EP1 Sistemas Operacionais

Lara Ayumi Nagamatsu, 9910568
Lucy Anne de Omena Evangelista, 11221776

Parte 1: Shell

Shell BCCSH

A ideia geral da implementação do BCCSH era realizar a execução externa e interna de comandos a partir de chamadas de sistema (syscalls).

Para a invocação externa de comandos, utilizamos a função fork() para fazer uma cópia do processo pai, e, mais tarde, usamos o execve() para mudar a imagem do processo filho, utilizando os devidos argumentos para cada comando exigido.

Utilizamos recursos do GNU/linux para atrelarmos a funcionalidade de histórico no terminal, com uso das bibliotecas readline.h e history.h

Shell BCCSH

Para as outras syscalls (comandos embutidos), não utilizamos a chamada de sistema da família exec*, em vez disso, utilizamos o comando strace no shell do terminal para checar quais eram as chamadas de sistema utilizadas para se realizar os comando exigidos (mkdir <dirname>, kill -9 <kill_pid>, ln -s <arquivo> <link>). Essas foram as funções utilizadas:

- mkdir(dirname, 0777);
- kill(kill_pid, SIGKILL);
- symlinkat(arquivo, AT_FDWCD, link);

Parte 2: Escalonadores

Decisões de Projeto

- utilizamos structs: Data (para armazenar processos) e Node (para a lista ligada usada no escalonador SRTN)
- estabelecemos 3 estados para as threads (usados no escalonador SRTN): Executando, Dormindo ou Espera
- criamos funções auxiliares para lidar com o processamento e criação de arquivos trace
- a ideia do nosso programa era utilizar apenas uma CPU para processamento, ou seja, apenas uma thread executaria por vez (isso funcionou para o FCFS, mas para o SRTN as CPUs mudavam a todo momento)

Implementações gerais

Em geral, a implementação do FCFS em comparação com o SRTN é completamente diferente. Para ambos, como esperado, foram usados semáforos mutex para o controle do acesso à memória.

Entretanto, o controle do tempo para o FCFS foi feito majoritariamente pela thread em execução, enquanto para o escalonador SRTN, o próprio escalonador que exerceu controle.

Por fim, implementamos dois dos três escalonadores.

Escalonador First-Come First-Served

O FCFS é baseado em FIFO (first in, first out) e não possui preempção.

Dessa forma, escolhemos criar um laço para iterar desde o primeiro item do vetor de processos ao último. Para possibilitar apenas uma thread rodando por vez, incluímos o mutex mutex_proc (além do utilizado dentro da função de criação de thread) para “travar” o programa até que a thread em execução acabasse seu processamento.

Escalonador Shortest Remaining Time Next

O escalonador SRTN é basicamente o SJN (Shortest Job Next) com preempção. A ideia na implementação deste escalonador foi criar todas as threads de processos no início da execução do programa como “dormindo” (isso foi feito a partir de sinais com pthread_cond_wait() e pthread_cond_signal() para cada um dos processos). Então, checando o tempo executado no programa, a fila de processos prontos é atualizada e checa-se se há thread executando que deve ser destruída. A partir desse ponto, são feitas as comparações das entidades da fila com o processo atual em execução, se este não foi eliminado. Se houver necessidade, há substituição do processo em execução (que volta para a fila) pelo primeiro da fila. Por fim, incrementa-se a variável de tempo do programa.

Realizando os experimentos

Após as implementação dos escalonadores, realizamos os testes com repetições de 30 entradas para cada entrada.

As entradas consistem em:

- entrada pequena: 10 processos
- entrada média: 20 processos
- entrada grande: 30 processos

Essa quantidade de processos foi escolhida para conseguirmos realizar todos os cálculos para todos os cenários necessários.

Realizando os experimentos

Foi realizado um jupyter notebook em Julia para automação dos experimentos, que para cada escalonador repetia a execução de cada uma das três entradas 30 vezes.

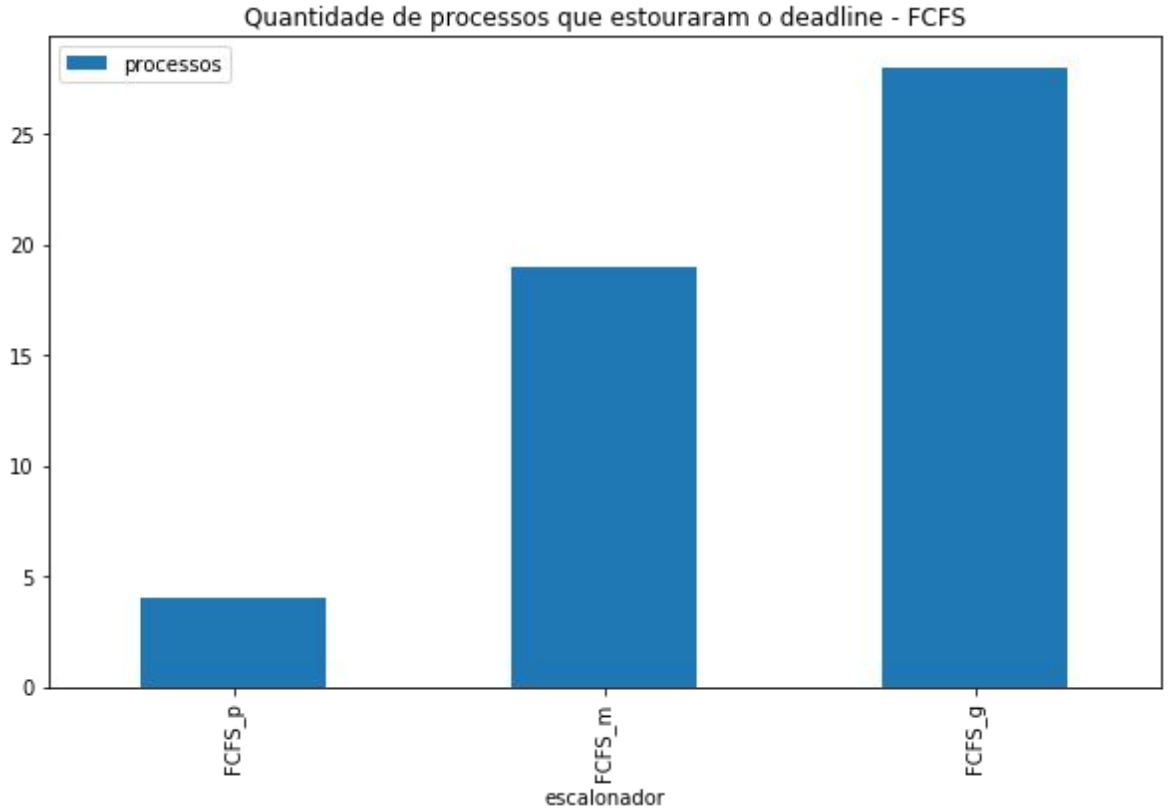
Ao final de sua execução, o programa salva os arquivos em .txt e calcula as métricas de intervalo de confiança para cada um dos contextos: escalonador e tamanho de entrada.

Realizando os experimentos

Durante a realização dos experimentos foi possível perceber e comprovar o comportamento linear e constante da implementação do FCFS. Acreditamos que isso ocorre pois usamos uma única CPU para as implementações, mesmo que utilizássemos computadores de processadores diferentes.

Também verificamos esse comportamento (mesmo tempo de execução e quantidade de trocas de contexto) no segundo escalonador. Nesse caso, como tivemos problema para controlar em qual CPU os processos estavam rodando por vez, acreditamos que esse comportamento se deve ao fato de que uma única CPU seria utilizada no processo.

Comparações



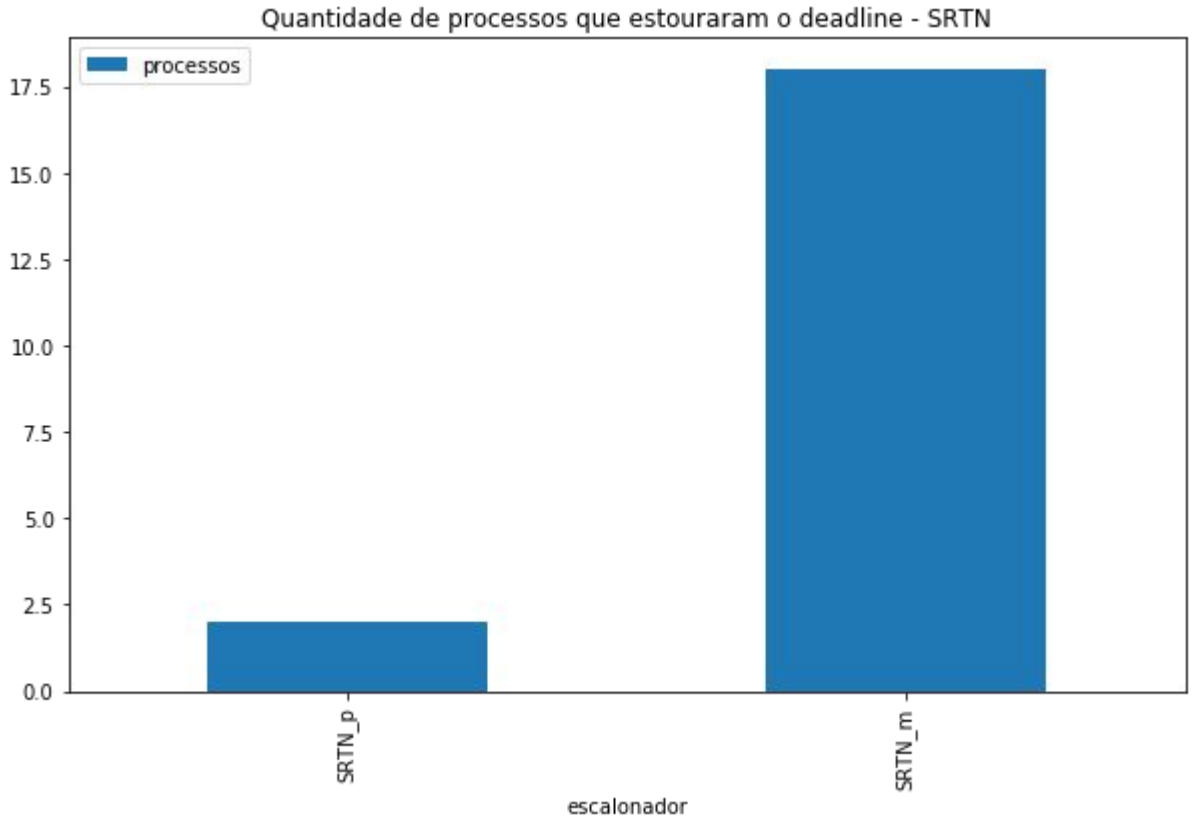
Comparações

Com o FCFS podemos perceber que quando há o atraso na realização de uma tarefa, os processos se acumulam.

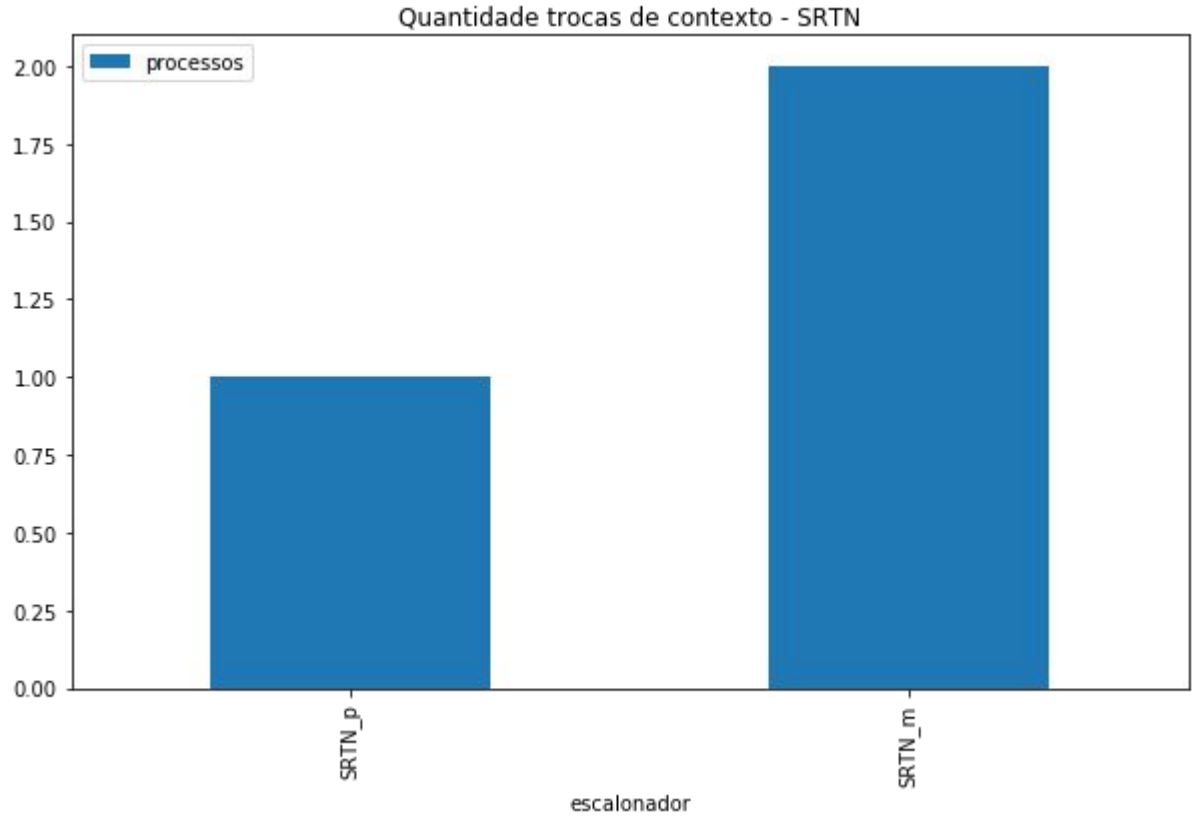
Para a nossa implementação que considera uma CPU, esse comportamento é nítido. Para o menor caso (10 processos) 4 se acumularam; para 20 processos, 18; e para 30 processos, 28.

O intervalo de confiança é 0 devido a implementação em única CPU.

Comparações



Comparações



Comparações

Para o SRTN, apesar das dificuldades de implementação que resultaram no fato de que somente duas das três saídas conseguiram ser computadas, os resultados obtidos pelo escalonamento de processos é melhor. A quantidade de processos que se acumulam em relação ao FCFS é menor.