Homework 5

CS211 - Fall 2024

(You should try to answer first and then compare your solution with the ChatGPT solution)

This homework is designed to help you become familiar with scheduling concepts and algorithms in operating systems

1. Let P_1, P_2, P_3, P_4 be processes with execution times of 53, 8, 68, and 24, respectively. Find the average waiting time and average completion time for the following scheduling scenarios: the best-case FCFS, the worst-case FCFS, and Round-Robin with quantum values of 1, 5, 8, 10, and 20.

2. Why does Shortest Remaining Time First (SRTF) scheduling help achieve the best response time and I/O throughput? Provide an example to illustrate your explanation.

3. What is the CPU scheduling policy of Linux? Briefly describe this policy and explain its strengths and weaknesses (e.g., in terms of response time, fairness, overhead, etc.)

(1) Best case FCFS: P_2, P_4, P_1, P_3

- avg waiting time: $\frac{0+8+(8+24)+(8+24+53)}{4} = 31.25$
- avg completion time: $\frac{8+(8+24)+(8+24+53)+(8+24+53+68)}{4} = 69.5$

Worst case FCFS: P_3, P_1, P_4, P_2

- avg waiting time: $\frac{0+68+(68+53)+(68+53+24)}{4} = 83.5$
- avg completion time: $\frac{68+(68+53)+(68+53+24)+(68+53+24+8)}{4} = 121.75$

Round-robin assuming FCFS order is: P_1, P_2, P_3, P_4

RR quantum = 1:

- avg waiting time: $\frac{0+1+2+3}{4} = 1.5$
- avg completion time: $$\left.\begin{array}{l}(53+8+52+24) \\ +(8+8+7+7) \\ +(68+53+8+24) \\ +(24+8+24+24)\end{array}\right| 4 = 100$$

RR quantum = 5:

- avg waiting time: $\frac{0+5+(5+5)+(5+5+5)}{4} = 7.5$
- avg completion time: $$\left.\begin{array}{l}(53+8+50+24) \\ +(8+10+5+5) \\ +(68+53+8+24) \\ +(24+25+8+25\end{array}\right| 4 = 99.5$$

RR quantum = 8

- avg waiting time: $\frac{0+8+16+24}{4} = 12$
- avg completion time: $\left.\begin{array}{l}(53+8+48+24)\\+(8+8)\\+(68+53+8+24)\\+(24+24+8+24)\end{array}\right| 4 = 95.5$

RR quantum = 10:

- avg waiting time: $\frac{0+10+(10+8)+(10+8+10)}{4} = 14$
- avg completion time: $\left.\begin{array}{l}(53+8+50+24)\\+(10+8)\\+(68+53+8+24)\\+(24+30+8+30)\end{array}\right| 4 \approx 995$

RR quantum = 20:

- avg waiting time: $\frac{0+(20+8)+(20+8+20)}{4} = 19$
- avg completion time: $\left.\begin{array}{l}(53+8+40+24)\\+(20+8)\\+(68+53+8+24)\\+(24+8+40+40)\end{array}\right| 4 = 104.5$

(2) Avg response time : $\frac{\sum_{i}^{n}(\text{Time job } i \text{ gets started} - \text{Time job } i \text{ enters queue})}{n}$

Avg I/O throughput: $\frac{\text{Number of jobs done}}{\text{Unit of time}}$ or $\frac{\text{Number of all jobs } (n)}{\text{Total time to complete all jobs}}$

SRTF prioritize shorter jobs → Reduce overall waiting time across tasks on queue

↙ Reduce response time theo cấp số cộng (only big jobs suffer delay, but many short jobs can reduce waiting for big jobs to get done)

↳ Benefits from this if there are fewer big jobs

↘ Increase the count of jobs done ⇒ Better throughput

Example: P_1, P_2, P_3, P_4 with duration 53, 8, 68, 24
Assume all jobs enter queue at time $t=0$

	Avg response time	Avg throughput
Vanilla FCFS	$\frac{0+53+8+68}{4} = 32.25$	$\frac{4}{(0+53)+(53+8)+(53+8+68)+(53+8+68+24)}$ $= \frac{4}{396} \approx 1.0101\%$
SRTF	$\frac{0+8+24+53}{4} = 21.25$ Better than FCFS, not as good as RR	$\frac{4}{(0+8)+(8+24)+(8+24+53)+(8+24+53+68)}$ $= \frac{4}{278} \approx 1.4388\%$ → Better than FCFS and RR
Round robin quantum = 10	$\frac{0+10+18+28}{4} = 14$	$\frac{4}{99.5 \times 4}$ $= \frac{4}{398} \approx 1.005\%$

⇒ SRTF is good but not outstanding :)

③	FIFO	CFS	RR	Batch
Response time	Depends (long if big jobs arrive first, short otherwise)	Short (newer jobs has exec. time = 0 → higher priority → get started soon)	Short (wait for at most quantum duration * queue length)	Long (not focus on optimize response time)
Fairness	Doesn't care	Very fair (ensure fair share of CPU, prioritize jobs with fewer exec. time)	Very fair (ensure fair shair of CPU time)	Doesn't care
Overhead	Very little (only a basic queue, no extra context switching)	A lot. (Needs a RBTree / priority queue & frequent context switching)	Quite a lot (needs a queue and frequent context switching)	Very little (pre-compute execution order – one off cost)
Throughput	Relatively high (jobs are run to completion, no splitting)	Very low (job gets split, takes forever to done)	Very low (job gets split, takes forever to done)	High (designed to optimize throughput)
Avg - completion time	Short (jobs are run to completion, no split)	Long (jobs get split ⇒ takes forever)	Long (jobs get split ⇒ takes forever)	Short (jobs are run to completion, no split)
Avg - waiting time (exec time – queue time)	Long (must wait for previous job to done to start)	Short (jobs get started soon after enter queue)	Short (jobs get started soon after enter queue)	Long (must wait for previous job to done to start)

Linux provides 4 scheduling algorithms:

- SCHED_FIFO : First come first serve, jobs arrive earlier gets started first, jobs are run to completion
- SCHED_FAIR : Completely fair scheduler, Maintains a priority queue (RBTree), jobs with lesser CPU time gets bump to top of queue → Always select job that's been run the least
- SCHED_RR : Round robin algorithm, CPU time is divided into non-overlapping time slice (quantum), then jobs are rotated to run until done (yield) or timeout (preempt by scheduler)
- SCHED_BATCH : Batch scheduling is often used when we know the list of jobs in advance & can organize them into batches. Once sorted out, each batch is run with little to no scheduling modification.
  Jobs are often not split but run to completion.
  ↳ Suitable for jobs like training models on server.

Above information are extracted from this man page:
man7.org/linux/man-pages/man7/sched.7.html

SCHED_OTHER and SCHED_IDLE are not mentioned because I don't quite understand them (& their use case) at the point of writing this