Trabalho Prático: Implementação e Análise de Algoritmos de Escalonamento

Mateus S. Ribeiro[1], Sergio A. Cezar[1]

[1]Departamento de Informática – Universidade Estadual de Maringá (UEM)

ra128459@uem.br, ra134680@uem.br

Resumo. *Este trabalho aborda a implementação e análise de algoritmos de escalonamento de tarefas em um simulador multiprocesso. O sistema, composto por processos para Clock, Emissor de Tarefas e Escalonador que se comunicam via sockets, implementa sete políticas distintas. O objetivo principal é analisar o desempenho de cada algoritmo medindo métricas de tempo de turnaround e de espera.*

1. Objetivo e Introdução

Aplicar o conhecimento teórico obtido na disciplina de Sistemas Operacionais (12035) para implementar e analisar diferentes algoritmos de escalonamento de tarefas, analisando na prática. Além dos conceitos de algoritmos de escalonamento de tarefas, serão também implementados conceitos relacionados à comunicação entre processos, mais especificamente, a utilização de sockets.

Este projeto implementa e analisa sete algoritmos de escalonamento diferentes, baseados na bibliografia de Maziero (2019), incluindo FCFS, Round-Robin, SJF, SRTF e três variações de prioridade (cooperativo, preemptivo e dinâmico). O desempenho de cada um é medido pelo cálculo do tempo de retorno (tempo total da tarefa no sistema) e do tempo de espera (quanto tempo a tarefa ficou na fila).

Para realizar essa análise, a simulação foi dividida em três processos que rodam de forma separada e se comunicam exclusivamente por sockets. Esses componentes são um Clock para controlar o tempo, um Emissor de Tarefas para ler as tarefas de um arquivo e colocá-las na fila, e um Escalonador de Tarefas, que executa o algoritmo escolhido e decide qual tarefa usar a CPU a cada instante

2. Decisões de Projeto e Implementação

2.1. Arquitetura Geral da Simulação O sistema foi modelado em três componentes principais (Clock, Emissor, Escalonador), cada um executado em um processo separado usando a biblioteca multiprocessing. Foi utilizado o multiprocessing.Event como mecanismo de sincronização. O processo do Clock só é iniciado após receber um sinal de "pronto" dos processos do Emissor e do Escalonador. O término da simulação é controlado pelo Escalonador, que envia uma mensagem de finalização para os outros processos quando todas as tarefas são concluídas.

A sincronização da inicialização entre os processos foi projetada para evitar condições de corrida durante o estabelecimento das conexões via socket. Utilizamos multiprocessing.Event para garantir que os processos "servidores" (Emissor e Escalonador)

estejam totalmente inicializados e escutando em suas respectivas portas antes que o processo "cliente" tente se conectar.

2.2. Implementação da Fila de Tarefas Prontas

Foram escolhidas estruturas de dados especializadas para cada classe de algoritmo:

- **FCFS e Round-Robin**: Para estes algoritmos, foi utilizada a estrutura collections.deque. A deque, vista em estrutura de dados, é uma fila de duas pontas com operações de inserção e remoção eficientes.
- **SJF, SRTF, e Prioridades (PRIOC, PRIOP, PRIOD)**: Para todos os algoritmos baseados em ordenação, foi utilizado o módulo heapq, que implementa uma fila de prioridade (min-heap). Essa abordagem é drasticamente mais eficiente do que reordenar uma lista a cada decisão de escalonamento. O critério de desempate utilizado foi o tempo de chegada da tarefa.

Além do critério primário (duração, tempo restante ou prioridade, dependendo do algoritmo), a função make_heap_item também utiliza critérios secundários para desempate: primeiro o tempo de chegada (arrival_time), e em seguida o identificador da tarefa (id). Essa hierarquia de desempate garante que a simulação seja totalmente determinística, evitando comportamentos não-reprodutíveis em situações de igualdade entre tarefas.**3.3.**

2.3 Comunicação entre Processos via Sockets A comunicação foi implementada com sockets TCP (socket.AF_INET, socket.SOCK_STREAM). Para garantir a integridade das mensagens, foi desenvolvido um protocolo que anexa um delimitador a cada mensagem enviada. O lado receptor armazena os dados em um buffer e processa apenas mensagens completas. As mensagens principais trocadas incluem:

- CLOCK|{n}||: Enviada pelo Clock.
- NEW_TASK|{dados_tarefa}||: Enviada pelo Emissor.
- ALL_TASKS_SENT||: Sinaliza que todas as tarefas foram emitidas.
- END_SIMULATION||: Sinaliza o fim da simulação.

Foram usados sockets no modo "não-bloqueante" com socket.setblocking(False). Isso impede que um processo pare completamente enquanto espera por uma mensagem. Com a ajuda da função select.select(), o programa podia verificar de uma só vez se havia dados em múltiplas conexões, evitando travamentos.

2.4. Leitura de Entrada e Escrita de Saída

A biblioteca argparse é utilizada para ler os comandos. O processo Emissor lê o arquivo com as tarefas, enquanto o Escalonador, no final da simulação, cria os arquivos de resultado (um texto com as métricas e o Diagrama de Gantt).

2.5. Estruturas Adicionais

- **Task**: Para facilitar a representação das tarefas, estas foram implementadas utilizando dataclass.

2.5. Explicação e implementação dos Algoritmos

- **PRIOp**: A preempção ocorre dinamicamente. A cada ciclo de clock, se uma tarefa recém-chegada ou já na fila tiver prioridade estritamente maior que a tarefa em execução, a tarefa atual é devolvida à fila de prontos (heap) e a nova tarefa de maior prioridade é escalonada.
- **PRIOD** : A implementação utiliza o mecanismo de *aging*. A cada ciclo de clock, a prioridade de todas as tarefas na fila de prontos é aumentada (o valor numérico é decrementado). Quando uma tarefa é escolhida para executar, sua prioridade dinâmica é resetada ao seu valor original estático para evitar a monopolização da CPU.
- **FCFS**: A execução segue a ordem de chegada,. A primeira tarefa que entra na fila de prontas (deque) é a primeira a ser executada. Por ser não-preemptivo, a tarefa roda até o fim sem ser interrompida. As outras precisam esperar a sua vez, independentemente de serem mais curtas ou mais importantes.
- **RR**: Este algoritmo busca ser justo, dando a cada tarefa uma pequena fatia de tempo na CPU, chamada de quantum. A tarefa roda por até 3 unidades de tempo. Se não terminar, ela é interrompida e movida para o final da fila. A CPU então passa para a próxima tarefa, garantindo que nenhum processo espere indefinidamente para começar a rodar.
- **SJF**: Quando a CPU fica livre, o escalonador olha a fila de prontas (heap) e seleciona a tarefa que tem a menor duração total de execução. Por ser não-preemptivo, uma vez que a tarefa começa, ela roda até o fim sem interrupções, mesmo que uma tarefa ainda mais curta chegue na fila. O objetivo é diminuir o tempo de espera geral, finalizando rapidamente os trabalhos mais fáceis.
- **SRTF**: A cada ciclo de clock, o escalonador verifica se alguma tarefa na fila de espera possui um tempo restante menor que o tempo restante da tarefa que está rodando. Se houver, a tarefa atual é interrompida e volta para a fila (heap), e a tarefa que está mais perto de terminar assume a CPU.
- **PRIOC**: Este algoritmo seleciona a tarefa com a maior prioridade (menor valor numérico) da fila de prontos (heap), mas apenas quando a CPU está ociosa. Por ser cooperativo (ou não-preemptivo), uma vez que uma tarefa começa a rodar, ela continua até terminar, mesmo que uma tarefa de prioridade muito maior chegue nesse meio tempo.

3. Análise dos Algoritmos de Escalonamento

3.1. First-Come, First-Served (FCFS)

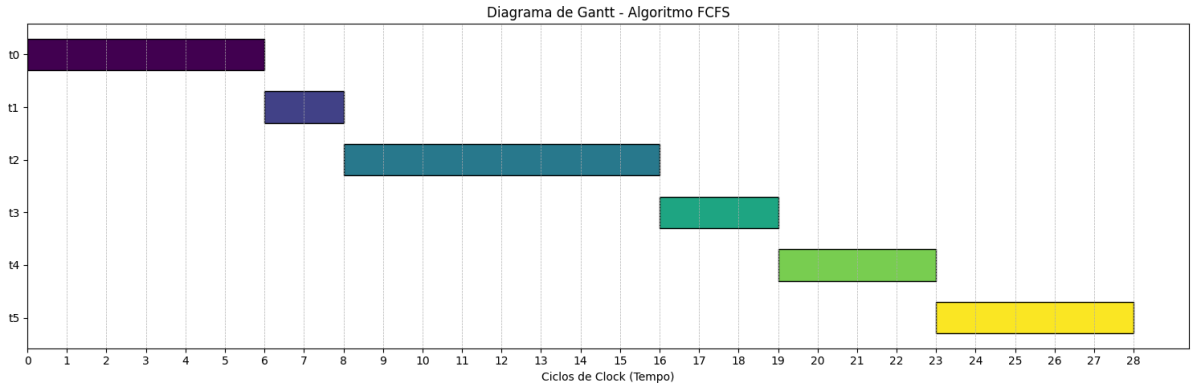

Figura 1. Diagrama de Gantt para a execução do FCFS.

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	6	6	0
t1	1	8	7	4
t2	2	16	14	6
t3	3	19	16	13
t4	5	23	18	14
t5	6	28	22	17
Médias			13,9	9,2

Tabela 1. Métricas de desempenho para o FCFS.

O FCFS executa as tarefas na ordem de chegada. A tarefa t1, que é curta, chega no clock 1 mas precisa esperar a t0, mais longa, terminar. Isso resulta em tempos de espera elevados para tarefas que chegam mais tarde, levando a uma média de espera alta.

3.2. Round-Robin (RR)

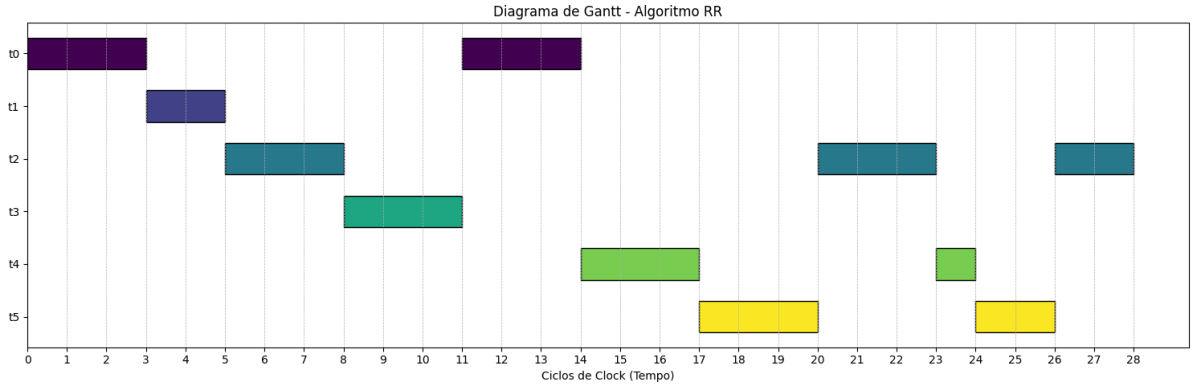

Figura 2. Diagrama de Gantt para a execução do RR (quantum=3).

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	14	14	8
t1	1	5	4	2
t2	2	28	26	18

t3	3	11	8	5
t4	5	24	19	15
t5	6	26	20	15
Médias			15.2	10.5

Tabela 2. Métricas de desempenho para o RR.

O RR utiliza preempção e melhora o tempo de resposta para tarefas curtas, como visto na redução do tempo de espera de t1 (de 5 para 2). No entanto, o custo são as trocas de contexto, que aumentam o tempo de turnaround de tarefas mais longas e resultam no maior tempo médio de espera geral.

3.3. Shortest Job First (SJF)

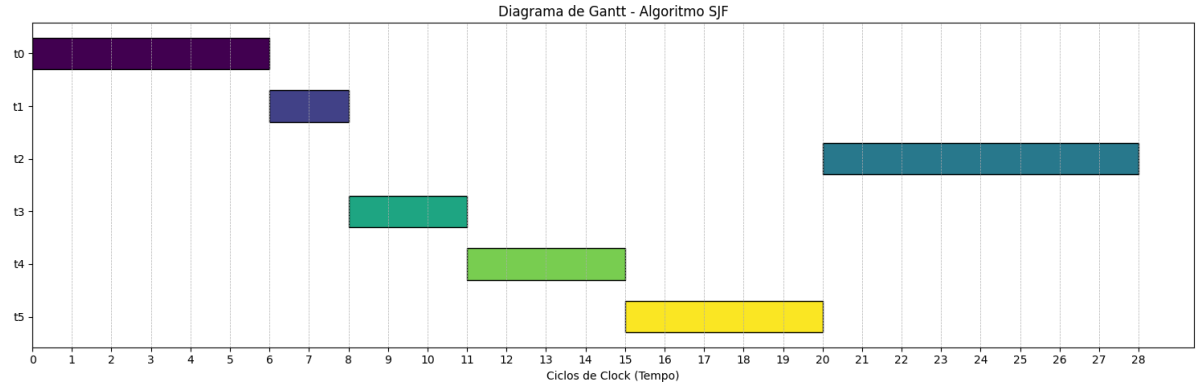

Figura 3. Diagrama de Gantt para a execução do SJF.

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	6	6	0
t1	1	8	7	4
t2	2	28	26	18
t3	3	11	8	5
t4	5	15	10	6
t5	6	20	14	9
Médias			11.9	7.2

Tabela 3. Métricas de desempenho para o SJF.

O SJF é o melhor para minimizar o tempo médio de espera. A implementação com heapq garante que a decisão de qual tarefa mais curta executar seja feita de forma eficiente. O algoritmo escolhe corretamente a sequência de tarefas mais curtas após t0, reduzindo o tempo de espera geral. A desvantagem é o potencial de inanição para tarefas longas, como t2, que é deixada para o final. Isso mostra a principal desvantagem do SJF: a ausência de garantias de justiça em relação à execução de tarefas de longa duração.

3.4. Shortest Remaining Time First (SRTF)

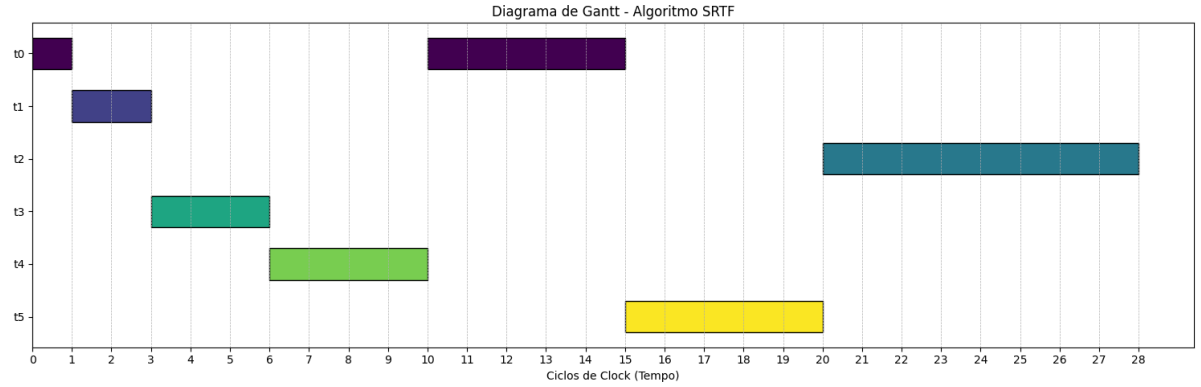

Figura 4. Diagrama de Gantt para a execução do SRTF.

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	15	15	9
t1	1	3	2	0
t2	2	28	26	18
t3	3	6	3	0
t4	5	10	5	1
t5	6	20	14	9
Médias			10.9	6.2

Tabela 4. Métricas de desempenho para o SRTF.

O SRTF alcança o menor tempo médio de espera de todos. O evento chave é a preempção da tarefa t0 pela t1 no clock 1, pois t1 tinha um tempo de execução total menor que o tempo *restante* de t0. Isso zera o tempo de espera para tarefas curtas que chegam, otimizando drasticamente a média geral.

3.5. Escalonamento por Prioridades Cooperativo (PRIOC)

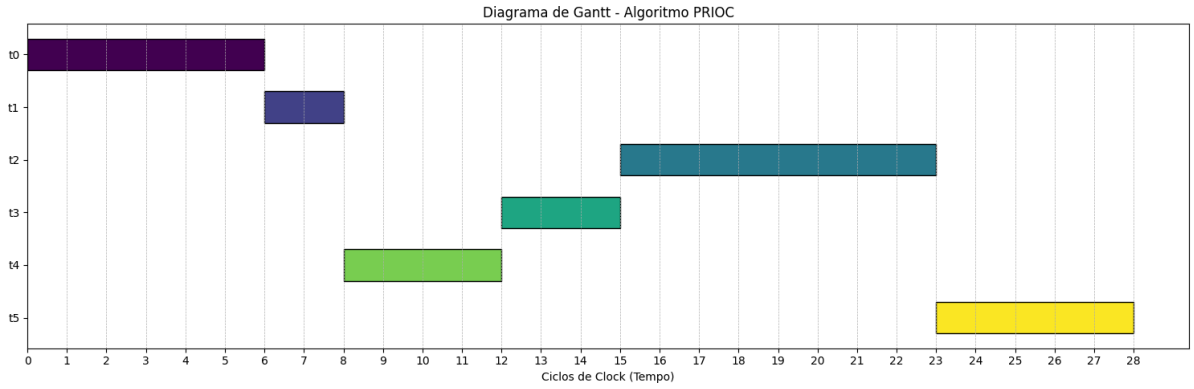

Figura 5. Diagrama de Gantt para a execução do PRIOC.

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	6	6	0
t1	1	8	7	5
t2	2	23	21	13
t3	3	15	12	9
t4	5	12	7	3
t5	6	28	22	17
Médias			12.5	7.9

Tabela 5. Métricas de desempenho para o PRIOC.

No PRIOC, a prioridade só é verificada quando a CPU está livre. Mesmo com a chegada de tarefas de alta prioridade (t1, t4), elas precisam esperar a t0 (de prioridade menor) terminar. A decisão de escalonamento baseada em prioridade só ocorre no clock 6.

4.6. Escalonamento por Prioridades Preemptivo (PRIOp)

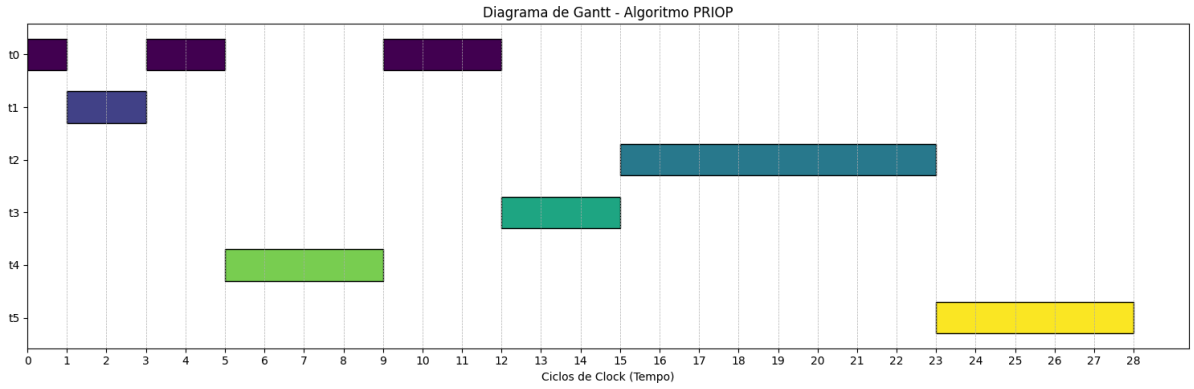

Figura 6. Diagrama de Gantt para a execução do PRIOp.

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	12	12	6
t1	1	3	2	0
t2	2	23	21	13
t3	3	15	12	9
t4	5	9	4	0
t5	6	28	22	17
Médias			12.2	7.5

Tabela 6. Métricas de desempenho para o PRIOp.

Percebe-se que a versão preemptiva é muito mais responsiva a tarefas de alta prioridade. A tarefa t1 (prioridade 1) preempta t0 (prioridade 2) imediatamente ao chegar, zerando seu tempo de espera . O mesmo ocorre com t4. Esse comportamento garante que a CPU esteja sempre com a tarefa mais importante.

3.7. Escalonamento por Prioridades Dinâmicas (PRIOD)

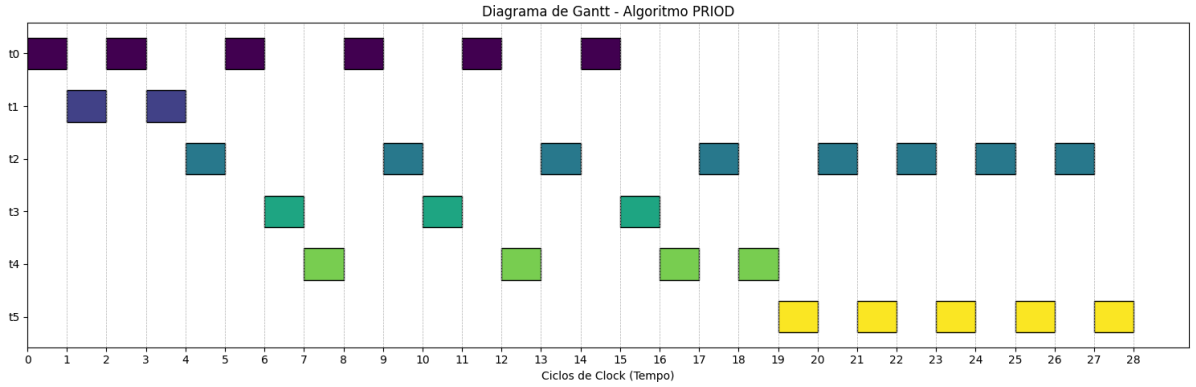

Figura 7. Diagrama de Gantt para a execução do PRIOD.

Tarefa	T. Chegada	T. Finalização	Turnaround	T. Espera
t0	0	17	17	11
t1	1	3	2	0
t2	2	28	26	18

t3	3	14	11	5
t4	5	9	4	0
t5	6	25	19	14
Médias			15.4	10.7

Tabela 7. Métricas de desempenho para o PRIOD.

A política implementada usa *aging* para prevenir inanição, aumentando a prioridade de tarefas que esperam na fila a cada ciclo de clock. Quando uma tarefa é escalonada, sua prioridade é revertida ao valor original . O efeito do *aging* é claro, pois altera a ordem de execução em comparação com o PRIOP, fazendo com que tarefas de menor prioridade estática (t2, t3) e ganhem a CPU mais facilmente.

4. Conclusão

Este projeto aplicou conceitos de sistemas operacionais na criação de um simulador de escalonador. A simulação confirmou a teoria, com o algoritmo SRTF mostrando-se o mais eficiente e o Round-Robin o mais lento em média para os testes executados, destacando a diferença entre tempo de resposta e eficiência geral. O principal desafio foi a sincronização de processos concorrentes. A principal conclusão é que a escolha do algoritmo de escalonamento ideal depende dos objetivos específicos de cada sistema, não havendo uma solução universalmente "melhor".

5. Referências

Maziero, C. A. (2019). *Sistemas Operacionais: Conceitos e Mecanismos*. Edição do Autor, Ponta Grossa, PR, 5ª edição.