Alunos: Felipe Bernardino, Guilherme Battisti Borba

Após realizarmos cerca de 6 testes, 3 com FIFO e 3 com Round-Robin alterando o quantus usando 2,5,10 para análise de resultado. Fizemos o uso das bibliotecas pandas e numpy para criar um código em python que seria capaz de retornar resultados através das tabelas fornecidas, tornado a análise mais confiável e automatizando o processo manual. O código em questão está descrito abaixo:

```
import pandas as pd
import numpy as np

# Nomes dos arquivos
files = {
    'fifo1': {'process': 'fifo1_process.csv', 'ticks':
'fifo1_ticks.csv'},
    'fifo2': {'process': 'fifo2_process.csv', 'ticks':
'fifo2_ticks.csv'},
    'fifo3': {'process': 'fifo3_process.csv', 'ticks':
'fifo3_ticks.csv'},
    'rr1': {'process': 'rr1_process.csv', 'ticks': 'rr1_ticks.csv'},
    'rr2': {'process': 'rr2_process.csv', 'ticks': 'rr2_ticks.csv'},
    'rr3': {'process': 'rr3_process.csv', 'ticks': 'rr3_ticks.csv'},
}

results = {}

def calculate_metrics(name, process_df, ticks_df):

    # 1. Métrica por Processo
    process_metrics = []
    pids = process_df['pid'].unique()

    # Tempo de Criação (Assumindo clock = 1 para todos)
    creation_time = 1
    num_processes = len(pids)

    for pid in pids:
        df_proc = process_df[process_df['pid'] == pid]

        # Tempo de Término (Turnaround)
        # O último clock em que o processo está como TERMINATED
        termination_df = df_proc[df_proc['state'] == 'TERMINATED']
        if termination_df.empty:
            # Se o processo não terminou, ele não entra na média de
Turnaround/Wait/Response
```

```
            continue

        termination_time = termination_df['clock'].max()

        # Tempo de Primeira Execução (Resposta)
        first_run_time_df = df_proc[df_proc['state'] ==
'RUNNING']['clock'].min()
        first_run_time = first_run_time_df if not
pd.isna(first_run_time_df) else np.nan

        # Tempo Total de Espera (Ready)
        waiting_time = df_proc[df_proc['state'] ==
'READY']['clock'].count()

        # Cálculo das métricas por processo
        turnaround_time = termination_time - creation_time
        response_time = first_run_time - creation_time if not
pd.isna(first_run_time) else np.nan

        process_metrics.append({
            'pid': pid,
            'turnaround_time': turnaround_time,
            'response_time': response_time,
            'waiting_time': waiting_time
        })

    metrics_df = pd.DataFrame(process_metrics).dropna()

    # 1. Turnaround Médio
    avg_turnaround = metrics_df['turnaround_time'].mean()

    # 2. Tempo Médio de Resposta
    avg_response = metrics_df['response_time'].mean()

    # 3. Tempo Médio de Espera
    avg_waiting = metrics_df['waiting_time'].mean()

    # 5. Fairness (Justiça) - Coeficiente de Variação do Turnaround
    std_turnaround = metrics_df['turnaround_time'].std()
    fairness_cv = (std_turnaround / avg_turnaround) * 100 if
avg_turnaround else np.nan

    # 4. Throughput
    max_clock = ticks_df['clock'].max()
    finished_processes = ticks_df['terminated_size'].max()
    throughput = finished_processes / max_clock
```

```
    # 6. Utilização da CPU
    # A CPU está ocupada quando running_pid é > 0 (e não -1, que é tempo
ocioso)
    cpu_busy_ticks = ticks_df[ticks_df['running_pid'] >
0]['clock'].count()
    cpu_idle_ticks = ticks_df[ticks_df['running_pid'] ==
-1]['clock'].count()

    # Tempo total é o max_clock, que é igual a soma de busy e idle
(inclui o último tick)
    cpu_utilization = (cpu_busy_ticks / max_clock) * 100

    results[name] = {
        'Turnaround Médio': f"{avg_turnaround:.2f}",
        'Tempo Médio de Resposta': f"{avg_response:.2f}",
        'Tempo Médio de Espera': f"{avg_waiting:.2f}",
        'Throughput (Proc/Clock)': f"{throughput:.4f}",
        'Fairness (CV Turnaround) (%)': f"{fairness_cv:.2f}",
        'Utilização da CPU (%)': f"{cpu_utilization:.2f}"
    }

# Executar a função para todos os arquivos
for name, file_pair in files.items():
    try:
        process_df = pd.read_csv(file_pair['process'])
        ticks_df = pd.read_csv(file_pair['ticks'])
        calculate_metrics(name, process_df, ticks_df)
    except Exception as e:
        results[name] = f"Erro ao processar: {e}"

# Criação da tabela final de comparação
comparison_df = pd.DataFrame(results).T
comparison_df = comparison_df.apply(pd.to_numeric)
comparison_df.index.name = 'Simulação'

comparison_df.head()
```

Com base no retorno do script, chegamos as seguintes tabelas abaixo:

1. Resultados de todas as simulações com as métricas solicitadas:

Simulação	Turnaround Médio	Tempo Médio de Resposta	Tempo Médio de Espera	Throughput (Proc/Clock)	Fairness (CV Turnaround) (%)	Utilização da CPU (%)
FIFO 1	358.50	22.00	194.50	0.0226	50.14	80.26
FIFO 2	599.50	11.40	338.20	0.0147	37.14	88.00
FIFO 3	772.30	15.60	479.90	0.0090	31.91	92.42
RR 1	632.00	2.50	377.60	0.0144	54.10	83.33
RR 2	733.20	2.50	473.80	0.0116	60.11	93.37
RR 3	696.00	2.50	436.60	0.0141	63.63	93.38

2. Análise dos Resultados

1. Turnaround Médio

Algoritmo	Média	Melhor Caso	Pior Caso
FIFO	576.70	**358.50 (FIFO 1)**	772.30 (FIFO 3)
RR	687.07	632.00 (RR 1)	733.20 (RR 2)

Aqui o FIFO 1 demonstrou a menor média de tempo total de execução. Isso se deve ao fato que o FIFO tem uma capacidade melhor de desempenho se os processos com pouca carga da cpu total estiverem no início da fila.

2. Tempo Médio de Resposta (Menor é melhor)

Algoritmo	Média	Melhor Caso	Pior Caso
FIFO	16.33	11.40 (FIFO 2)	22.00 (FIFO 1)
RR	2.50	2.50 (Todos RR)	2.50 (Todos RR)

Podemos verificar aqui a grande vantagem do Round Robin, como ele o divide a execução em quantus pequenos e consegue alterar rapidamente entre os processos ele garante que todos os processos recebam a CPU mais rapidamente, diferente do FIFO que todos os processos têm que esperar os processos anteriores terminarem sua fatia da CPU.

3. Tempo Médio de Espera (Menor é melhor)

Algoritmo	Média	Melhor Caso	Pior Caso
FIFO	337.53	**194.50 (FIFO 1)**	479.90 (FIFO 3)
RR	429.33	377.60 (RR 1)	473.80 (RR 2)

Devido ao context switching o Round Robin pode introduzir um pequeno overhead, o que leva a aumentar ligeiramente o tempo de esperar em comparação ao FIFO, aqui o FIFO por ter processos com pouca CPU no início liberou a CPU mais rapidamente.

4. Throughput (Maior é melhor)

Algoritmo	Média	Melhor Caso	Pior Caso
FIFO	0.0154	0.0226 (FIFO 1)	0.0090 (FIFO 3)
RR	0.0134	0.0144 (RR 1)	0.0116 (RR 2)

O FIFO 1 teve o menor tempo total, o que resultou no maior *throughput*. Nas condições médias que observamos no (FIFO 2, RR 1, RR 2, RR 3), o desempenho é similar.

5. Fairness (CV Turnaround) (Menor é melhor)

Algoritmo	Média	Melhor Caso	Pior Caso
FIFO	39.72	31.91 (FIFO 3)	50.14 (FIFO 1)

RR	59.28	54.10 (RR 1)	63.63 (RR 3)

No tópico Fairness o FIFO demonstrou menor variação no tempo de turnaround, isso por que a ordem de execução do FIFO é fixa. A variação que vemos no turnaround é causada pela diferença da carga de trabalho dos processos. Já no Round-Robin essa alternância de processos penaliza os processos longos.

6. Utilização da CPU (Maior é melhor)

Algoritmo	Média	Melhor Caso	Pior Caso
FIFO	86.89	80.26 (FIFO 1)	92.42 (FIFO 3)
RR	90.03	83.33 (RR 1)	93.38 (RR 3)

Por último, o Round Robin teve uma utilização maior de CPU, fazendo com que o RR seja mais eficaz em manter a CPU ocupada, evitando que a CPU fique ociosa.

3. Discussão

3.1 Como o quantum influência no desempenho do sistema?

R: Para o RR, o quantum é crucial, pois ele define o equilíbrio entre agilidade e eficiência, se o quantum for pequeno o sistema se torna responsivo garantindo que novos processos recebam a CPU, mas essa agilidade pode custar caro em termos de eficiência, pois a CPU gasta muito tempo em trocas de contexto. Por outro lado, se o Quantum for grande, o RR começa a se comportar como o FIFO e o sistema se torna eficiente, contudo esta eficiência vem ao custo da agilidade, pois o processo pode monopolizar a CPU por um período maior. Portanto o quantum ideal é um valor grande o suficiente para minimizar o overhead desnecessário e pequeno o suficiente para garantir a agilidade.

3.2 Qual política apresenta maior justiça, resposta e utilização da CPU.

R: Na ordem da pergunta: FIFO, Round Robin e Round Robin.

3.3 Como o tipo de processo (CPU-bound / IO-bound) afeta o comportamento do escalonador.

R: O escalonador FIFO é muito afetado pelo tipo do processo, se um processo CPU-Bound chegar primeiro, ele irá monopolizar o processador até ser concluído, o que irá causar um desempenho ruim para os outros processos, resultando em um longo tempo de espera e de resposta para todos os processos. Por outro lado um processo I/O-bound é ideal para o FIFO, pois irá liberar a CPU rapidamente, portanto o risco do FIFO está no seu potencial de starvation de processos curtos e I/O-bound por um único processo CPU-bound.