HW3

姓名: 陈锐林, 学号:21307130148

2023 年 9 月 24 日

Chapter7

Question1

这题中三个任务用时相同，SJF 和 FIFO 策略下，周转时间和响应时间是一样的。周转时间 =(200+400+600)/3=400，而响应时间 =(0+200+400)/3=200。-c 验证后也确实如此。

Question2

这题中三个任务用时不同,SJF 和 FIFO 策略下,周转时间和响应时间可能会不一样，但是因为三个任务以 100/200/300 的顺序到达，于是两个策略的执行顺序仍是一样的。于是周转时间 =(100+300+600)/3=333.3,而响应时间 =(0+100+300)/3=133.3。-c 验证后也确实如此。

Question3

采用 RR 策略，显而易见地在响应时间上会有很大进步，为 (0+1+2)/3=1；而周转时间会变大不少，为 (598+599+600)/3=599。-c 验证后也确实如此。

Question4

根据前面几题和课本内容可以知道，要让 SJF 和 FIFO 策略的周转时间相同，工作负载需要满足：(1) 工作不同时到达；(2) 同时到达的工作需要以递增的方式排序（即 FIFO 策略近似为 SJF 策略调度后的任务序列）。

Question5

若现在任务序列长度为$a_1, a_2, a_3, \cdots$,(递增)。对于 SJF 策略,响应时间就是$0, a_1, a_1+a_2, \cdots$,，而对于 RR 策略，时间片为 x，则响应时间为$0, x, 2x, 3x, \cdots$,，那么要相同也就意味着每个任务的长度要等于 RR 的时间片。

Question6

根据 SJF 的安排，随着作业长度的增加，会导致后续任务的开始时间更晚；所以这题答案是响应时间会增加。而利用命令"python3 ./scheduler.py -p SJF -l 100,100,100 -c"，安排三个相同长度的作业；从 100-1000，取 10 次也能看出，响应时间从 100 递增到 1000。

Question7

设 RR 策略的时间片为 T，共调度 N 个任务，假设任务长度 $\gg T$，那么平均响应时间就是 (0+T+2T+$\cdots$+(N-1)T)/N=$\frac{n(n-1)}{2n}T$。

Chapter8

Question1

根据题目的要求，两个工作/两个队列，限制作业长度和关闭 IO。调用如下命令"python3 ./mlfq.py -n 2 -j 2 -m 15 -M 0 -s 50"：两个任务长度分别是 7 和 9。根据计算策略，先进行 7 再进行 9，（两个队列时间片都是 10）；所以会得到响应时间为 (0+7)/2=3.5，周转时间为 (7+16)/2=11.5s。

Question2

(1)Figure8.2：单个长工作的例子；可调用命令："python3 ./mlfq.py -n 3 -l 0,200,0 -q 10 -c"，该工作逐渐从最高优先级往下掉。

(2)Figure8.3：即某个任务运行时，有新任务插入到最高优先级队列；可调用命令："python3 ./mlfq.py -n 3 -q 10 -l 0,180,0:100,20,0 -c"，时间片设为 10，第一个任务 0ms 开始，需做 180ms；第二个任务 100ms 加入，需做 20ms。

(3)Figure8.4：针对混合 I/O 密集型和 CPU 密集型，调用："python3 ./mlfq.py -n 3 -Q 10,10,5 -l 0,175,0:50,25,1 -i 5 -S -c"。

(4)Figure8.5：不采用优先级提升，调用："python3 ./mlfq.py -n 3 -q 10 -l 0,120,0:100,50,5:100,50,5 -i 5 -S -c"；采用优先级提示，调用"python3 ./mlfq.py -n 3 -q 10 -l 0,120,0:100,50,5:100,50,5 -i 5 -S -B 50 -c"。可以看到响应时间由 1.67 变为 1.0；周转时间也降低 1s 左右。

(5)Figure8.6：不采用愚弄反制时，调用"python3 ./mlfq.py -n 3 -q 10 -i 1 -S -l 0,200,0:80,100,9 -c"；调用时去掉-S 即可；从结果来看，不采取反愚弄机制会导致一个任务占据 CPU 过久。

(6)Figure8.7：要体现优先级越低，时间片越长，可以调用："python3 ./mlfq.py -n 3 -a 2 -Q 10,20,40 -l 0,200,0:0,200,0 -c"，设置时间片为 10/20/40，从时间片为 10 的最高优先级队列开始执行。

Question3

根据题目要求，要达到轮转调度类似的效果，应该满足：$T \leq \frac{maxL}{numJ}$，其中 T 是时间片大小；maxL 是任务的最大长度；numJ 是任务数。

Question4

调用如下命令：”python3 ./mlfq.py -n 3 -l 0,200,0:50,99,9 -i 1 -S -c”。这样能得到三个队列，时间片都是 10；io 时间为 1；第一个任务 0ms 开始，需做 200ms，ioFreq 为 0；第二个任务 50ms 开始，需要使用 99ms，ioFreq 为 9。于是在这条命令下，总执行 299s，在 100s 多的阶段，会出现第二个任务长时间占据 CPU 的情况。

Question5

为了保证能占据至少 5% 的 CPU，在时间片为 10ms 的情况下，应该让提升的时间间隔在 10/5%，即 200ms 之内提升一次，让该任务提高到最高优先级，并且得到执行。考虑到其他任务的影响，应该在 200ms 以内更好。

Question6

这题是探究完成 I/O 的作业插入在队列哪个位置对最后执行时间的影响。我们可以用接下来两条代码简单模拟：”python3 ./mlfq.py -n 2 -q 10 -l 0,50,0:0,50,11 -i 1 -S -c” 和”python3 ./mlfq.py -n 2 -q 10 -l 0,50,0:0,50,11 -i 1 -S -I -c”。第一条代码意为 iobump=False，即 I/O 返回后并不会马上执行该任务；而第二条代码加上了-I 标记，意为 iobump=True，即 I/O 返回后马上执行。于是分析结果我们会发现，用到 I/O 的任务在周转时间上得到了提升；但是在这个例子中整体的效率在下降（大大减缓了另一个任务的完成）。

以下是一些运行截图：

```
Final statistics:
  Job  0: startTime   0 - response   0 - turnaround   7
  Job  1: startTime   0 - response   7 - turnaround  16

  Avg  1: startTime n/a - response 3.50 - turnaround 11.50
```

图 1: Question1

```
[ time 139 ] Run JOB 0 at PRIORITY 0 [ TICKS 1 ALLOT 1 TIME 141 (of 200) ]
[ time 140 ] IO_DONE by JOB 1
[ time 140 ] Run JOB 1 at PRIORITY 2 [ TICKS 9 ALLOT 1 TIME 17 (of 99) ]
[ time 141 ] Run JOB 1 at PRIORITY 2 [ TICKS 8 ALLOT 1 TIME 16 (of 99) ]
[ time 142 ] Run JOB 1 at PRIORITY 2 [ TICKS 7 ALLOT 1 TIME 15 (of 99) ]
[ time 143 ] Run JOB 1 at PRIORITY 2 [ TICKS 6 ALLOT 1 TIME 14 (of 99) ]
[ time 144 ] Run JOB 1 at PRIORITY 2 [ TICKS 5 ALLOT 1 TIME 13 (of 99) ]
[ time 145 ] Run JOB 1 at PRIORITY 2 [ TICKS 4 ALLOT 1 TIME 12 (of 99) ]
[ time 146 ] Run JOB 1 at PRIORITY 2 [ TICKS 3 ALLOT 1 TIME 11 (of 99) ]
[ time 147 ] Run JOB 1 at PRIORITY 2 [ TICKS 2 ALLOT 1 TIME 10 (of 99) ]
[ time 148 ] Run JOB 1 at PRIORITY 2 [ TICKS 1 ALLOT 1 TIME 9 (of 99) ]
[ time 149 ] IO_START by JOB 1
IO DONE
[ time 149 ] Run JOB 0 at PRIORITY 0 [ TICKS 0 ALLOT 1 TIME 140 (of 200) ]
[ time 150 ] IO_DONE by JOB 1
[ time 150 ] Run JOB 1 at PRIORITY 2 [ TICKS 9 ALLOT 1 TIME 8 (of 99) ]
[ time 151 ] Run JOB 1 at PRIORITY 2 [ TICKS 8 ALLOT 1 TIME 7 (of 99) ]
[ time 152 ] Run JOB 1 at PRIORITY 2 [ TICKS 7 ALLOT 1 TIME 6 (of 99) ]
[ time 153 ] Run JOB 1 at PRIORITY 2 [ TICKS 6 ALLOT 1 TIME 5 (of 99) ]
[ time 154 ] Run JOB 1 at PRIORITY 2 [ TICKS 5 ALLOT 1 TIME 4 (of 99) ]
[ time 155 ] Run JOB 1 at PRIORITY 2 [ TICKS 4 ALLOT 1 TIME 3 (of 99) ]
[ time 156 ] Run JOB 1 at PRIORITY 2 [ TICKS 3 ALLOT 1 TIME 2 (of 99) ]
[ time 157 ] Run JOB 1 at PRIORITY 2 [ TICKS 2 ALLOT 1 TIME 1 (of 99) ]
[ time 158 ] Run JOB 1 at PRIORITY 2 [ TICKS 1 ALLOT 1 TIME 0 (of 99) ]
[ time 159 ] FINISHED JOB 1
```

图 2: Question4-任务 1 长时间占用 CPU

```
Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  72
  Job  1: startTime   0 - response  10 - turnaround 102

  Avg  1: startTime n/a - response 5.00 - turnaround 87.00
```

(a) 无-I

```
Final statistics:
  Job  0: startTime   0 - response   0 - turnaround  93
  Job  1: startTime   0 - response  10 - turnaround 101

  Avg  1: startTime n/a - response 5.00 - turnaround 97.00
```

(b) 有-I

图 3: Question6