

**1. Execute alguns problemas gerados aleatoriamente com apenas dois trabalhos e duas filas; calcule o rastreamento de execução do MLFQ para cada um. Faça sua vida mais fácil limitando a duração de cada trabalho e desligando I/O.**

Dois trabalhos -j 2;

Duas filas -n 2;

Limitando tamanho -m 2;

Desativando I/O -M 0

Simulando o comportamento de um escalonador mlfq:

```
run_mlfq(['-n', '2','-j','2', '-m', '2','-M','0', '-l', '0,30,0:20,20,0', '-c'], outfile)
```

De T = 0 a T = 10 a tarefa 0 executa com prioridade 1

De T = 10 a T = 20 a tarefa 0 executa com prioridade 0

De T = 20 a T = 30, a tarefa 1 chega com prioridade 1

De T = 30 a T = 40 a tarefa 0 executa até o fim com prioridade 0

A partir de T = 40 a tarefa 1 executa até o fim.

Você consegue mais informações no arquivo output\_cap8\_1.txt

**2. Como você executaria o escalonador para reproduzir cada um dos exemplos do capítulo?**

**Exemplo 1: A Single Long-Running Job**

-n =3 → 3 filas

-l = 0:200:0 → inicio:fim:i/o

```
run_mlfq(['-n','3','-l','0,200,0','-c'],outfile)
```

**Exemplo 2: Along Came A Short Job**

Tarefa B chega no T= 100ms e dura por 20ms

```
run_mlfq(['-n','3','-l','0,180,0:100,20,0','-c'],outfile)
```

**Exemplo 3: : What About I/O?**

-S ativo para manter a propriedade da regra 4b da tarefa b. Mantendo-o no nível 2 (alta prioridade).

Tarefa B sendo executada desde T = 50ms, 25x, com I/O de 1ms

-i I/O a cada 5ms

```
run_mlfq(['-S','-n','3','-l','0,175,0:50,25,1','-i','5','-c'],outfile)
```

**3. Como você configuraria os parâmetros do escalonador para se comportarem exatamente como um escalonador round-robin?**

Definindo o número de filas no MLFQ como 1. Isso cria uma fila única, eliminando o comportamento multinível do MLFQ e simulando o round-robin, onde todos os processos são tratados igualmente.

Com isso, temos a atuação da regra 2:

Rule 2: If Priority(A) = Priority(B), A & B run in RR.

Colocando o número de filas igual a 1: -n = 1

```
run_mlfq(['-n','1', '-q','5','-l','0,10,0:0,10,0','-c'],outfile)
```

**4. Crie uma carga de trabalho com dois jobs e parâmetros do escalonador para que um trabalho aproveita as antigas Regras 4a e 4b (ativadas com a flag -S) para manipular o escalonador e obter 99% da CPU durante um determinado intervalo de tempo.**

**Regra 4a:** Uma tarefa que usa menos de 50% do seu quantum recebe um aumento de prioridade.

**Regra 4b:** Uma tarefa que usa todo o seu quantum sem liberar a CPU é rebaixada.

Revisar

=====

```
run_mlfq(['-S', '-n', '2', '-q', '100', '-l', '0,100,99:0,100,0', '-c'], outfile)
```

Tarefa 0: Executa por 100 quantum, durante as quais usa 99% da CPU.

Pelos parâmetros de carga de trabalho fornecidos, a Tarefa 0 usará quase todo o tempo da CPU (99%) durante seu intervalo.

**5. Dado um sistema com comprimento quântico de 10 ms em sua fila mais alta, com que frequência você teria que impulsionar os empregos de volta à prioridade mais alta nível (com a flag -B) para garantir que um único trabalho de longa duração (e potencialmente faminto) obtenha pelo menos 5% da CPU?**

Revisar

=====

Para garantir que uma tarefa obtenha pelo menos 5% da CPU com quantum de 10 ms na fila de maior prioridade, devemos promover a tarefa de volta ao nível de prioridade mais alto a cada 200 ms ( $B \geq 200$ ).

EX:

```
run_mlfq(['-S', '-n', '2', '-q', '10', '-B', '200', '-c'], outfile)
```

**6. Uma questão que surge no escalonador é qual final da fila adicionar um trabalho que acabou de concluir a I/O; a flag -I altera esse comportamento para este simulador de escalonador. Experimente algumas cargas de trabalho e veja se você consegue ver o efeito desse sinalizador.**

Com a flag -I, uma tarefa volta a executar assim que terminar o I/O, caso não tenha terminado seu quantum.

```
run_mlfq(['-i', '1', '-n', '2', '-l', '0,20,5:0,20,5', '-c'], outfile)
run_mlfq(['-I', '-i', '1', '-n', '2', '-l', '0,20,5:0,20,5', '-c'], outfile)
```

Devido à flag -I no segundo caso. A tarefa que completou uma operação de I/O continuou a ser executado sem ser movido para o final da fila, impactando ligeiramente a distribuição do tempo de CPU entre as tarefas e resultando em tempos de turnaround levemente diferentes em comparação ao caso sem -I.