

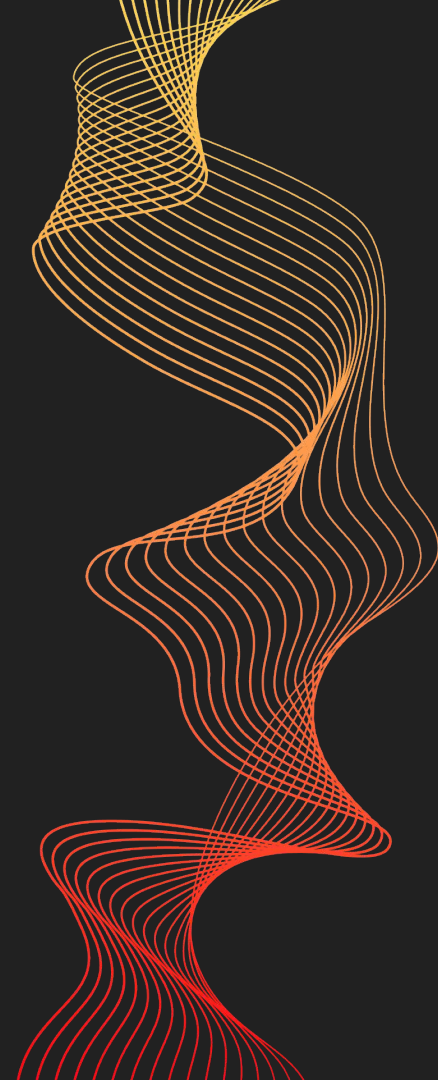


# Sistemas Operacionais - EP1

Aluna: Laís Nuto Rossman

NºUSP: 12547274

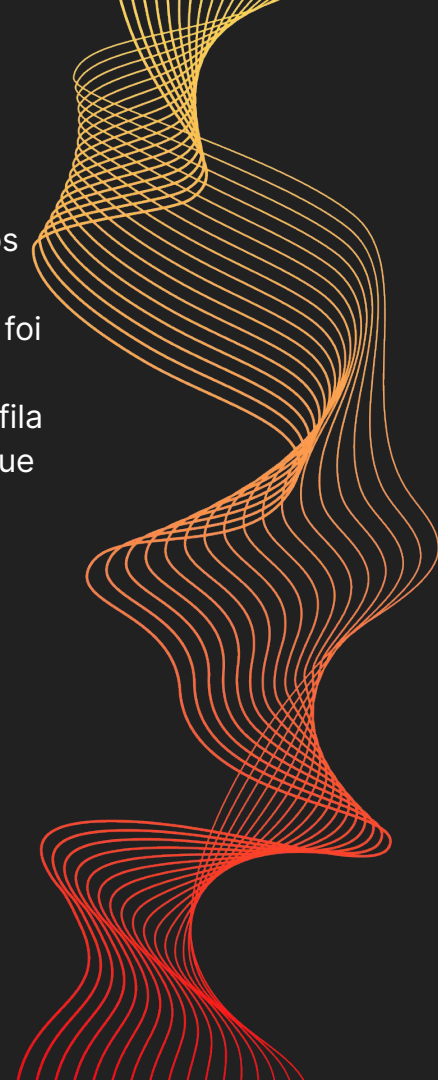
Descrição: Simulação de Processos com diferentes tipos de escalonadores





# Descrição geral do sistema

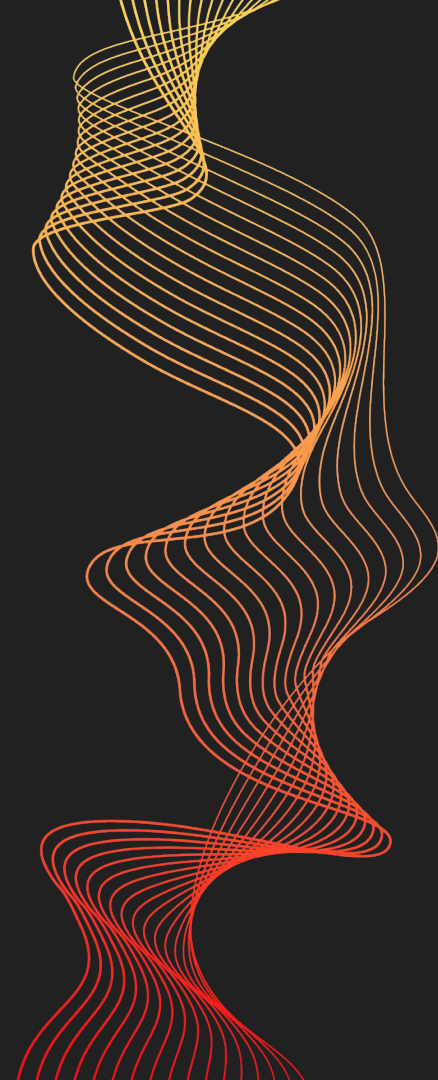
- A ideia principal do código é poder escalonar os diferentes processos fornecidos pelo usuário de acordo com o escalonador que ele escolheu
- Dependendo do escalonador fornecido, usei diferentes estruturas de dados: sjf foi fila de prioridades, round robin e escalonador foi fila circular
- Num primeiro instante, ordenamos os processos pelo seu tempo de entrada na fila ( $t_0$ ) e criamos uma thread para poder adicionar eles na fila no momento certo que é em cada respectivo  $t_0$
- Enquanto a thread de adicionar na fila está rodando, na função principal são chamadas as funções relacionadas a cada respectivo escalonador para que comesse a execução de processos usando escalonamento
- Nas funções dos escalonadores, enquanto ainda existir processos para serem executados, o processo que irá ser executado é retirado da fila e uma thread é criada para executá-lo conforme cada algoritmo
- Por fim, depois de executar todos e encerrar todas as threads, o programa se encerra e imprime as informações necessárias
- Como a thread de executar e de adicionar na fila mexiam com a mesma variável global, foram usados semáforos para que essas operações fossem thread safe





# Shortest Job First

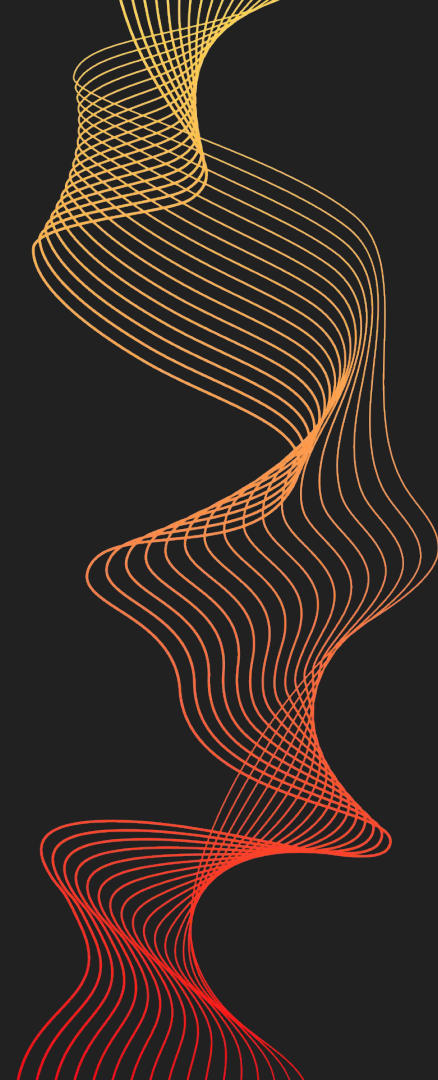
- A implementação desse escalonador foi feita usando uma fila de prioridade usando o dt como prioridade. Assim, toda vez que um processo for removido da fila para ser executado, ele é o que tem a menor duração
- A função que é responsável por esse escalonador basicamente remove um processo da fila de prioridade, se tiver, e cria uma thread para executá-lo por dt segundos. Isso é feito até que a fila esteja vazia e todos os processos sejam executados





# Round Robin

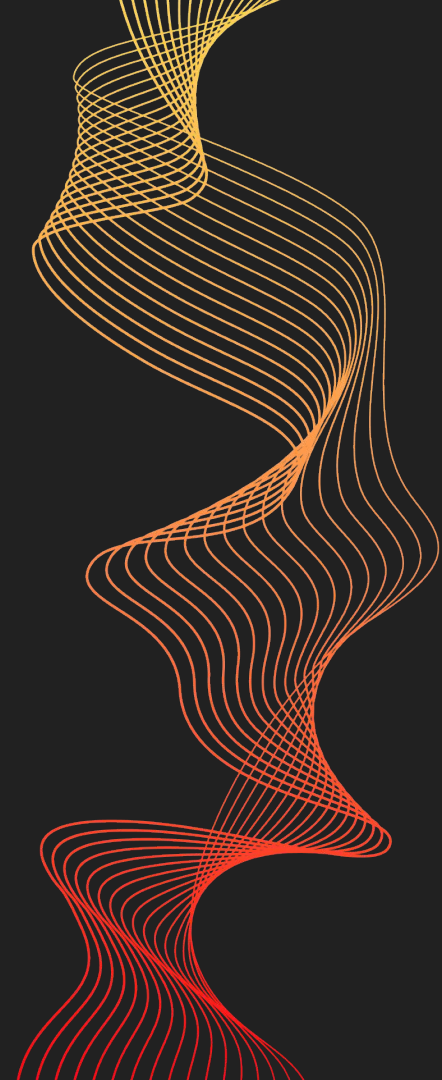
- O Round Robin foi implementado usando uma fila circular, pois essa estrutura de dados é adequada para representar a rotatividade do algoritmo
- Além disso, esse algoritmo tem o quantum fixo de 1 segundo para todos os processos
- A função que é responsável por esse escalonador basicamente remove um processo da fila circular, se tiver, e cria uma thread para executá-lo pelo tempo do quantum ou até o processo ser finalizado. Se depois do tempo de execução dado pelo quantum o processo não tiver acabado, ele é reinserido na fila circular. Isso é feito até que a fila esteja vazia e todos os processos sejam executados





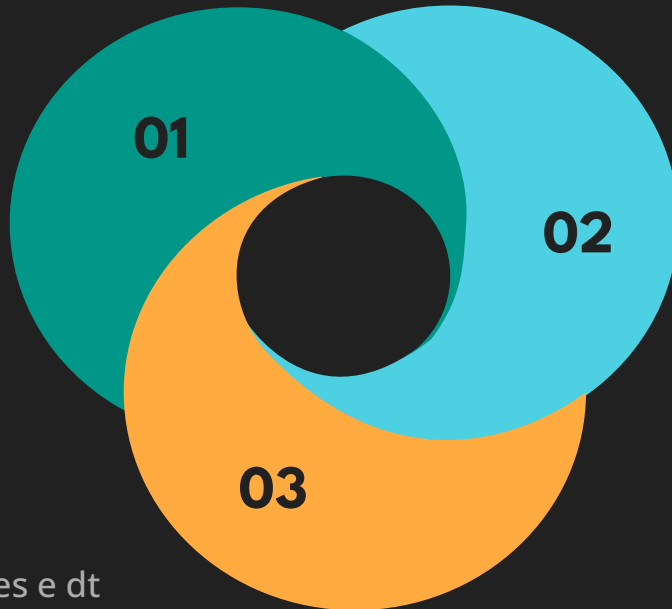
# Escalonamento com prioridade

- O Escalonador com prioridade foi implementado usando uma fila circular, pois, assim como no Round Robin essa estrutura de dados é adequada para representar a rotatividade do algoritmo
- Além disso, esse algoritmo tem um quantum que é calculado toda vez antes do processo ser executado. O cálculo é feito para dar mais quantums para processos com deadlines mais apertadas (e, portanto, maior prioridade). A função para calcular quantum retorna um número entre 1 e 10 segundos, e usa uma fator urgência que é a  $(\text{deadline} - \text{tempo atual}) / (\text{deadline} - t_0)$ , quanto maior for esse fator, maior será o quantum calculado.
- A função que é responsável por esse escalonador basicamente remove um processo da fila circular, se tiver, e cria uma thread para executá-lo pelo tempo do quantum ou até o processo ser finalizado. Se depois do tempo de execução dado pelo quantum o processo não tiver acabado, ele é reinserido na fila circular. Isso é feito até que a fila esteja vazia e todos os processos sejam executados



# Testes dos algoritmos de escalonadores

10 processos em que, num geral, os processos mais urgentes (com deadline mais apertada) tinham menor dt




15 processos em que as distribuições de dt e deadline foram um pouco mais uniformes, com exceção do primeiro processos mais demorado (com maior dt)

20 processos em as deadlines e dt são mais aleatórios, com exceção do primeiro processo mais demorado

# Informações dos ambientes de teste


```
lalsnuto@lalsnuto-550XDA
-----
OS: Ubuntu 20.04.6 LTS x86_64
Host: 550XDA P17CFB
Kernel: 5.15.0-87-generic
Uptime: 20 hours, 5 mins
Packages: 2082 (dpkg), 11 (snap)
Shell: bash 5.0.17
Resolution: 1920x1080
DE: GNOME
WM: Mutter
WM Theme: Adwaita
Theme: Yaru [GTK2/3]
Icons: Yaru [GTK2/3]
Terminal: gnome-terminal
CPU: 11th Gen Intel i5-1135G7 (8) @
GPU: Intel Device 9a49
Memory: 11082MiB / 15712MiB
```



Máquina A

```
neofetch

duvanel@dellvanel
-----
OS: Ubuntu 22.04.4 LTS x86_64
Host: Dell G15 5520
Kernel: 6.5.0-26-generic
Uptime: 6 mins
Packages: 2170 (dpkg), 16 (snap)
Shell: zsh 5.8.1
Resolution: 1920x1080, 1920x1080
DE: GNOME 42.9
WM: Mutter
WM Theme: Material-DeepOcean-BL
Theme: Material-DeepOcean-BL [GTK2/3]
Icons: Papirus-Dark [GTK2/3]
Terminal: gnome-terminal
CPU: 12th Gen Intel i5-12500H (16) @
GPU: Intel Alder Lake-P
GPU: NVIDIA GeForce RTX 3050 Mobile
Memory: 2741MiB / 15668MiB
```

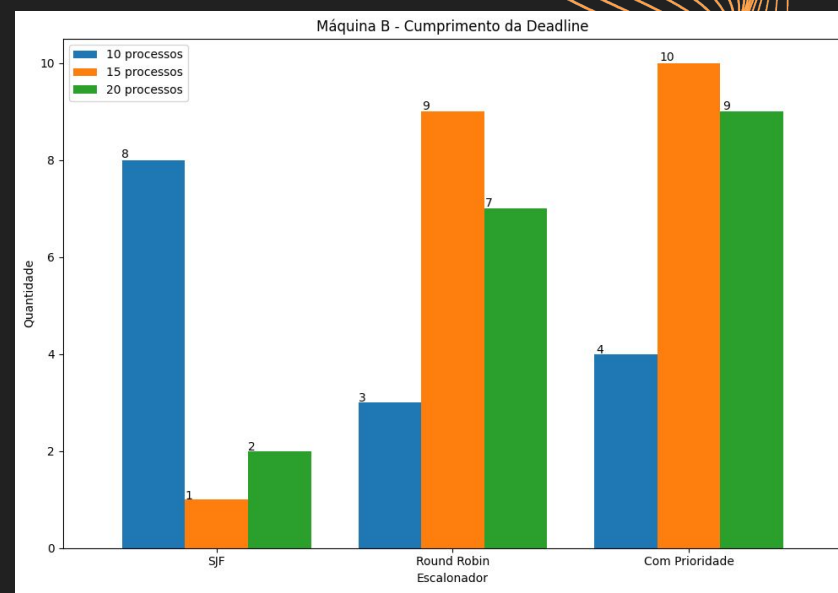
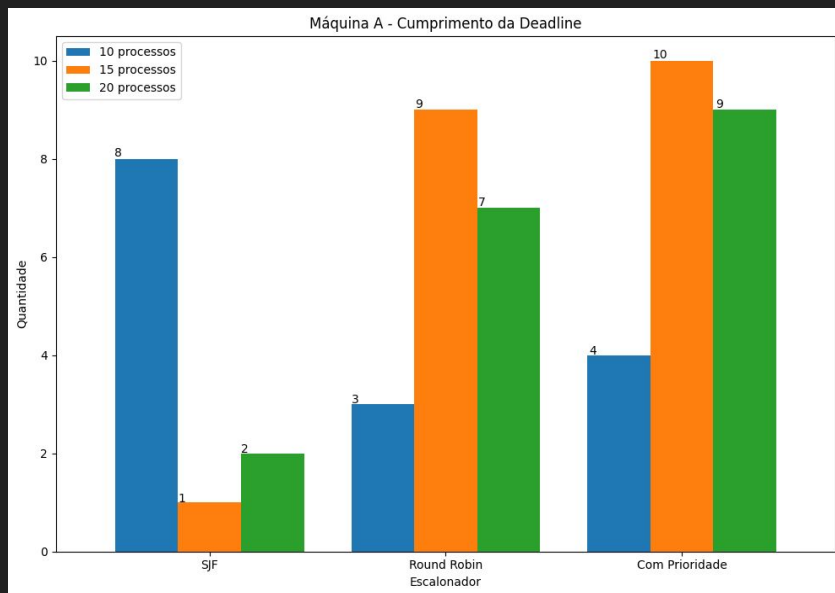


Máquina B



# Análise dos resultados

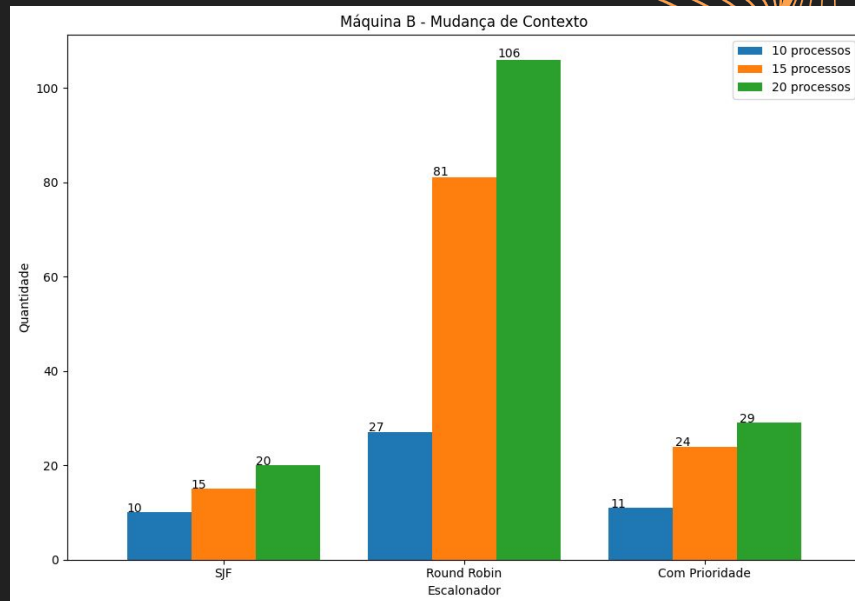
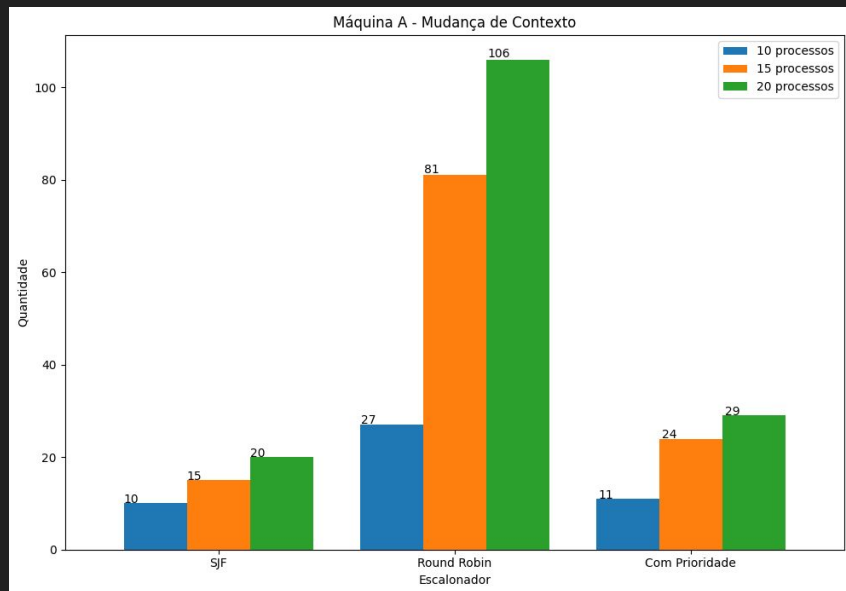
- Nos gráficos é possível ver que o desempenho de cada algoritmo em cada teste varia bastante, mas é o mesmo em ambas as máquinas.





# ▶ Análise dos resultados

- É possível verificar que as mudanças de contexto no SJF correspondem exatamente ao número de processos. Além disso como no Round Robin o quantum é fixo 1 seg e no com prioridade é no mínimo 1 segundo (mas varia de acordo com a proximidade da deadline), é esperado assim como foi observado que o round robin tenha um numero de mudanças de contexto bem maior



# Análise dos testes

- Os arquivos de tinham entradas com o objetivo de evidenciar vantagens e desvantagens de cada algoritmo
- Teste 1: Esse teste tinham 10 processos em que os mais urgentes tinham menor dt. Assim como esperado, o melhor desempenho tanto no cumprimento das deadlines assim na mudança de contextos foi do Short Job First, mostrando sua eficácia em testes onde há uma uam relação entre a urgência dos processos e suas durações. Em contraste, os algoritmos de Round-Robin e de Escalonamento com Prioridade não se saíram tão bem, já que ambos têm a característica de alternância e não necessariamente prioriza a execução dos processos mais urgentes imediatamente.
- Teste 2: Esse teste tinha 15 processos mais uniformes com exceção do primeiro que tinha um maior dt, o que já explica o desempenho ruim do Shortest Job First, já sua preferência por iniciar e concluir o primeiro processo da fila (que era o mais longo) atrasou a execução dos demais. Por outro lado, o Round-Robin, que distribui igualmente o quantum), e o Escalonamento com Prioridade, que ajusta o quantum com base na proximidade das deadlines, demonstraram melhor desempenho, lidando mais eficientemente com essa distribuição mais uniforme de processos.
- Teste 3: Esse teste tinha 20 processos mais aleatorizados com exceção do primeiro que tinha um maior dt, que desfavorece novamente o SJF pela mesma razão do teste 2. O Escalonamento com Prioridade destacou-se por sua habilidade em adaptar o quantum de cada processo baseando-se em sua urgência, resultando no melhor cumprimento das deadlines. O algoritmo conseguiu balancear bem as necessidades de execução dos processos de maneira eficaz, priorizando aqueles com maior urgência.
- Nos 3 testes o escalonamento com prioridade foi melhor que o Round Robin, já que o quantum do round robin é fixo 1 segundo e o quantum do escalonamento com prioridade é no mínimo 1 segundo. Ou seja, com prioridade poderia desempenhar igual o round robin ou melhor já que leva em consideração a urgência dos processos para calcular o quantum.

# Conclusão

- Os testes feitos mostraram uma visão geral dos diferentes comportamentos e desempenho dos três algoritmos de escalonamento sob diferentes condições. Vimos que o SJF se mostrou eficaz em cenários onde a duração dos processos é inversamente proporcional à sua urgência, porém em situações onde um processo longo precede outros mais curtos e urgentes não tinha um bom desempenho. O Round-Robin, por sua característica de distribuição igual de quantum, funcionou bem com cargas de trabalho uniformes e até aleatórias, mas sua eficiência pode ser comprometida em cenários com grandes variações na urgência dos processos. O Escalonamento com Prioridade, adaptando os quantums com base na proximidade das deadlines, se mostrou versátil e eficiente, especialmente em cenários mais complexos com uma variedade de urgências e durações de processos.
- Além disso, os resultados foram os mesmos na máquina A e na máquina B, mostrando que, empiricamente, podemos concluir que o código é determinístico para as entradas testadas. Neste contexto, a razão para a consistência dos resultados pode ser associada ao fato de que cada algoritmo segue um conjunto específico de regras lógicas para tomar decisões de escalonamento. Apesar das diferenças em capacidade de processamento e configuração, os algoritmos aplicam consistentemente suas regras aos processos na fila, resultando em um comportamento previsível em ambas as máquinas.
- Por fim os resultados obtidos confirmam a importância de escolher o algoritmo de escalonamento apropriado com base nas características específicas da carga de trabalho.



Obrigado pelo seu tempo e atenção 😊