

# EP1 - *newsh* e simulador de processos

MAC0422 - Sistemas Operacionais  
Beatriz Viana Costa - 13673214





# SUMÁRIO

01 ●

***newsh***

02 ●

**Simulador de  
processos**

03 ●

***Tracefiles***

04 ●

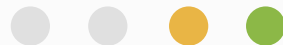
**Escalonamento  
por prioridade**

05 ●

**Ambiente de  
execução dos  
testes**

06 ●

**Resultados**

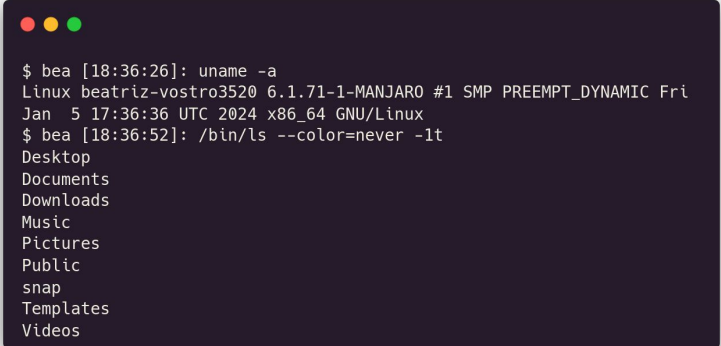


# newsh

- Implementação de um novo *shell* que permite a invocação externa de executáveis e possui também três comandos internos (`rm <arquivo>`, `cd <diretório>` e `uname -a`).
- O *shell* pode ser encerrado com o comando Ctrl+C.
- Segue o formato padrão do prompt do *shell* implementado e um exemplo de execução.




A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The prompt is displayed as `<username> [HH:MM:SS]:`.



A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It shows the execution of two commands: `uname -a` and `/bin/ls --color=never -lt`. The output of `uname -a` is: `Linux beatriz-vostro3520 6.1.71-1-MANJARO #1 SMP PREEMPT_DYNAMIC Fri Jan 5 17:36:36 UTC 2024 x86_64 GNU/Linux`. The output of `/bin/ls --color=never -lt` is a list of directories: `Desktop`, `Documents`, `Downloads`, `Music`, `Pictures`, `Public`, `snap`, `Templates`, and `Videos`.



# Simulador de processos

- Fez utilização da biblioteca `pthread.h` para a paralelização de tarefas, e também da `sched.h` para a utilização de apenas uma CPU.
  - Todos os algoritmos de escalonadores utilizam como estrutura de dados a lista ligada circular.
  - A estratégia adotada foi a utilização de duas *threads* principais + 1 *thread* para cada processo:
    - I. A *thread* principal do escalonador fica encarregada de simular os processos;
    - II. A outra fica encarregada de criar as *threads* para cada processo, que por sua vez ficam encarregadas de colocar os processos na fila.
- 

# Tracefiles

- Cada **arquivo de trace** foi pensado a fim de ressaltar as características de cada escalonador.
- **tracefile1**: Possui poucos processos (8). Contudo, os primeiros a chegar no simulador possuem duração mais longa, e os demais duração mais curta.
- **tracefile2**: Possui uma quantidade média de processos (12), todos com a mesma duração.
- **tracefile3**: Possui mais processos do que as demais (17), todos com durações pequenas e deadlines próximas.
- Os algoritmos dos escalonadores **não se mostraram determinísticos** para as entradas testadas, pois utilizam threads para inserir os processos na fila do escalonador, o que pode gerar resultados diferentes para cada execução quando mais de um processo se inicia no mesmo instante **t0**.

# Escalonamento por prioridade

- Ao serem inseridos, os processos têm uma prioridade atribuída.
- Ao fim de cada **quantum**, todos os processos têm suas prioridades atualizadas e a fila é reordenada.
- O **quantum** utilizado no algoritmo é uma **constante**.
- A prioridade **p** é um inteiro que varia de -19 (mais importante) e 20 (menos importante).
- A relação utilizada para prioridade é o quão “atrasado” ou “adiantado” um processo está em relação à sua **deadline**.

```
prior = (deadline - (time(NULL)-init_time));  
  
if(prior > 19)  
    prior = 19;  
else if(prior < -20)  
    prior = -20;
```

# Ambiente de execução dos testes

## Máquina A

- Sistema operacional:  
Manjaro Linux 23.0.4  
Uranos
- Modelo: 12th Gen  
Intel(R) Core(TM)  
i7-1255U
- CPU(S): 12

## Máquina B

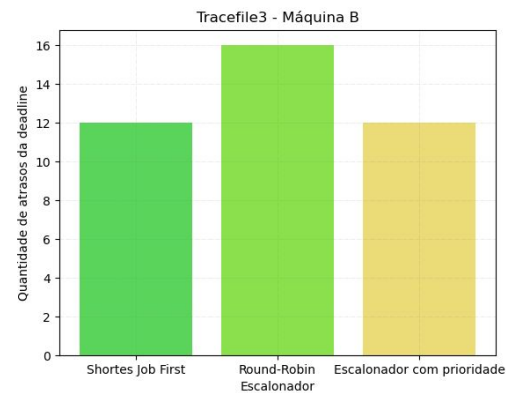
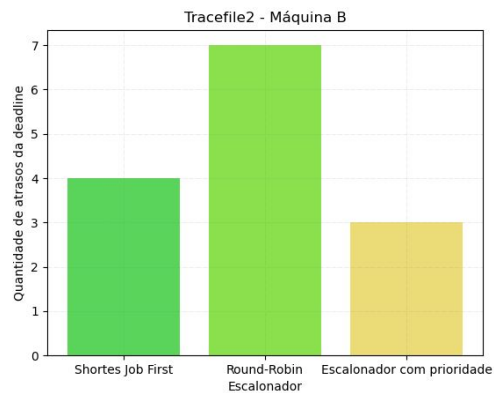
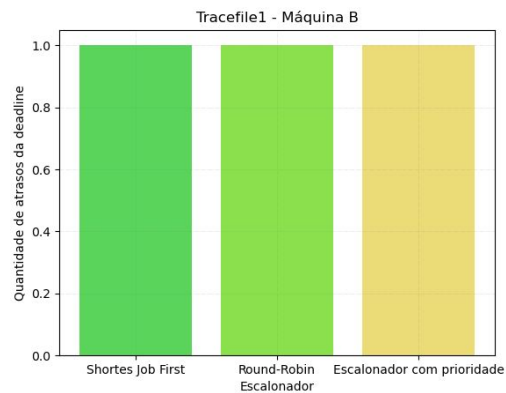
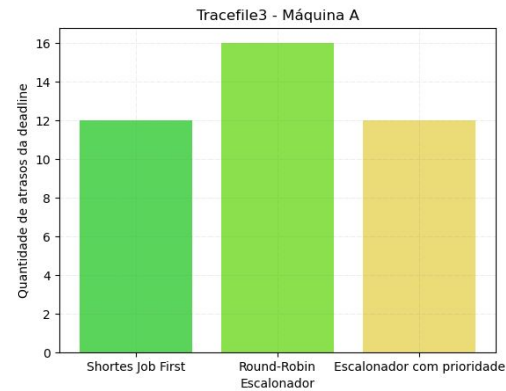
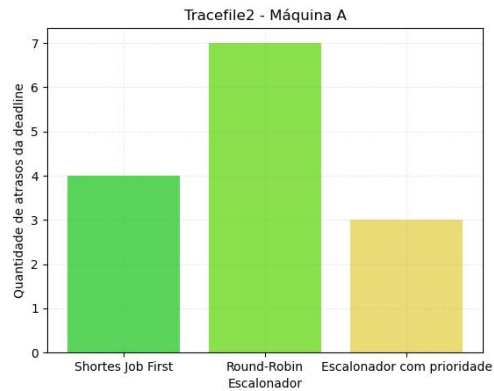
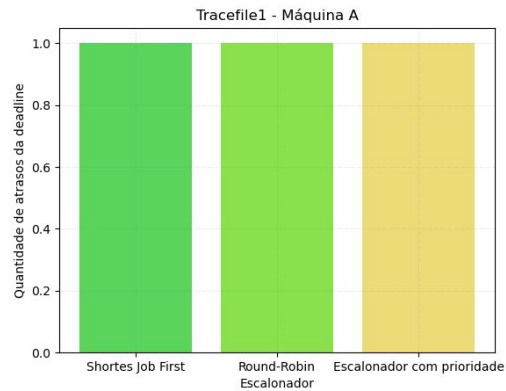
- Sistema operacional:  
Debian GNU/Linux 10  
(buster)
- Modelo: Intel(R)  
Xeon(R) CPU E5-2670  
0 @ 2.60GHz
- CPU(S): 32

## Testes

- Foram realizadas diversas medições e tomados a média e o intervalo de confiança, com nível de confiança de 95%.
- Para os escalonadores *Round-Robin* e com prioridade foi utilizado `quantum=3`.

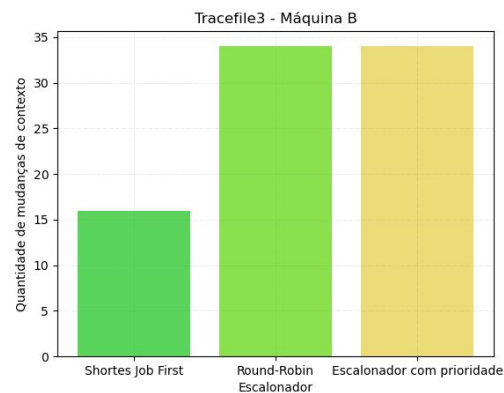
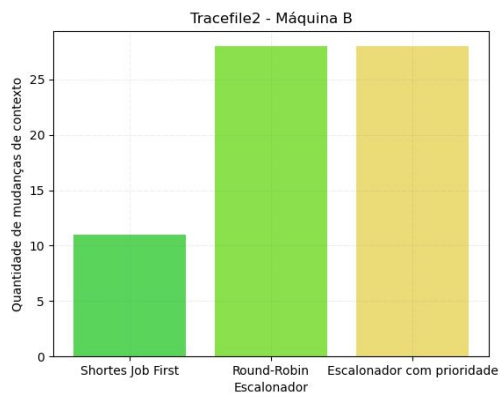
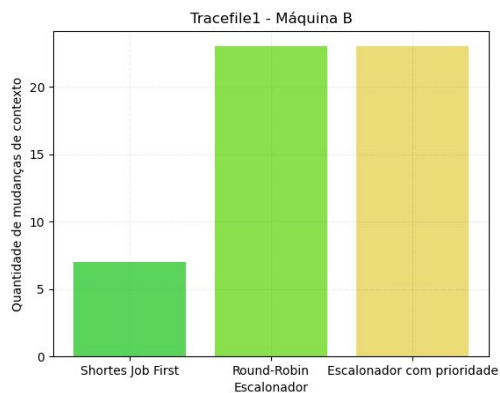
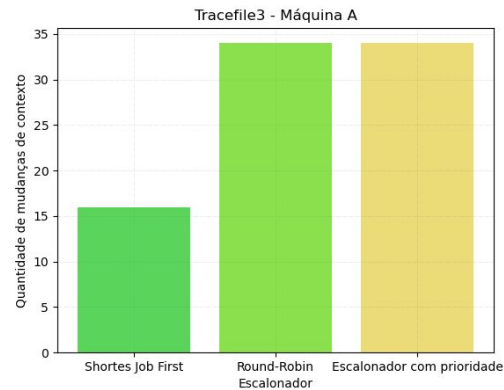
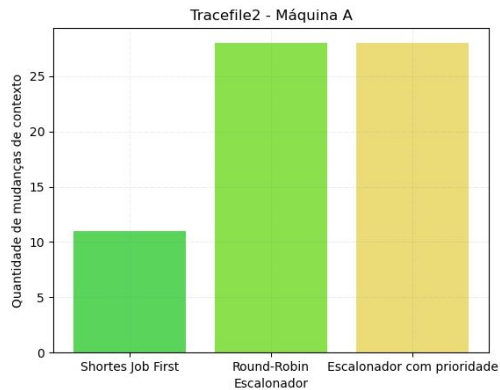
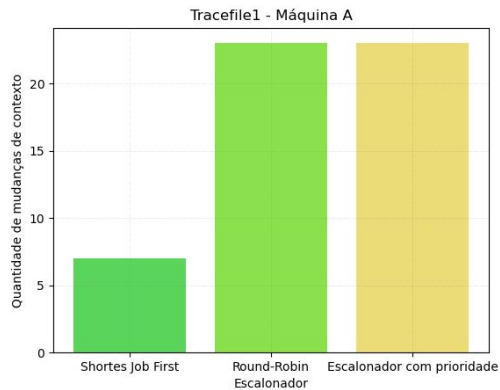


# Resultados: Atraso da *deadline*

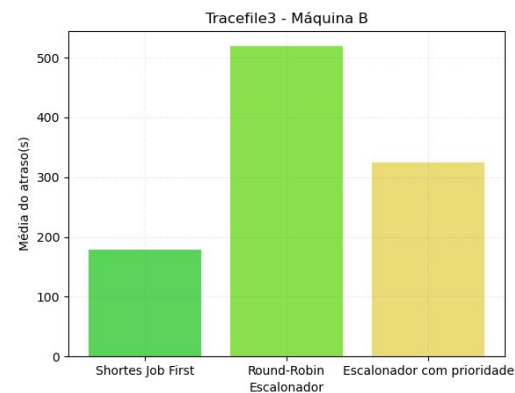
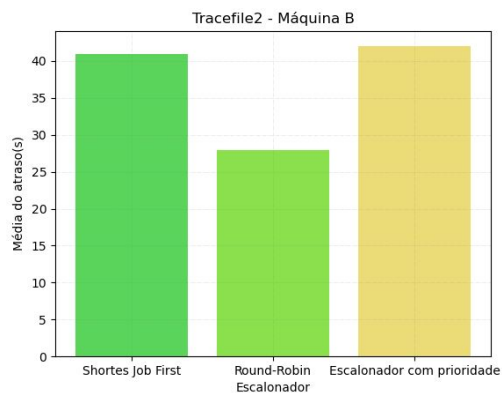
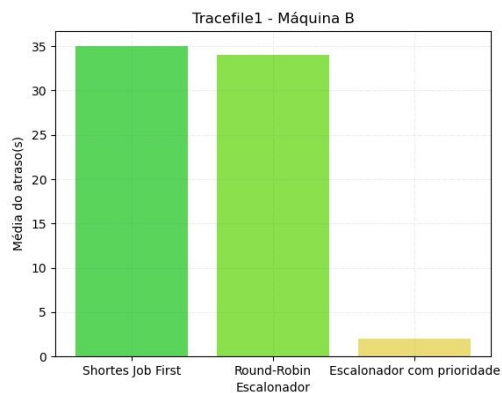
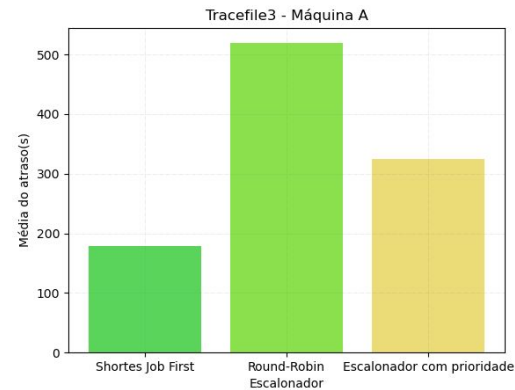
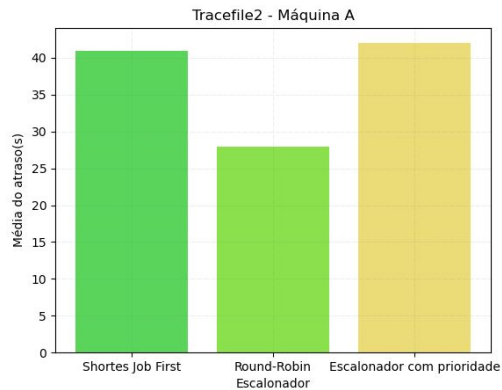
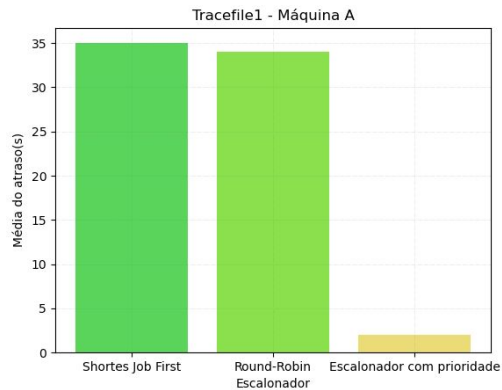




# Resultados: mudanças de contexto



# Resultados: média de atraso de processos



# Análise da *deadline* e tempo de atraso

- **Shortest Job First:** Funciona melhor para processos de duração curta e em menor quantidade;
- **Round-robin:** Apesar de ser mais justo do que o **SJF** ao não permitir o monopólio da CPU, apresenta como falha não levar em consideração a *deadline* de cada processo, o que causou um baixo desempenho nos testes.
- **Escalonador com prioridade:** Foi o que obteve melhor desempenho nos testes, de modo que apesar de ter uma quantidade considerável de processos atrasados, não fez com que um processo ficasse muito atrasado, o que ocorreu nos demais algoritmos.
- Os resultados estão dentro do esperado, **com exceção do Round-Robin**, que obteve desempenho bem mais abaixo do que esperávamos inicialmente.

# Análise das mudanças de contexto

→ Novamente, como esperado o **SJF** foi o que obteve menor quantidade de mudanças de contexto.

E, apesar de o **Round-Robin** e o **escalonador com prioridades** apresentarem um melhor desempenho em relação ao cumprimento das deadlines, apresentaram também uma alta quantidade de mudanças de contexto, mesmo no caso em que tínhamos apenas 8 processos. O que pode ser muito custoso para um sistema operacional.

→ Foi possível perceber assim, que uma melhor “adaptação” do algoritmo à quantidade de processos e suas respectivas deadlines, levou a uma maior computação (das novas prioridades que eram atribuídas no fim de cada **quantum**) e também à uma maior quantidade de mudanças de contexto.