Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática

Ciência da Computação

Sistemas Operacionais(12035)

Relatório técnico de desenvolvimento do trabalho

Discente

Nome	**RA**
Eduardo Andrade Manfre	116409

Maringá - PR - 2025

Desenvolvimento do Trabalho

O trabalho consiste em uma simulação de um ambiente de escalonamento de processos, utilizando sockets e threads para reproduzir o comportamento de um sistema operacional multitarefa. O sistema foi dividido em três módulos principais. O clock.py envia ticks periódicos aos demais módulos. O emissor.py envia tarefas com base em um arquivo de entrada, enquanto o escalonador.py recebe e organiza as tarefas segundo o algoritmo de escalonamento informado, utilizei um arquivo chamado comun.py, que contém uma parte do código que é utilizado em mais de um arquivo, então para não repetir código utilizei isso.

A fila de tarefas prontas foi implementada como uma lista global compartilhada no arquivo do escalonador. Essa fila é preenchida conforme as tarefas são recebidas via socket. Os processos são representados por objetos da classe Processo, definidos no arquivo comun.py. O escalonador utiliza duas threads: uma para receber os processos (tarefas) e outra para receber os ticks (clock). A comunicação foi feita por meio de objetos serializados com pickle, que envia e recebe objetos da classe Processo entre emissor, escalonador e clock, por meio de sockets.

Durante o desenvolvimento, foi necessário criar estruturas adicionais como listas auxiliares e dicionários de controle, por exemplo, para armazenar tempo restante de execução ou filas intermediárias, dependendo do algoritmo. Os dados de entrada são lidos no emissor.py, que envia os processos no tempo correto, e os dados de saída são gerados pelo escalonador.py por meio da função salvar_saida, que escreve a sequência de execução, as métricas individuais de cada processo e as médias finais.

As principais dificuldades envolveram a coordenação entre os módulos executados em paralelo, o controle da ordem correta dos eventos via socket e a implementação dos algoritmos preemptivos, que são chatos de gerenciar o controle de tempo restante e alternância entre processos.

Resultados dos Algoritmos de Escalonamento

Algoritmo	ID	Clock de Ingresso na Fila de Prontas	Clock de Finalização	Turnaround Time	Waiting Time
FCFS	t0	0	6	6	0
	t1	6	8	7	5
	t2	8	16	14	6
	t3	16	19	16	13
	t4	19	23	18	14

	t5	23	28	22	17
	Média	**-**	**-**	**13.9**	**9.2**
RR	t0	0	18	18	12
	t1	2	4	3	1
	t2	4	28	26	18
	t3	8	19	16	13
	t4	10	21	16	12
	t5	12	26	20	15
	Média	**-**	**-**	**16.5**	**11.9**
SJF	t0	0	6	6	0
	t1	6	8	7	5
	t2	20	28	26	18
	t3	8	11	8	5
	t4	11	15	10	6
	t5	15	20	14	9
	Média	**-**	**-**	**11.9**	**7.2**
SRTF	t1	5	7	6	4
	t3	7	10	7	4
	t4	10	14	9	5
	t5	14	19	13	8
	t0	19	25	25	19
	t2	25	33	31	23
	Média	**-**	**-**	**15.2**	**10.5**
PRIOC	t0	0	6	6	0
	t1	6	8	7	5

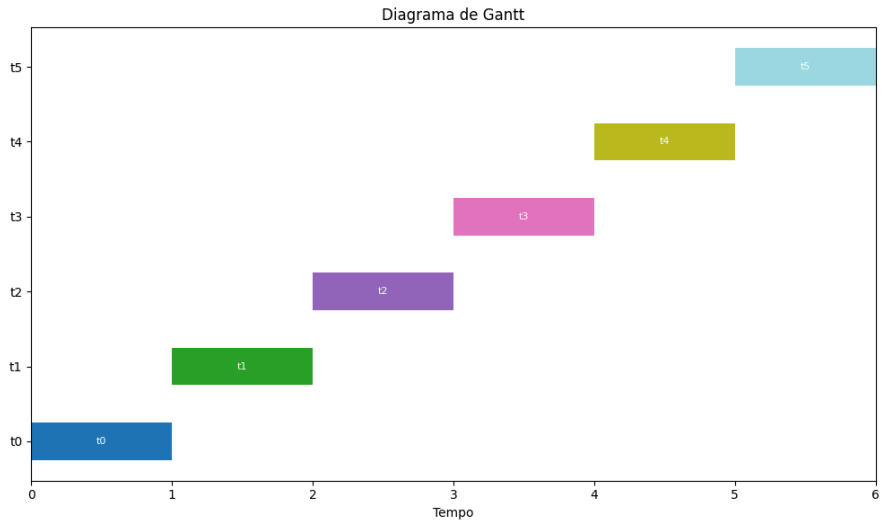
	t2	15	23	21	13
	t3	12	15	12	9
	t4	8	12	7	3
	t5	23	28	22	17
	Média	**-**	**-**	**12.5**	**7.9**
PRIOP	t1	1	3	2	0
	t4	5	9	4	0
	t0	0	14	14	8
	t3	3	15	12	9
	t2	15	23	21	13
	t5	23	28	22	17
	Média	**-**	**-**	**12.5**	**7.9**

A análise dos resultados, com base no arquivo entrada00.txt, mostrou o comportamento esperado. No FCFS, os processos são executados na ordem de chegada. O SJF e o SRTF priorizam os processos mais curtos, com o SRTF sendo preemptivo. O RR distribui o tempo entre os processos de forma mais justa, utilizando um quantum fixo (utilizei valor 2). Já os algoritmos com prioridade (prioc, priop e priod) consideram a prioridade dos processos, com variações entre preemptivos e dinâmicos.

Abaixo encontram-se os Diagramas de Gantt que cada algoritmo de escalonamento resultou.

Diagrama de Gantt

FCFS



RR

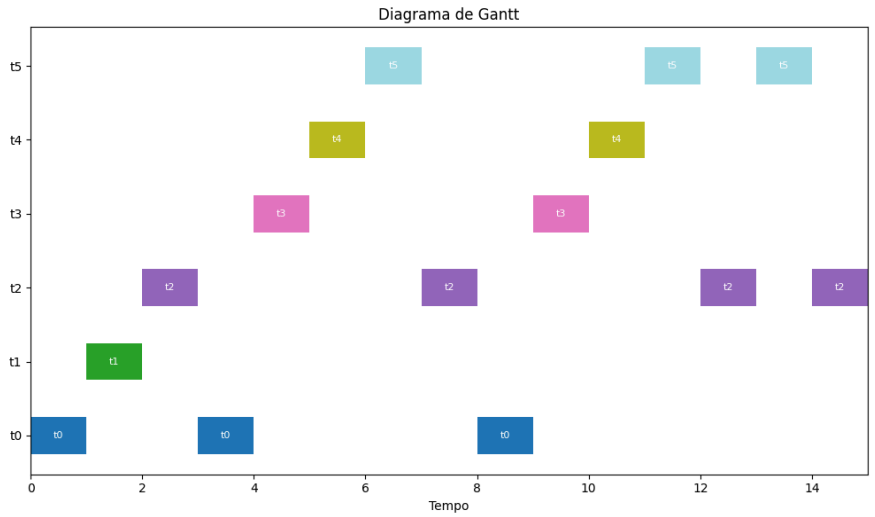


Diagrama de Gantt
t5
t4
t3
t2
t1
t0
0
2
4
6
8
10
12
14
Tempo

SJF

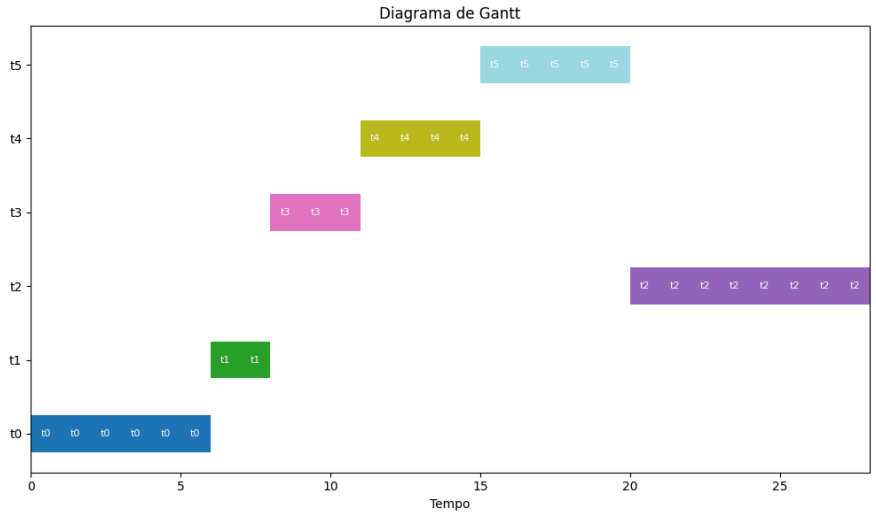


Diagrama de Gantt
t5
t4
t3
t2
t1
t0
0
5
10
15
20
25
Tempo

SRTF

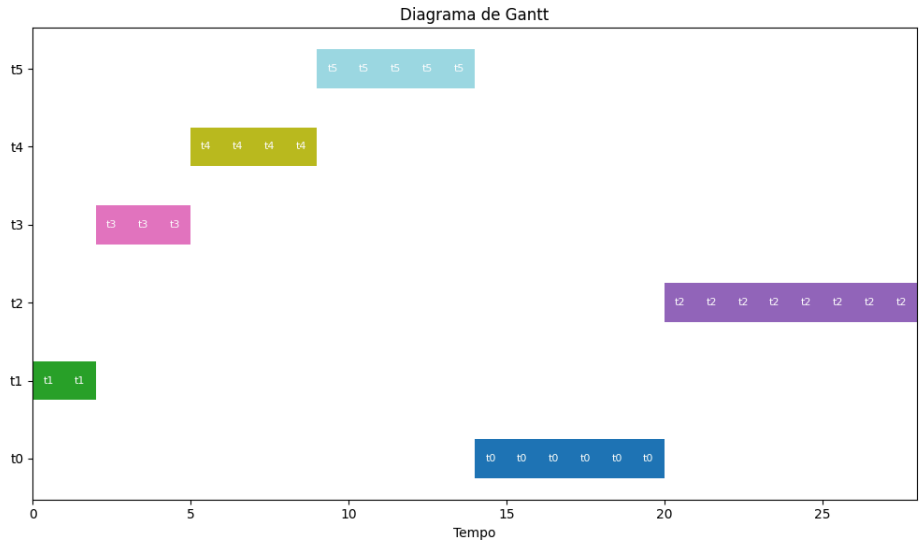
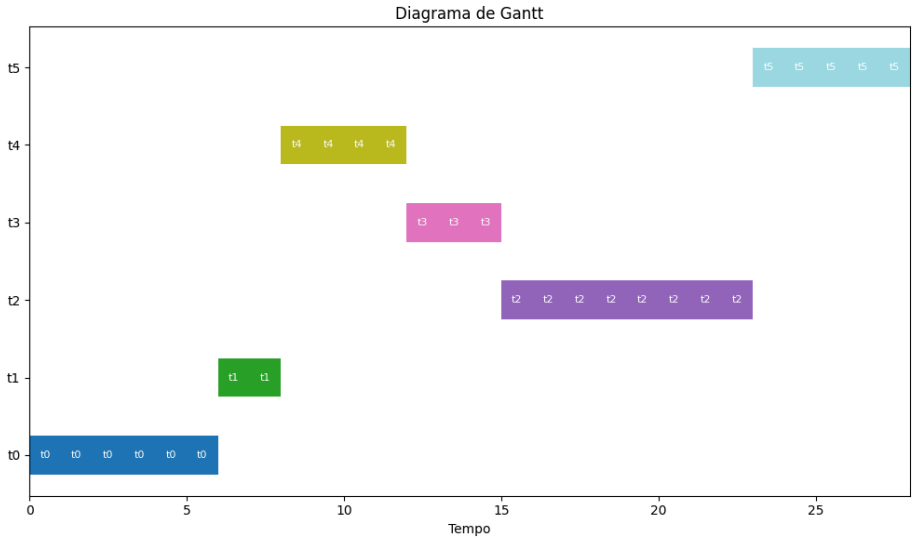
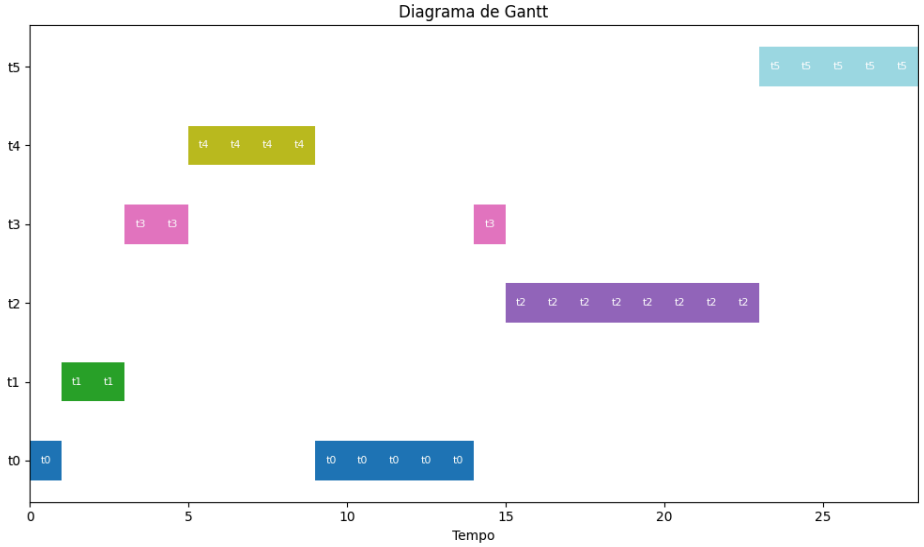


Diagrama de Gantt
t5
t4
t3
t2
t1
t0
0
5
10
15
20
25
Tempo

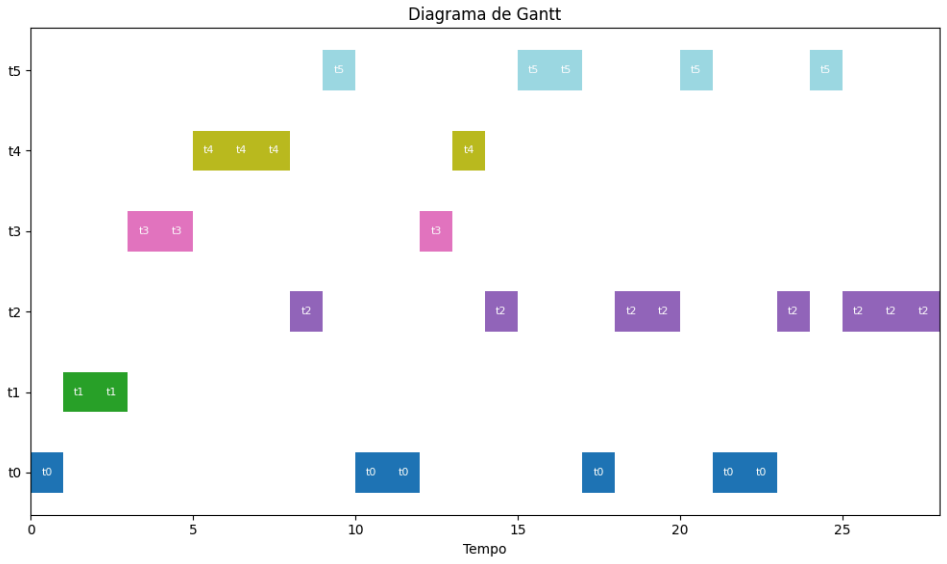
PRIOC



PRIOP



PRIOD



Manual de Compilação e Execução

O sistema de escalonamento de processos desenvolvido foi implementado em Python e utiliza três arquivos principais para sua execução: clock.py, que simula o clock do sistema; escalonador.py, que executa os algoritmos de escalonamento; e emissor.py, que envia os processos definidos em um arquivo de entrada. O sistema se comunica por meio de sockets, exigindo a execução coordenada entre os módulos. O projeto requer Python 3.8 ou superior e a instalação da biblioteca matplotlib, que pode ser feita com o comando pip install matplotlib, outras bibliotecas também são utilizadas mas são padrão do python, logo não é necessário instalação.

A execução do sistema deve ser feita com três terminais abertos simultaneamente.
No primeiro terminal, executa-se o escalonador com o comando: “python escalonador.py <algoritmo>”, onde <algoritmo> pode ser fcfs, rr, sjf, srtf, prioc, priop ou priod.
No segundo terminal, executa-se o emissor com o comando: “python emissor.py entrada00.txt”, onde o arquivo de entrada contém os processos a serem simulados.
No terceiro executa-se “python clock.py”, responsável por iniciar a geração de ticks do clock.

Ao término da execução, o sistema gera o arquivo saida.txt contendo a sequência de execução dos processos, os tempos de início e fim, turnaround e waiting time de cada processo, além das médias. Também é exibido automaticamente um diagrama de Gantt com a representação visual da execução dos processos. Para reiniciar a simulação, é necessário encerrar os três processos e repeti-los na mesma ordem.