Atividade 2: Análise do Escalonador MLFQ

Objetivo

Executar o simulador `mlfq.py` com diferentes configurações de filas, quanta e boosts, analisando a saída para compreender o funcionamento do escalonador MLFQ (Multilevel Feedback Queue), com foco em prioridade, aging, escalonamento e justiça.

Parâmetros relevantes do simulador

`-n`	Número de filas de prioridade.
`-q`	Quantum fixo para todas as filas.
`-Q`	Lista de quanta por fila, ex: `-Q 5,10,20`.
`-B`	Intervalo de boost (em unidades de tempo) que restaura todos os jobs para a prioridade máxima.
`-j`	Número de jobs gerados aleatoriamente.
`-l`	Lista manual de jobs: `start,run,io:start,run,io:...`.
`-m`, `-M`	Máximo tempo de execução / frequência de I/O dos jobs aleatórios.
`-i`	Tempo fixo de I/O.
`-S`	Reinicia ticks e mantém prioridade ao fazer I/O.
`-I`	Ao terminar I/O, move o job para o início da fila.
`-c`	Exibe estatísticas finais automaticamente.

Experimentos

Experimento 1 – Comportamento Básico do MLFQ

```
python mlfq.py -n 3 -Q 5,10,20 -B 30 -l 0,20,0:2,8,0:4,12,0 -c
```

a) Qual foi o tempo de resposta e turnaround de cada job?

b) Algum job desceu de prioridade? Em que momento?

c) O boost ocorreu? Qual foi seu impacto?

Experimento 2 – Alterando o Quantum

Modifique o comando anterior para:

–Q 2,4,8

a) O tempo de resposta dos jobs aumentou ou diminuiu?

b) Houve mais ou menos trocas de contexto?

c) Quais efeitos foram observados em jobs curtos?

Experimento 3 – Starvation e Boost

Execute:

python mlfq.py –n 3 –Q 5,10,20 –B 10000 –l 0,60,0:2,60,0:4,60,0:6,5,0 –c

Depois, altere para:

–B 20

a) O job mais curto sofreu *starvation*?

b) O que mudou ao incluir o boost frequente?

Experimento 4 – Testando `-S` e `-I`

python mlfq.py –n 3 –Q 3,6,10 –B 50 –l 0,60,5:0,60,5 –S –I –c

a) Qual job teve menor turnaround?

b) Que diferença os parâmetros `-S` e `-I` provocaram?

c) O comportamento do escalonador foi justo?