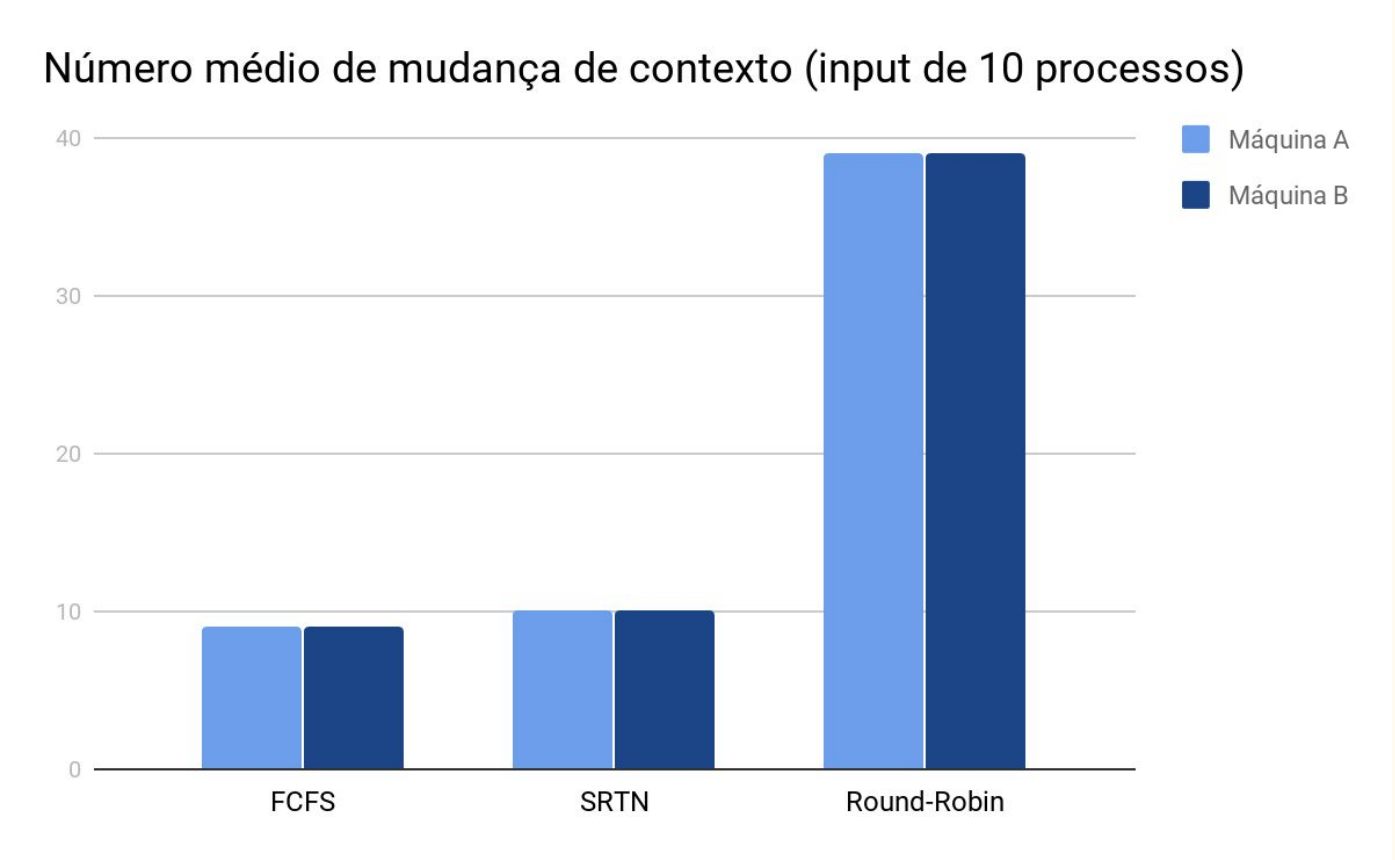
1. t0 do próximo processo é sempre maior ou igual ao do processo anterior
2. $dt = 1 + rand() \% 3$
3. $deadline = t0 + dt + rand() \% 5$

Com esse input, levamos cerca de 10h para testar 30 vezes os 3 escalonadores com uma entrada de tamanho 200. Nesse sentido, utilizamos valores de dt entre 1 e 3, pois com valores maiores não teríamos tempo suficiente para realizar os testes desejados.

Os escalonadores foram testados em duas máquinas A e B. A máquina A possui 4 CPUs e a máquina B possui 2 CPUs.

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

Mudança de Contexto para um input de 10 processos



Intervalos de confiança para o número médio de mudanças de contexto (input de 10 processos)

Para a máquina A

IC da média amostral para FCFS: [9,9]

IC da média amostral para SRTN: [10,10]

IC da média amostral para Round Robin: [39,39]

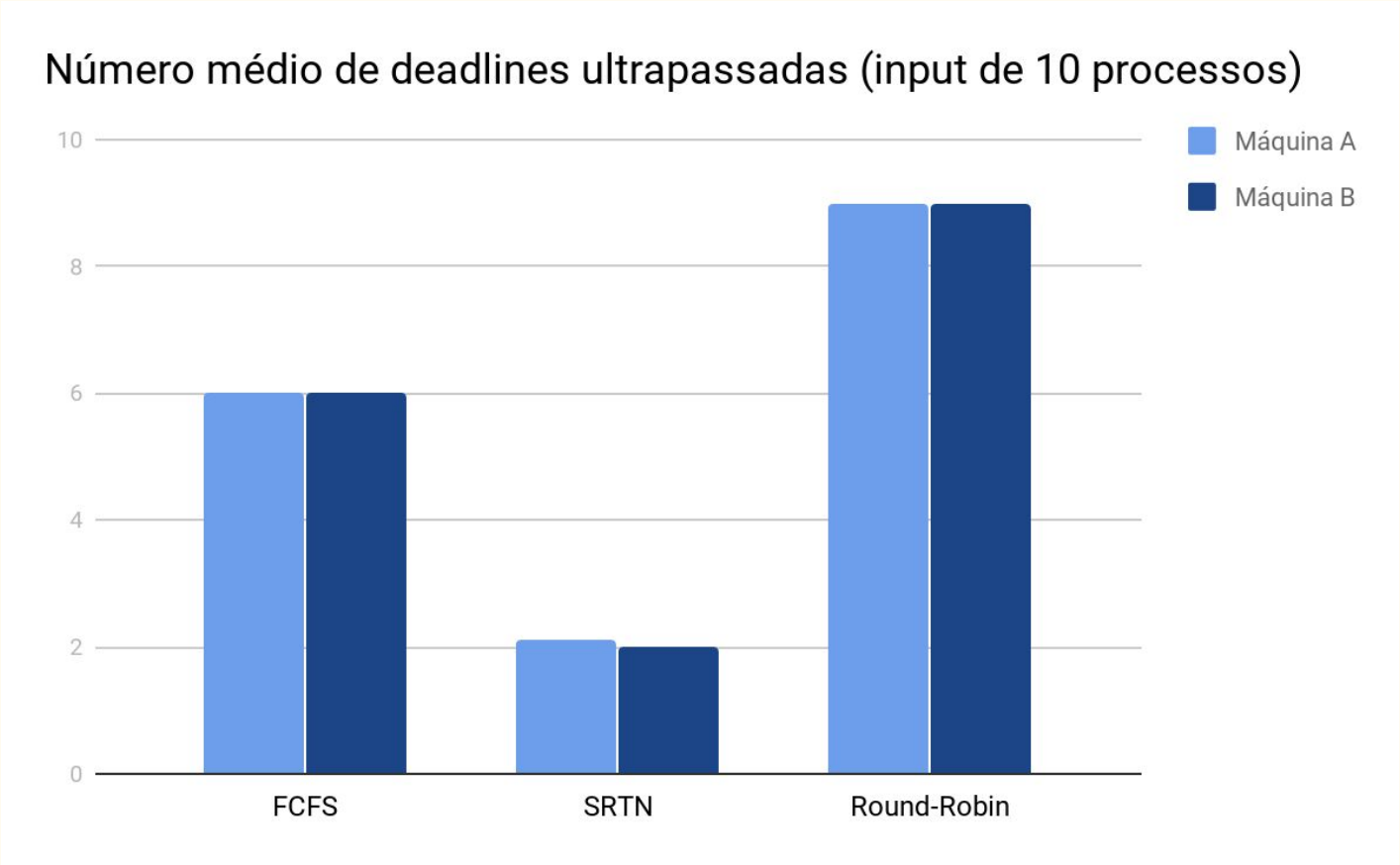
Para a máquina B

IC da média amostral para FCFS: [9,9]

IC da média amostral para SRTN: [10,10]

IC da média amostral para Round-Robin: [39,39]

Deadlines ultrapassadas para um input de 10 processos



Intervalos de confiança para o número médio de deadlines ultrapassadas (input de 10 processos)

Para a máquina A

IC da média amostral para FCFS: [6,6]

IC da média amostral para SRTN: [1.9, 2.3]

IC da média amostral para Round Robin: [9,9]

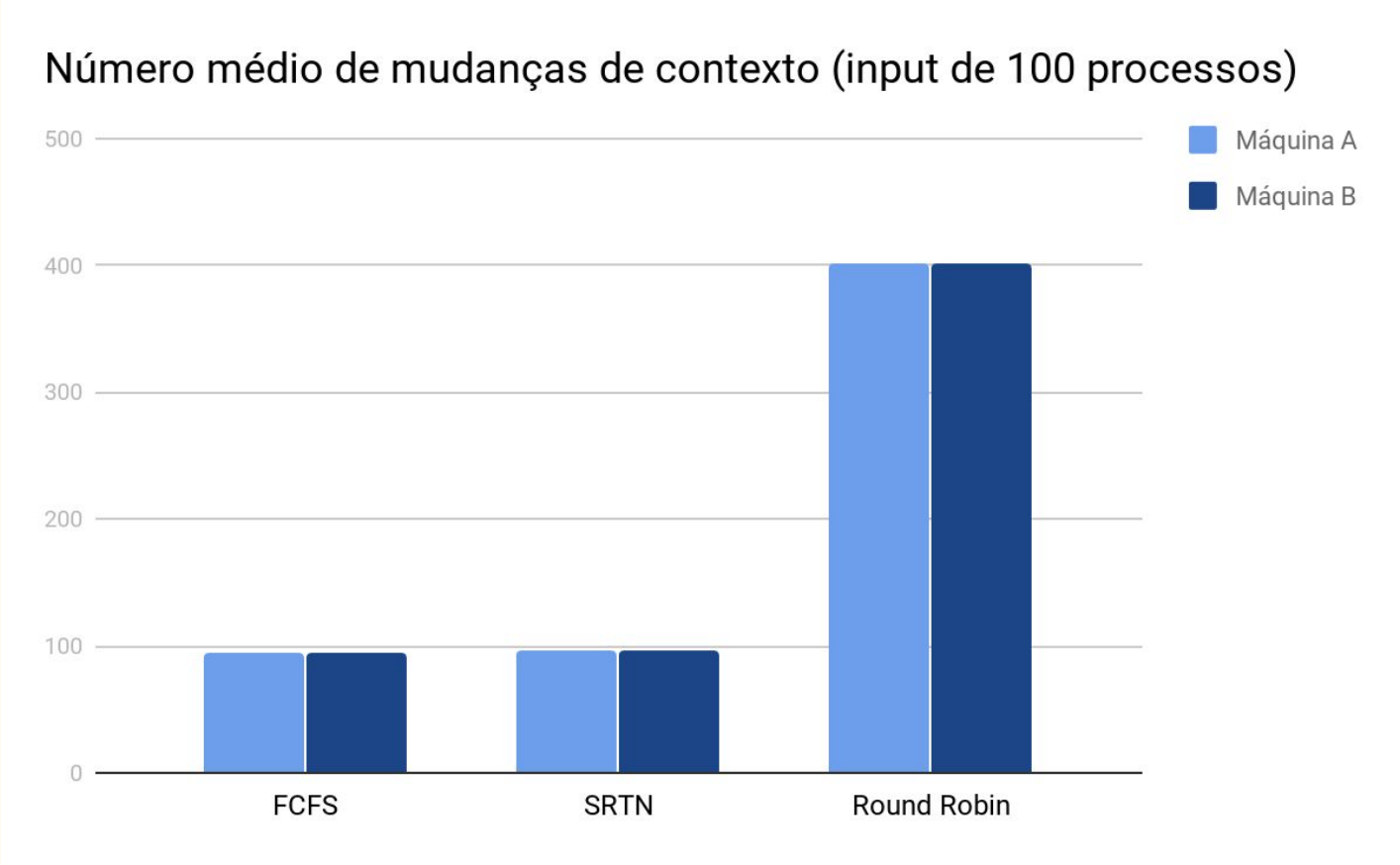
Para a máquina B

IC da média amostral para FCFS: [6,6]

IC da média amostral para SRTN: [2,2]

IC da média amostral para Round-Robin: [9,9]

Mudanças de contexto para um input de 100 processos



Intervalos de confiança para o número médio de mudanças de contexto (input de 100 processos)

Para a máquina A

IC da média amostral para FCFS: [94.9, 95.04]

IC da média amostral para SRTN: [97, 97]

IC da média amostral para Round Robin: [402,402]

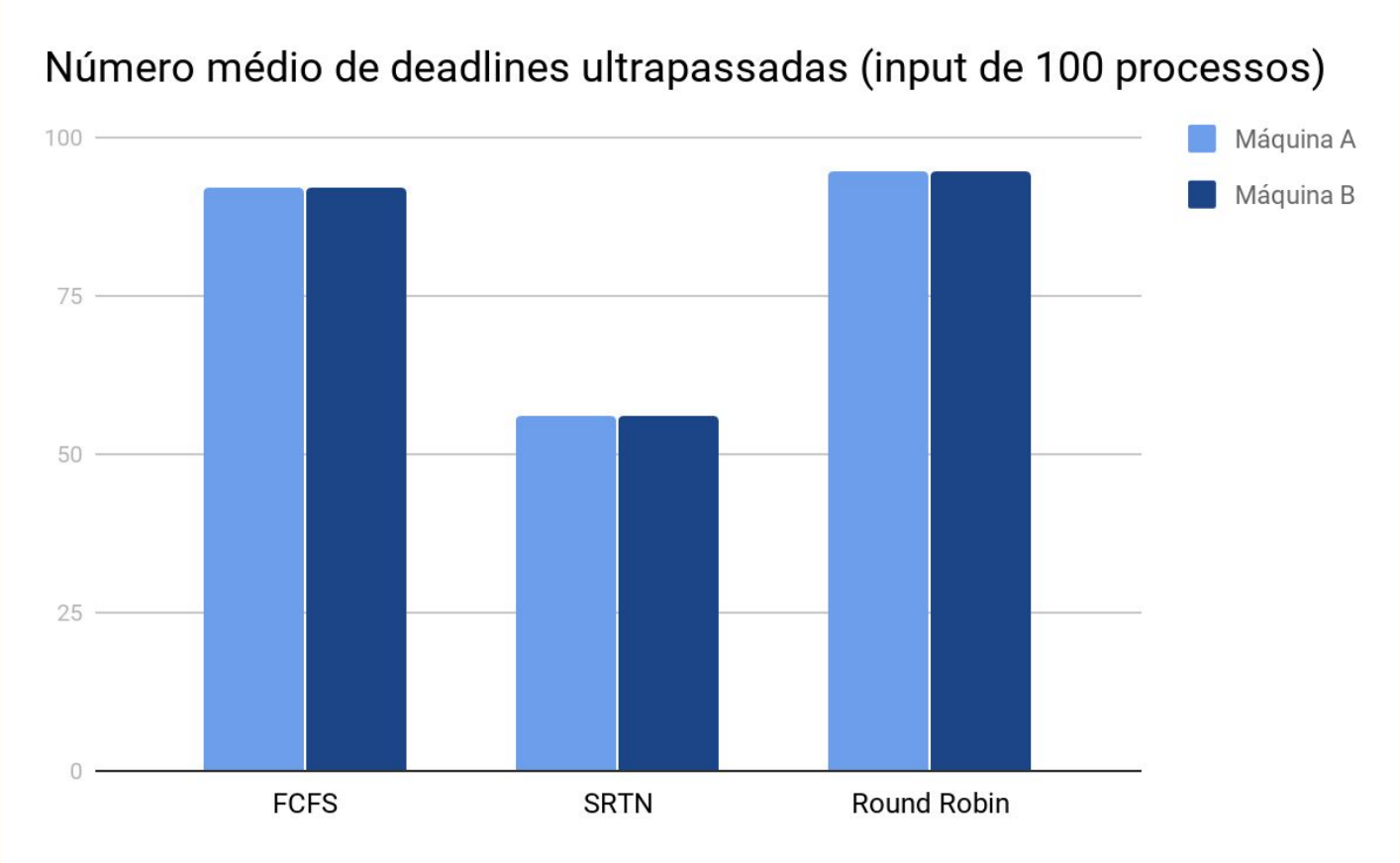
Para a máquina B

IC da média amostral para FCFS: [94.84, 95.03]

IC da média amostral para SRTN: [97,97]

IC da média amostral para Round-Robin: [402,402]

Deadlines ultrapassadas para um input de 100 processos



Intervalos de confiança para o número médio de deadlines ultrapassadas (input de 100 processos)

Para a máquina A

IC da média amostral para FCFS: [92,92]

IC da média amostral para SRTN: [56, 56]

IC da média amostral para Round Robin: [94.31,94.68]

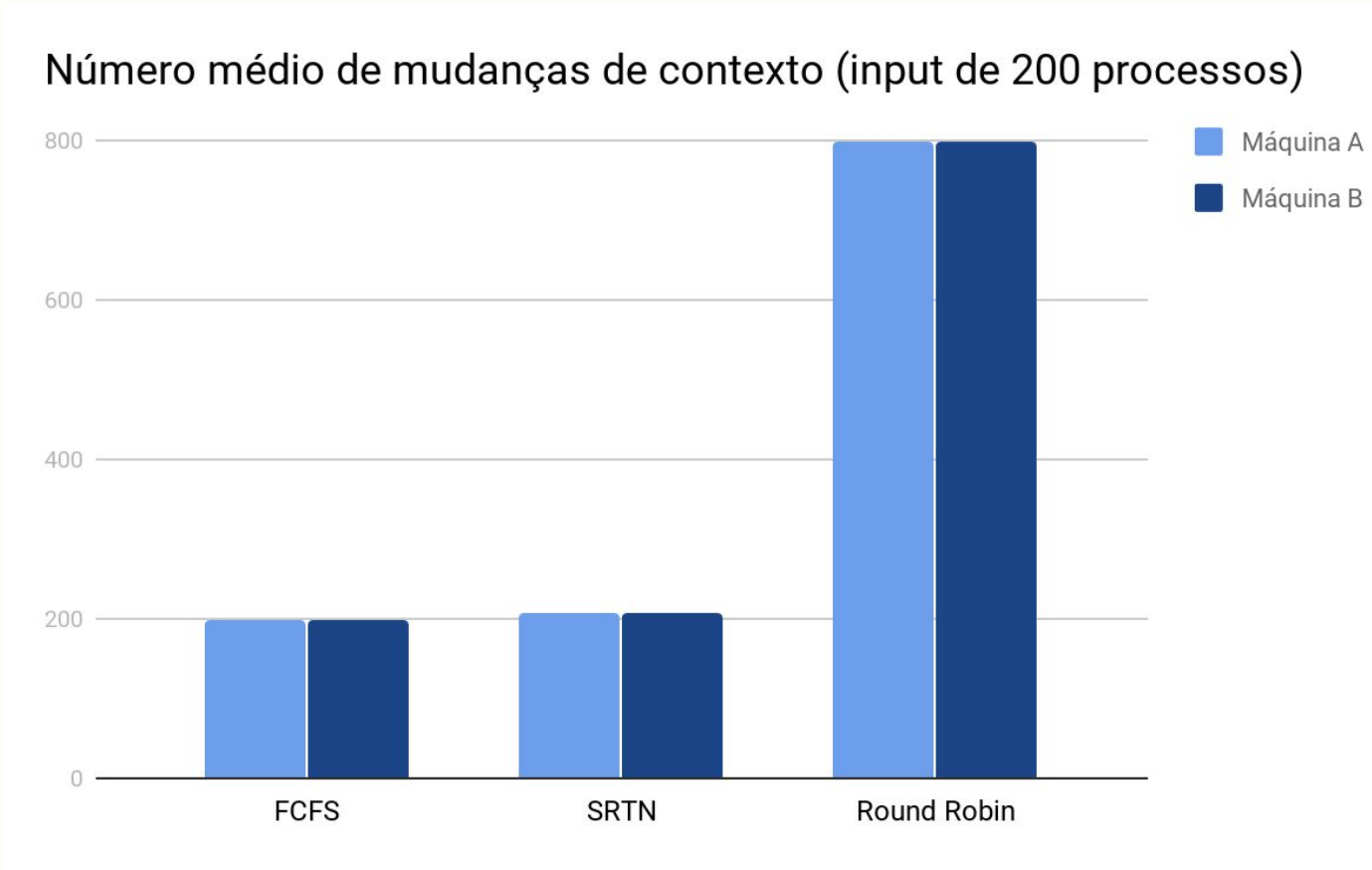
Para a máquina B

IC da média amostral para FCFS: [92,92]

IC da média amostral para SRTN: [56,56]

IC da média amostral para Round-Robin: [94.31,94.68]

Mudanças de contexto para um input de 200 processos



Intervalos de confiança para o número médio de mudanças de contexto (input de 200 processos)

Para a máquina A

IC da média amostral para FCFS: [199, 199]

IC da média amostral para SRTN: [208, 208]

IC da média amostral para Round Robin: [799, 799]

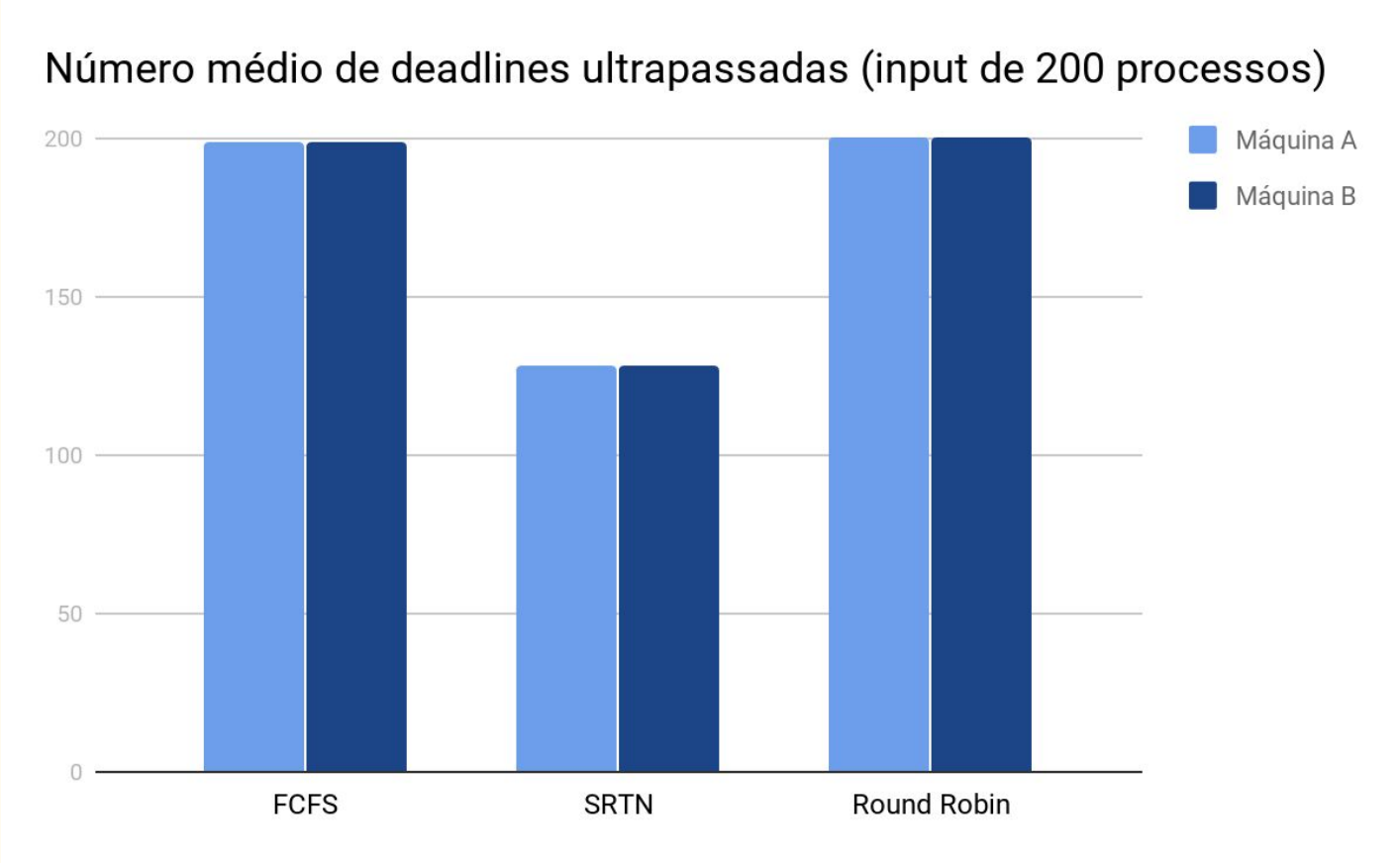
Para a máquina B

IC da média amostral para FCFS: [199, 199]

IC da média amostral para SRTN: [208,208]

IC da média amostral para Round-Robin: [799,799]

Deadlines ultrapassadas para um input de 200 processos



Intervalos de confiança para o número médio de deadlines ultrapassadas (input de 200 processos)

Para a máquina A

IC da média amostral para FCFS: [199,199]

IC da média amostral para SRTN: [128, 128]

IC da média amostral para Round Robin: [200,200]

Para a máquina B

IC da média amostral para FCFS: [198.42, 198.91]

IC da média amostral para SRTN: [128,128]

IC da média amostral para Round-Robin: [200,200]

Conclusão

Percebemos que, pela quantidade de testes nas duas máquinas, os algoritmos de escalonamento se comportaram de maneira quase determinística, com resultados praticamente iguais em todos os 30 testes para cada escalonador.

Quanto às mudanças de contexto, percebemos que o Round-Robin, pela sua característica de preempção a cada Quantum, possui maior quantidade de mudanças de contexto que os outros dois para um mesmo input.

Round-Robin e FCFS possuem maior quantidade de deadlines ultrapassadas para um mesmo input. Isso acontece porque o FCFS simplesmente roda o processo até ele acabar e o Round-Robin executa apenas um quantum e vai para o próximo processo. Essas duas estratégias acabam acumulando muitos processos, causando o não cumprimento da deadline. Em contrapartida, o SRTN busca executar o processo com menor dt, favorecendo o cumprimento da deadline.

Conclusão

Com input gerado aleatoriamente, a quantidade média de mudanças de contexto do FCFS e do SRTN ficaram bem próximas, pois o dt escolhido nos inputs variam entre 1 e 3 de maneira equiprovável. Isso acaba afetando um pouco a característica do SRTN de executar o processo com menor dt. Se o tempo de execução de um processo tivesse um intervalo maior de valores possíveis, haveria maior probabilidade de ocorrer preempção pelo dt no SRTN, aumentando a quantidade de mudanças de contexto.