EP1 – MAC0422
SISTEMAS OPERACIONAIS

INTEGRANTES / NUSP:

BRUNO MAZETTI SAITO | 11221838

WILLIAN HIROSHI TAKIHI | 11221755

ROTEIRO

- Arquitetura do Shell;
- Escalonadores de Processos;
- Testes;
- Gráficos;
- Resultados e Conclusão.

Arquitetura do Shell

- Função *read_prompt ()* para ler a linha de comando do Shell;
- Função *type_prompt ()* para imprimir o nome de usuário e diretório atual no Shell;
- Binários executados com *execve()*;
- A seguir, é mostrado uma tabela com as funções utilizadas para implementar as chamadas de sistema:

Comando do *bash*	Chamada de sistema
mkdir <diretorio>	*mkdir (diretorio, 0777)*
kill -9 <PID>	*kill (PID, SIGKILL)*
ln -s <arquivo> <link>	*symlink (arquivo, link)*

- OBS: parâmetro 0777 dá permissão máxima para manipulação de diretório

Escalonadores de Processos

- Decisões de Projeto Geral
- Decisões de Projeto Específicas
  - First-Come First-Served (FCFS);
  - Shortest Remaining Time Next (SRTN);
  - Round Robin (RR).

Decisões de Projeto Geral

- Definido um número máximo de processos (maxProcessos) que pode ser alterado no arquivo *ep1.c*;
  - maxProcessos = 300000.
- Considerado uma mudança de contexto quando:
  - um processo interrompido para que outro possa ser executado (preempção);
  - um processo termina sua execução e há outro logo em sequência para ser executado.
- Utilizado uma biblioteca *header.h* na qual estão definidos:
  - *struct processo* que representa o processo a ser executado;
  - estrutura de dados fila para guardar próximos processos a serem executados e suas funções de manipulação;
  - fila implementada de maneira circular em um vetor.
- Considerado que a máquina apresenta uma CPU, ou seja, é executado uma *thread* por vez;

Decisões de Projeto Geral

- Escalonador atua em uma *thread* sempre em execução;
- Os processos executam a função *work ();*
  - executa operações de soma e subtração para o consumo da CPU;
  - utilizado semáforos e a variável global *finishedOp* para realizar a preempção de processos;
  - utilizado a variável global *finishedDef* para término de processos.
- Utilizado função *usleep ()* para gerenciamento do tempo.

Decisões de Projeto – FCFS

- Ciclos de 1 segundo para atuação da *thread*, gerenciado com a função *usleep ()*;

Decisões de Projeto – SRTN

- Ciclos de 1 segundo para atuação da *thread*, gerenciado com a função *usleep ()*;
- Utilizada a função *sortFila ()* presente no *header.h* para ordenação da fila de processos com base no tempo restante de execução;
  - executada sempre que novos processos chegam no sistema;
  - OBS: implementado com o algoritmo *bubblesort*, consequentemente filas de processos maiores afetam visivelmente no tempo de execução do escalonador.
- Se um novo processo chega no sistema com tempo restante de execução igual ao do processo atual, aquele que estava sendo executado possui prioridade na fila, ou seja, não é interrompido;

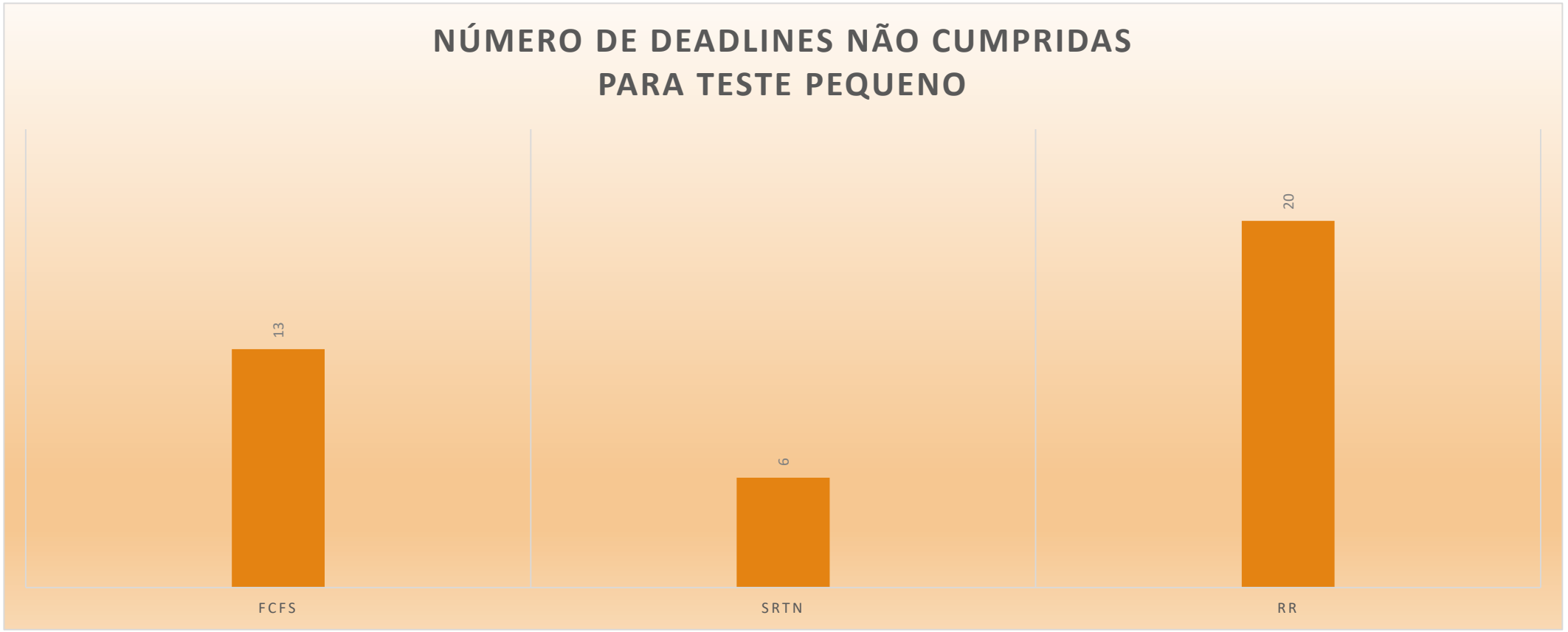
Decisões de Projeto – RR

- Definido um *quantum* na função do escalonador no arquivo *ep1.c*;
  - quantum = 200000, que representa 0.2 segundos em tempo real;
  - suposto que o valor de quantum é menor ou igual a 1000000 e divisor de 1000000.

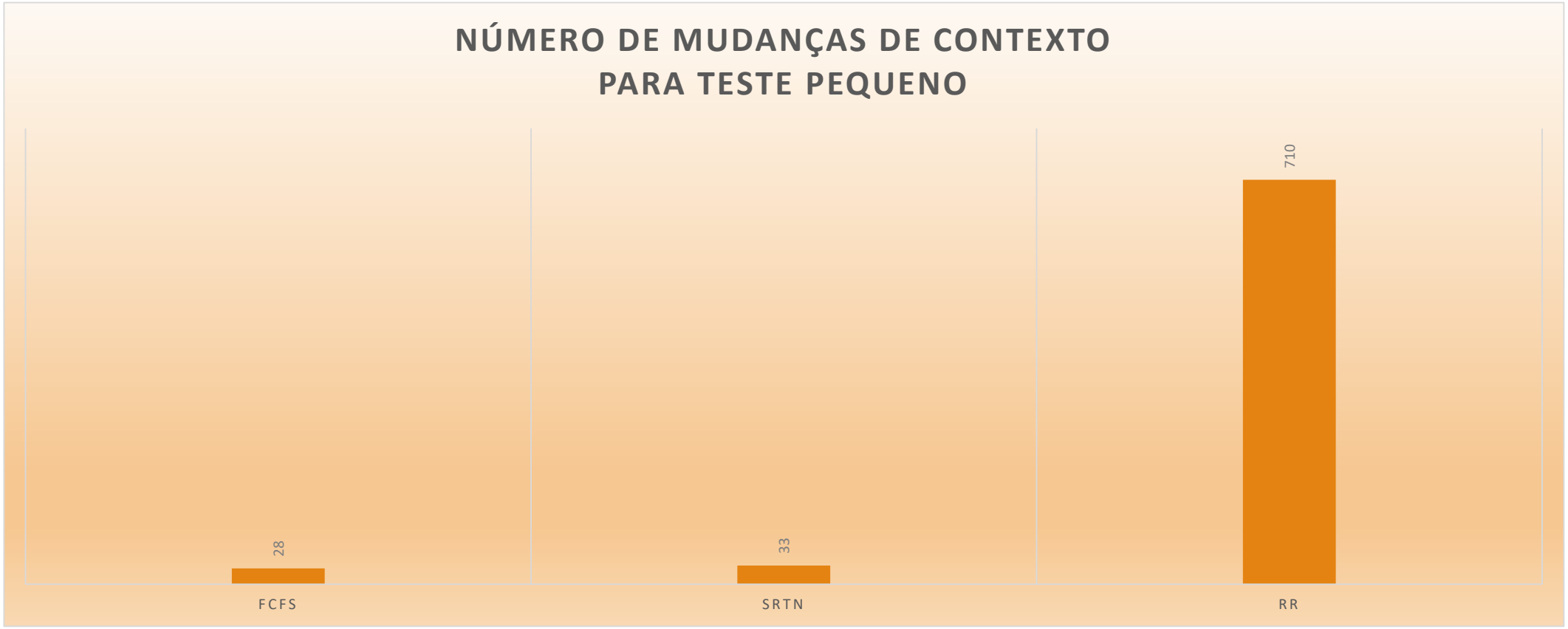
Testes

- O número de processos dos testes realizados são apresentados a seguir:
  - Trace com poucos processos – 30
  - Trace com número médio de processos – 100
  - Trace com muitos processos – 500
- Os valores dos parâmetros de cada processo dos testes são aleatórios;
  - utilizado as funções srand (time (0)) e rand ().
- Simulamos cada um dos testes 30 vezes;
- A simulação foi feita em dois computadores:
  - Máquina A - Processador i7-8550U: 4 cores com 8 threads;
  - Máquina B - Processador i7-7700HQ: 4 cores com 8 threads;

Gráficos – Máquina A



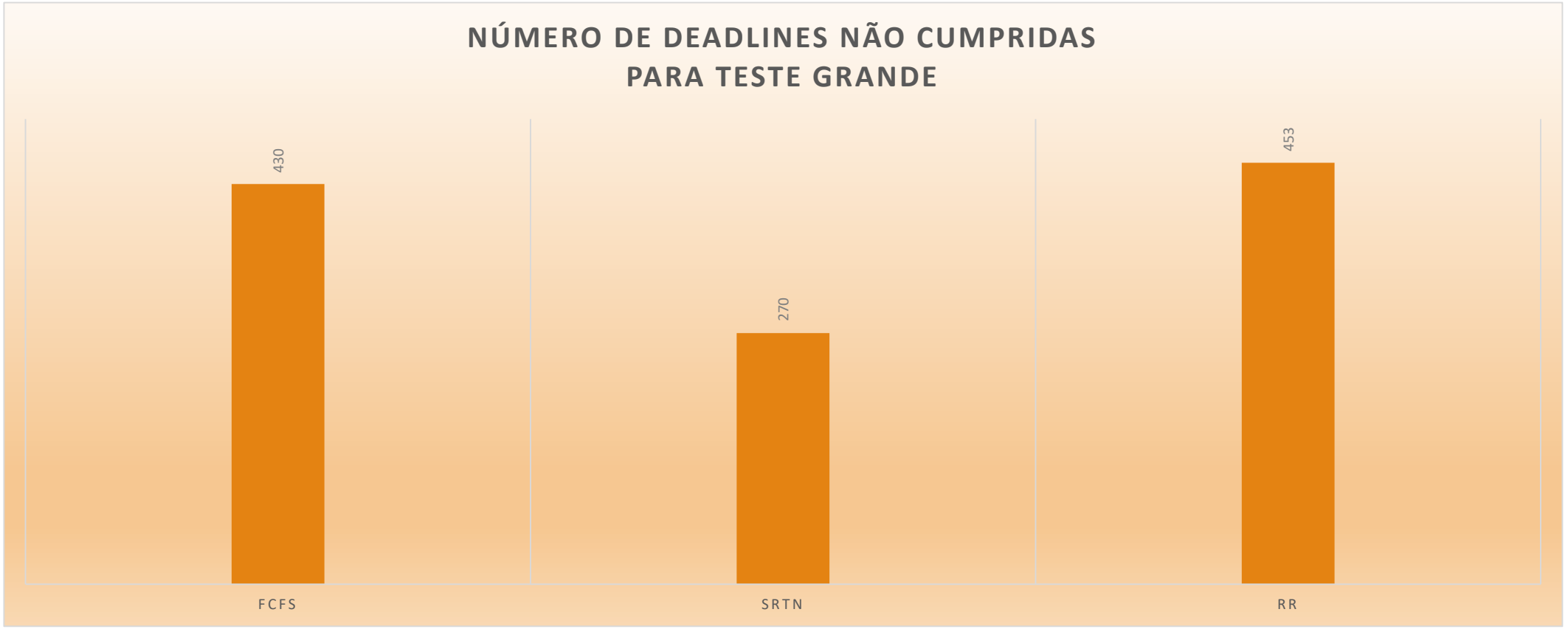
Gráficos – Máquina A



Gráficos – Máquina A

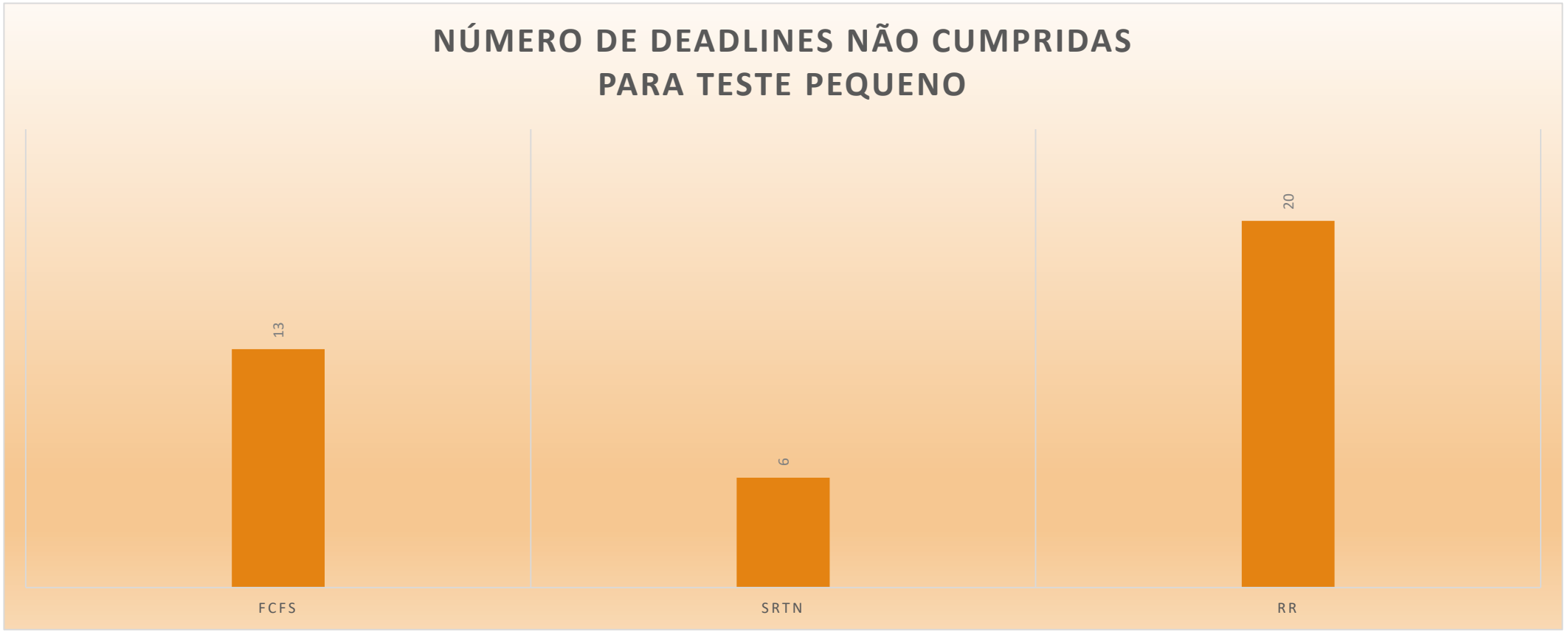
Gráficos – Máquina A

Gráficos – Máquina A

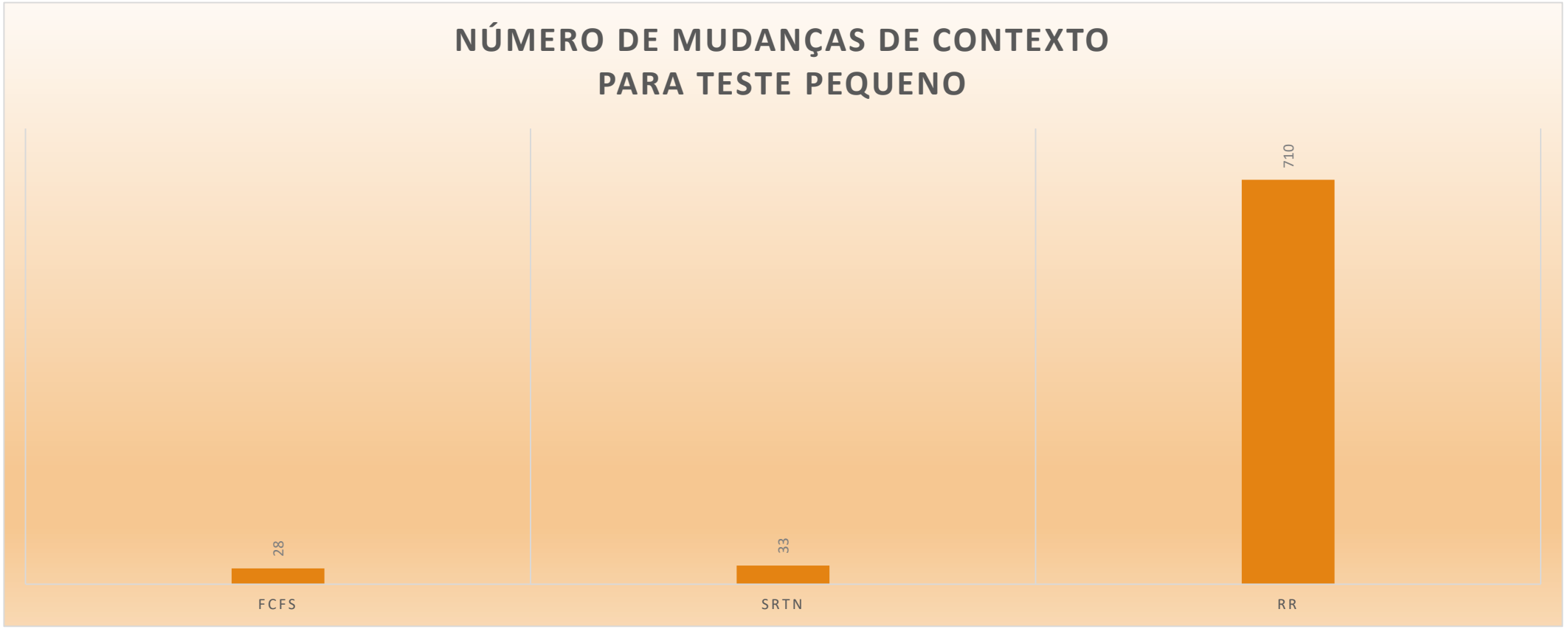


Gráficos – Máquina A

Gráficos – Máquina B



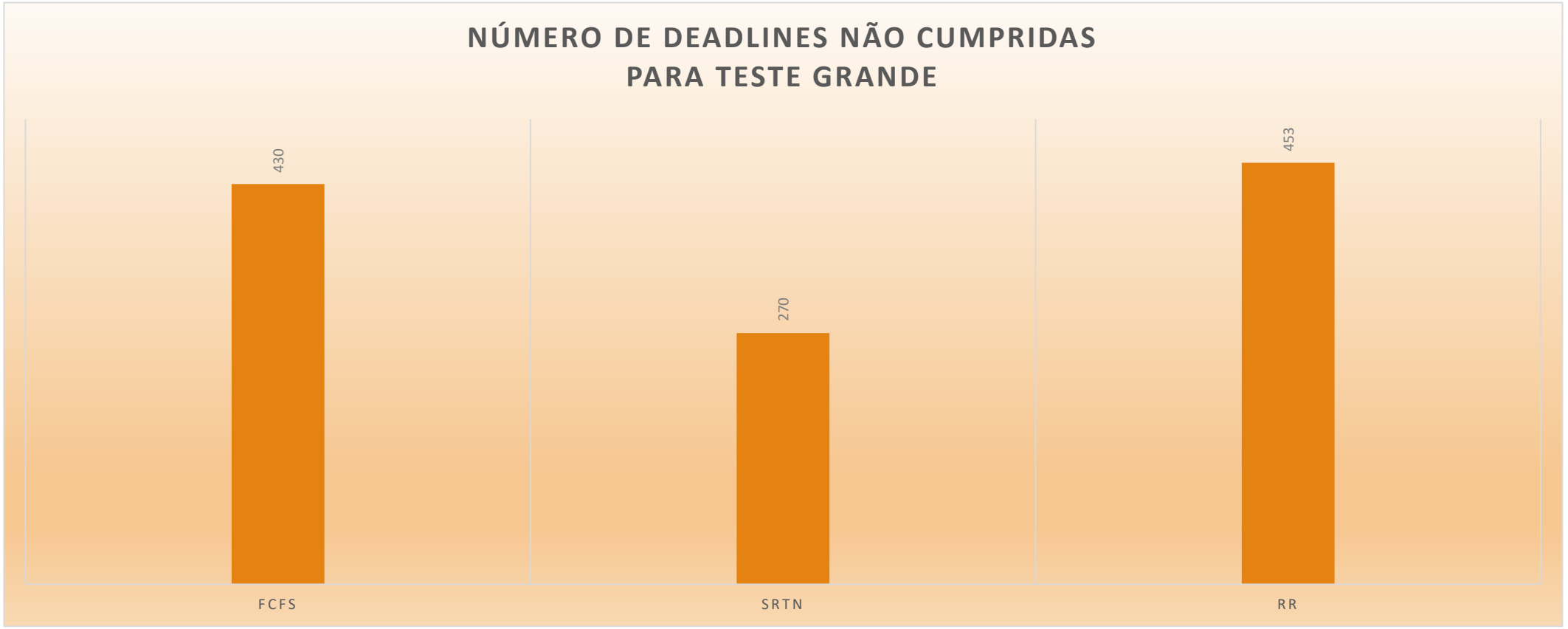
Gráficos – Máquina B



Gráficos – Máquina B

Gráficos – Máquina B

Gráficos – Máquina B



Gráficos – Máquina B

Resultados e Conclusão

- O número de deadlines não cumpridas e a quantidade de mudanças de contexto não se alteraram no decorrer dos 30 testes realizados para o mesmo arquivo de trace, independente da máquina na qual foi executado o teste. Isso ocorreu pois o programa cria apenas uma *thread* por vez, supondo que há apenas uma CPU na máquina, assim o algoritmo de escalonamento é determinístico.

- Como não houve diferença entre os resultados, o intervalo de confiança não existe.

Resultados e Conclusão

- Resultado esperado em testes aleatorizados:

	FCFS	SRTN	RR
Nº deadlines não cumpridas	INTERMEDIÁRIO	MENOR	MAIOR
Nº mudanças de contexto	MENOR	INTERMEDIÁRIO	MAIOR

  - A seguir é explicado o motivo a respeito dos valores esperados em relação ao número de deadlines não cumpridas:
    - FCFS – dependendo dos *dts* dos primeiros processos, o número de deadlines não cumpridos pode variar. Porém, em um caso aleatorizado, é esperado que esse valor esteja entre os dos demais escalonadores;
    - SRTN – a ordenação da fila faz com que os processos tenham tendência a finalizar antes de suas deadlines;
    - RR – devido a preempção, processos que possam ser relativamente curtos são colocados no final da fila.

Resultados e Conclusão

- A seguir é explicado o motivo a respeito dos valores esperados em relação ao número de mudanças de contexto:
  - FCFS – não há preempção;
  - SRTN – há preempção apenas quando há a chegada de novos processos;
  - RR – sempre há preempção.

- Com base nos valores de deadlines não cumpridas e no número de mudanças de contexto, podemos observar que eles coincidem com os valores esperados mostrados anteriormente.

- O escalonador SRTN tem melhor desempenho nos casos testados no EP no sentido de cumprir deadlines. Enquanto o escalonador RR não é recomendado para tal propósito.