MAC0422 - Sistemas Operacionais

EP1 - 2017

Marcelo Trylesinski Vinicius Agostini

NUSPs: 9297996 e 4367487

Bacharelado em Ciência da Computação
Universidade de São Paulo

11 de Setembro de 2017

Outline

Outline

Arquitetura do Shell

- `getcwd()` devolve o diretório atual

Arquitetura do Shell

- `getcwd()` devolve o diretório atual
- `using_history()` para criar histórico de comandos no shell

Arquitetura do Shell

- `getcwd()` devolve o diretório atual
- `using_history()` para criar histórico de comandos no shell
- `readline()` imprime o prompt e lê uma linha inteira de entrada do usuário

Arquitetura do Shell

- `getcwd()` devolve o diretório atual
- `using_history()` para criar histórico de comandos no shell
- `readline()` imprime o prompt e lê uma linha inteira de entrada do usuário
- `add_history()` adiciona o comando ao histórico do shell

Arquitetura do Shell

- `getcwd()` devolve o diretório atual
- `using_history()` para criar histórico de comandos no shell
- `readline()` imprime o prompt e lê uma linha inteira de entrada do usuário
- `add_history()` adiciona o comando ao histórico do shell
- tokeniza o input para separar o comando e os argumentos correspondentes

Arquitetura do Shell

- `getcwd()` devolve o diretório atual
- `using_history()` para criar histórico de comandos no shell
- `readline()` imprime o prompt e lê uma linha inteira de entrada do usuário
- `add_history()` adiciona o comando ao histórico do shell
- tokeniza o input para separar o comando e os argumentos correspondentes
- funções separadas para execução de binários ou comandos embutidos

Outline

Executando Binários

- cria novo processo - `fork()`
- executa o comando neste processo - `execve()`
- processo pai espera o término do filho - `waitpid()`

Outline

Comandos Embutidos

- `chown(path, owner, group)`
- `time()`, `localtime()` e `strftime()`

Outline

main

- Argumentos
  - escalonador desejado
  - caminho para o arquivo de trace
  - caminho para o arquivo de saída
  - flag opcional para o log de debug
- Processa o arquivo de trace linha por linha, adicionando um `Process` ao vetor de processos
- `switch` determina qual algoritmo rodar baseado na escolha do usuário

Shortest Job First

- `heap` de `Process`
- comparador para ordenar por `dt`
- insere processos quando chegam em $\mathcal{O}(\log n)$ e determina o próximo processo a ser executado em $\mathcal{O}(1)$.

Round Robin

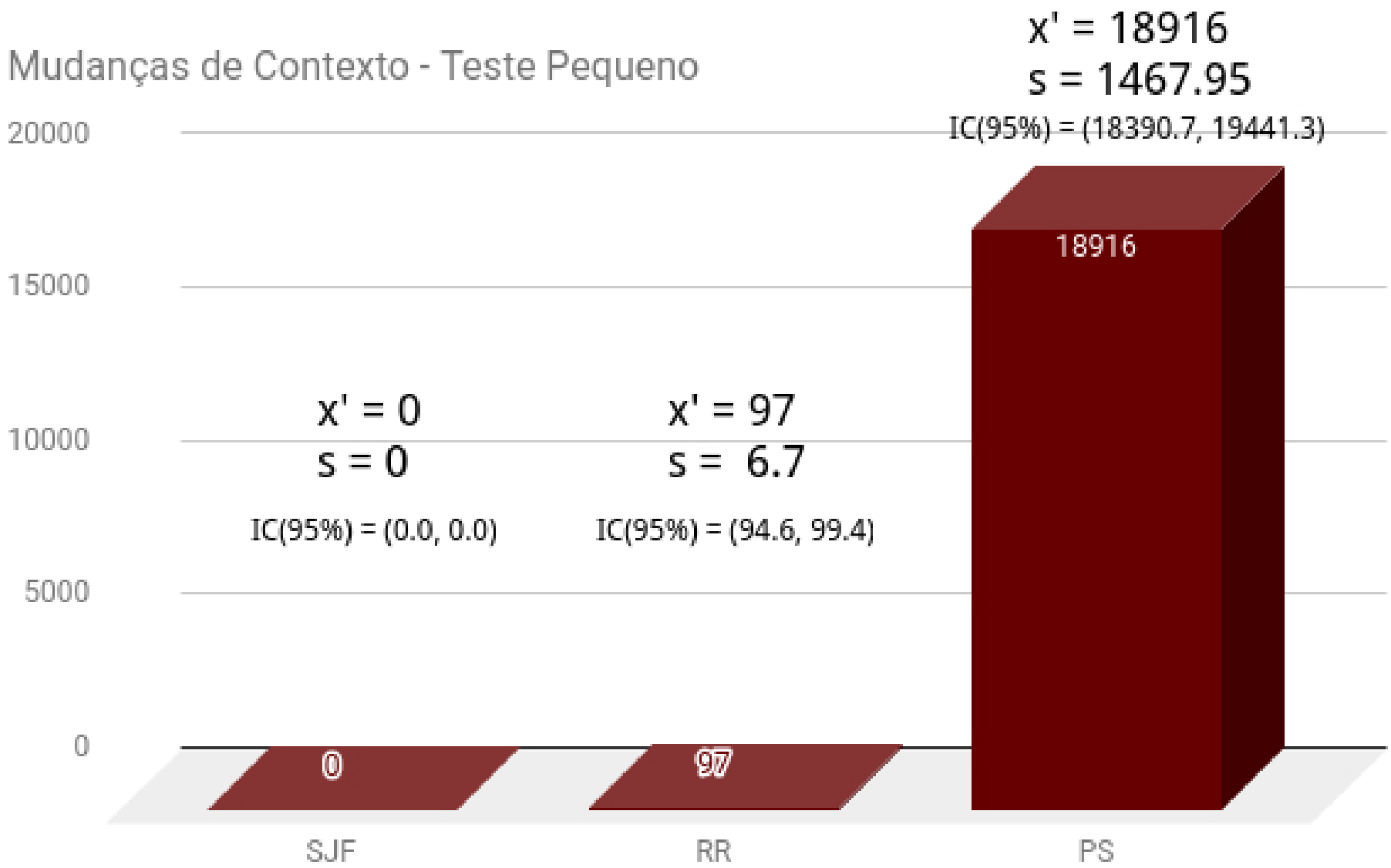
- fila circular de `Process`
- `enqueue()` e `dequeue()` em $\mathcal{O}(1)$.
- lista ligada = sem redimensionamento
- `quantum = 0.5`

Escalonamento com prioridade

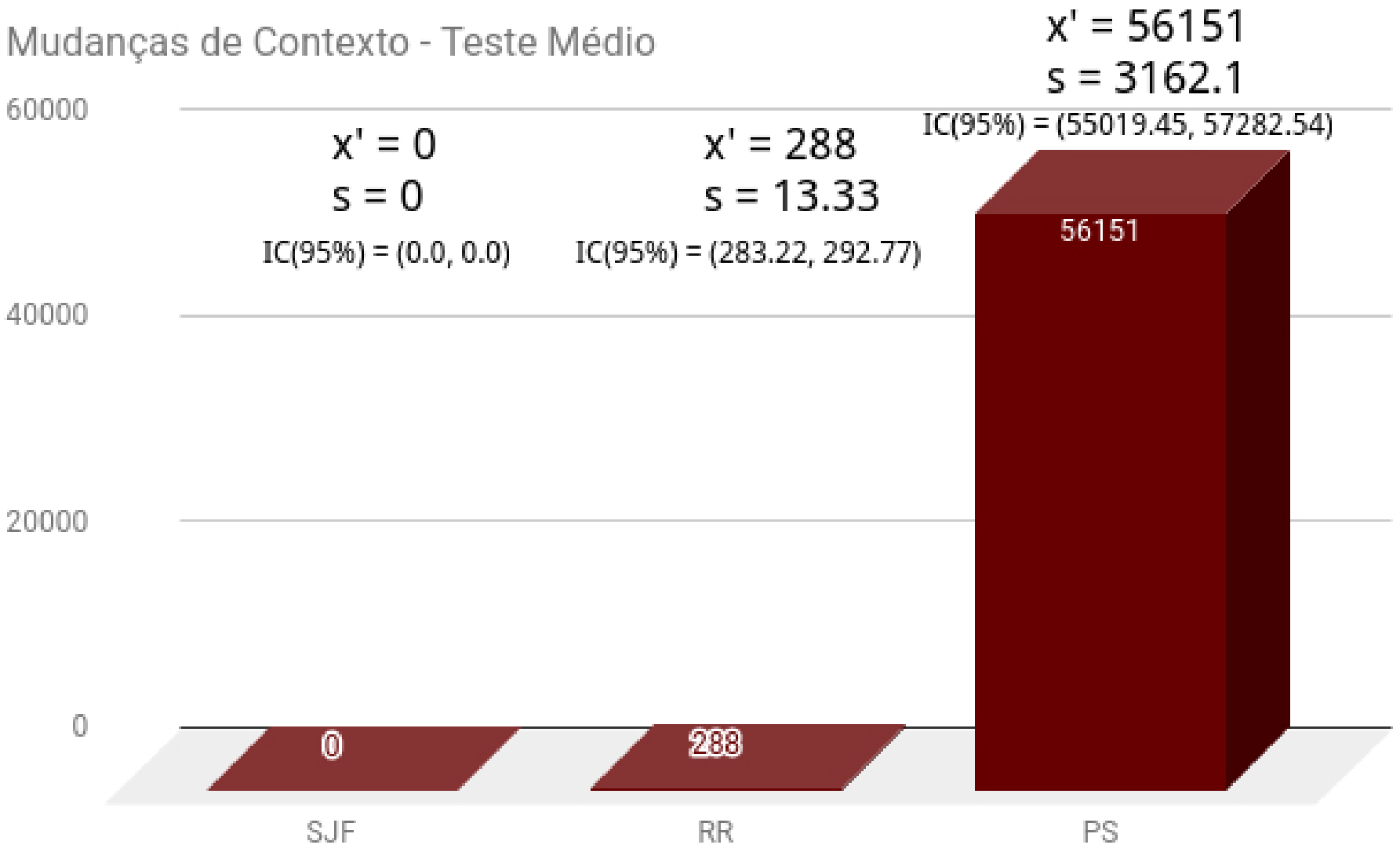
- heap de `Process`
- comparador para ordenar por `deadline`
- roda processo com menor deadline
- usa a função `1/deadline * quantum` para determinar quanto tempo o processo vai rodar
- usamos o mesmo quantum do Round Robin e causou muitas mudanças de contexto desnecessárias devido à função acima

Outline

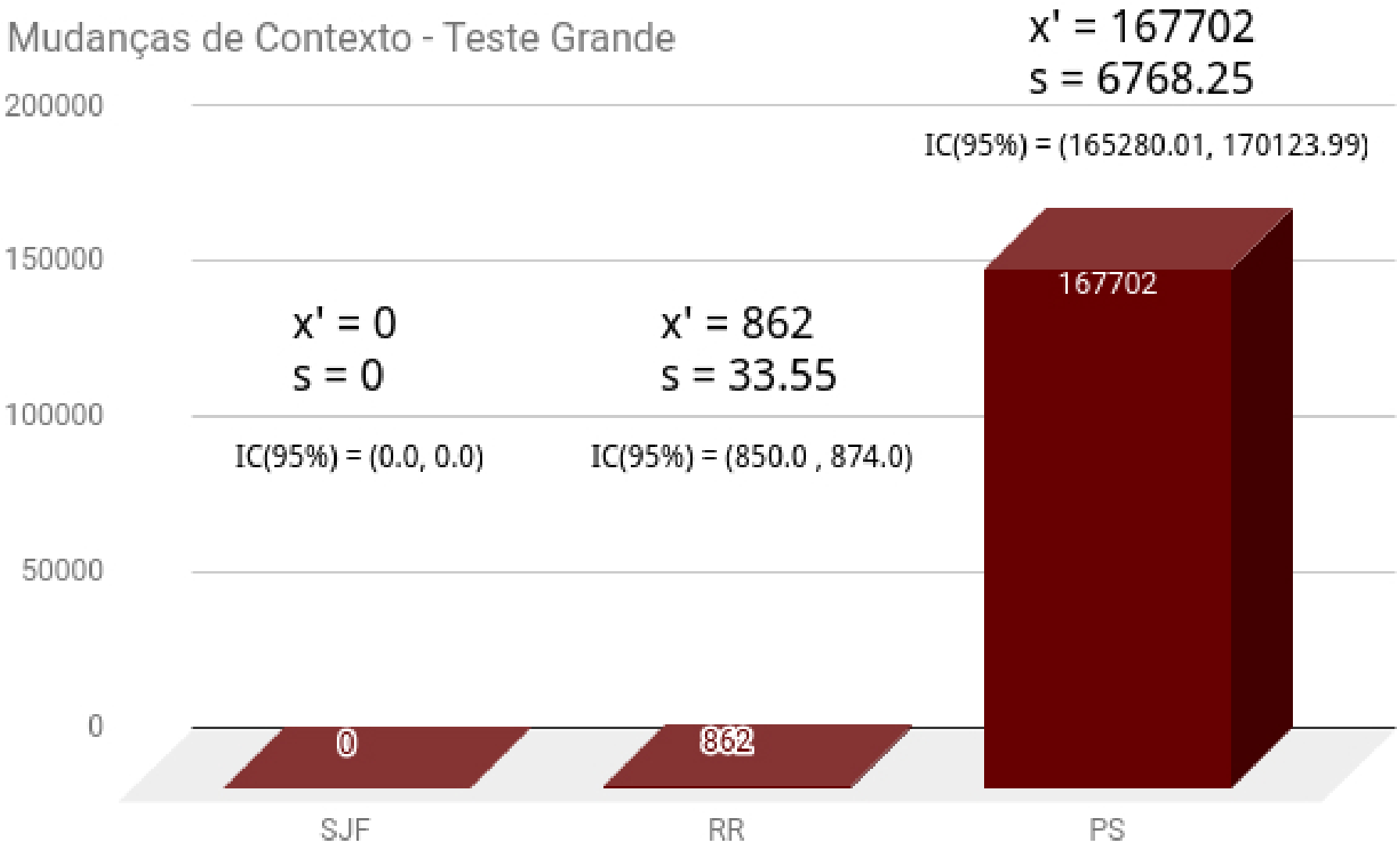
Testes Pequenos (50 processos)



Testes Médios (150 processos)

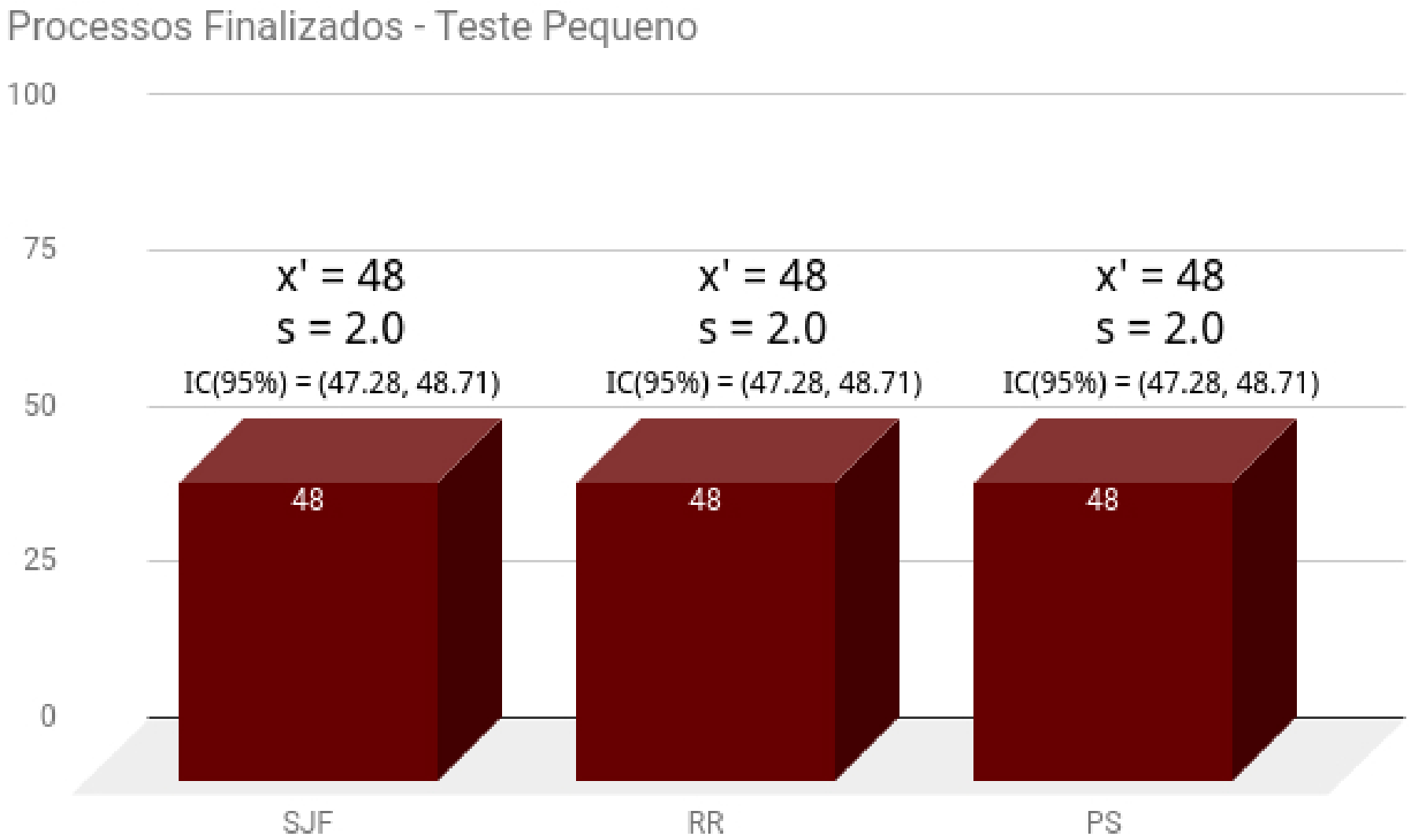


Testes Grandes (450 processos)

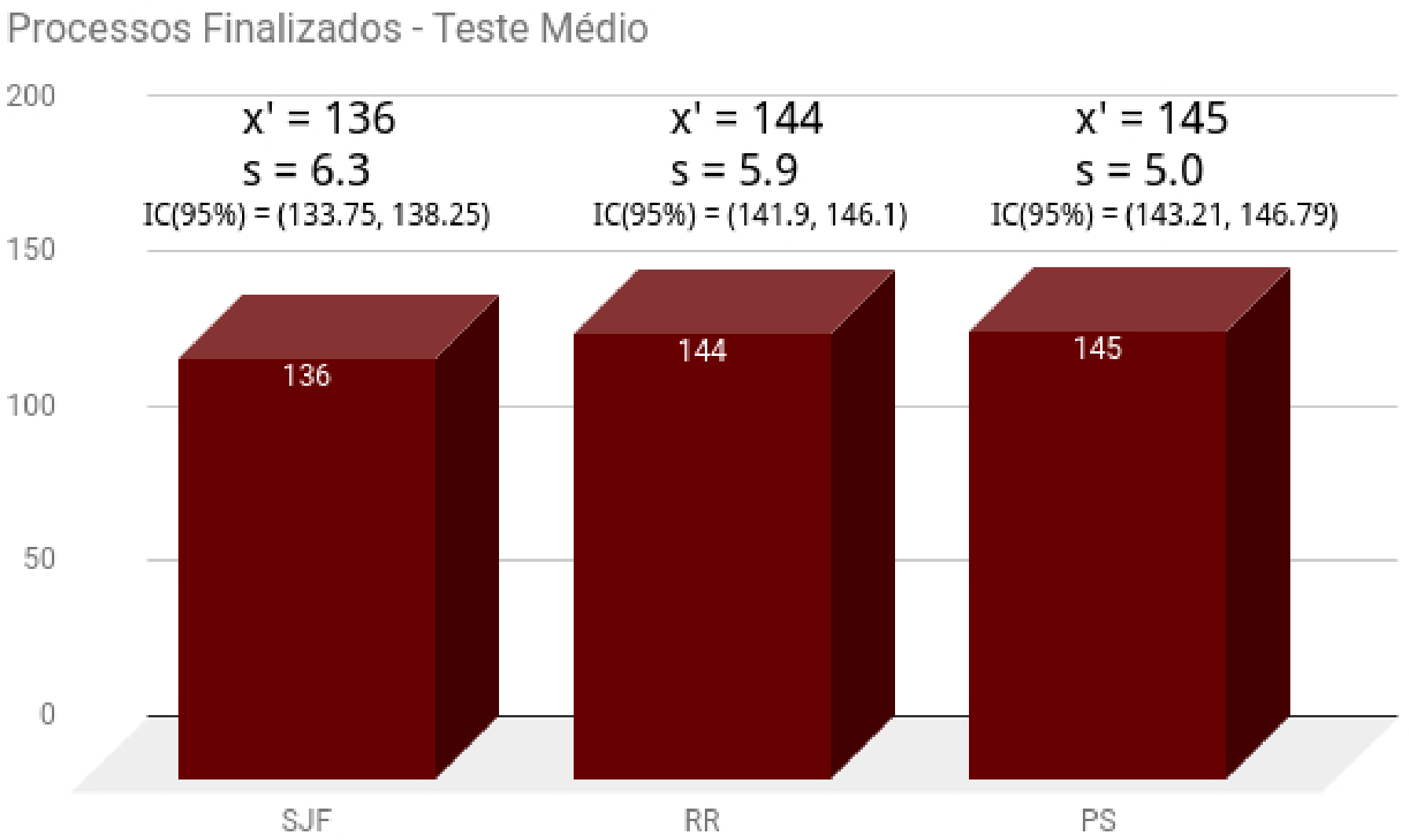


Outline

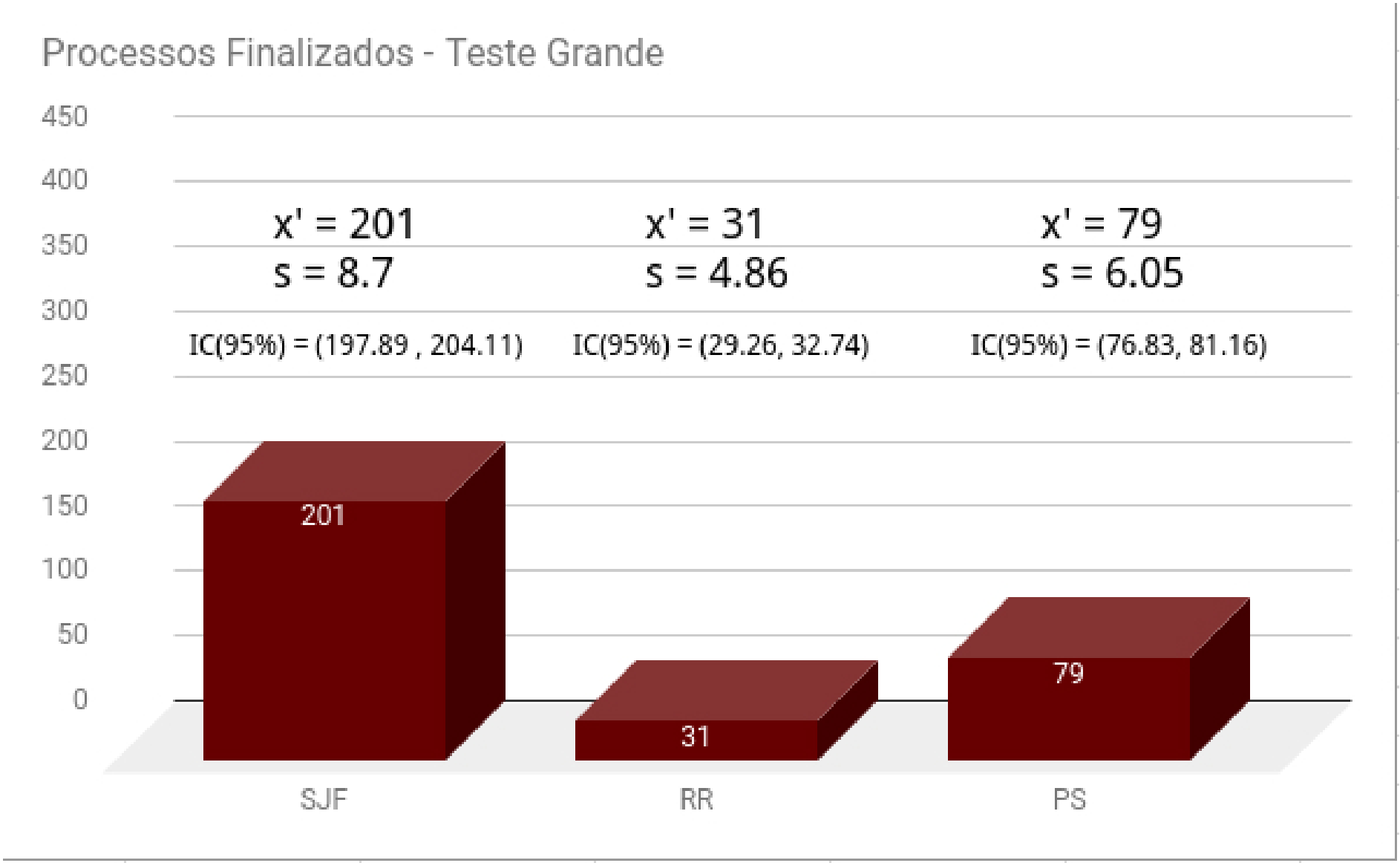
Testes Pequenos (50 processos)



Testes Médios (150 processos)



Testes Grandes (450 processos)



Considerações

- Round Robin e Escalonamento com prioridade sofreram muito quando a quantidade de processos aumentou
- SJF também sofreu, mas muito menos
- Os inputs acabaram causando filas longas
- Os processos não conseguem terminar a tempo se tiverem que esperar uma eternidade
- balanço entre `t0`, `dt`, `deadline`