Spring 2019 COMP 3511 Homework Assignment #2

Handout Date: March 8, 2019 Due Date: March 22, 2019

Name: _Huang Daoji_ ID: _20623420_E-Mail: _dhuangal@connect.ust.hk_

Please read the following instructions carefully before answering the questions:

- You should finish the homework assignment **individually**.
- There are total of **4** questions.
- When you write your answers, please try to be precise and concise.
- Fill in your name, student ID, email at the top of each page.
- Please fill in your answers in the space provided, or you can type your answers in the MS Word file.

- **Homework Submission**: the homework is submitted to **assignment #2** on **CASS**

1. [20 points] Multiple choices

   1) Which of the following components of program state are shared across threads in a multithreaded process?
   ①Register values ②Heap memory ③Global variables ④Stack memory

   A) ①②
   B) ②③
   C) ①④
   D) ③④
   Answer: _B_

   2) Which one of the following is not shared by threads?
   A) program counter
   B) stack
   C) both program counter and stack
   D) none of the mentioned
   Answer: _C_

   3) Which module gives control of the CPU to the process selected by the short-term scheduler?
   A) dispatcher
   B) interrupt
   C) scheduler
   D) none of the mentioned
   Answer: _A_

4) Cancelling a thread asynchronously _____.

A) spoils the process execution
B) may not free each resource
C) frees all the resources properly
D) allows the target thread to periodically check if it should be cancelled
Answer: _B_

5) According to Amdahl's Law, what is the speedup gain for an application that is 40% parallel and we run it on a machine with 4 processing cores?

A) 0.7
B) 1.82
C) 1.43
D) 0.55
Answer: _C_

6) Which of the following scheduling algorithms could result in starvation?
①First-come, first served ②Shortest job first (non-preemptive)
③Round Robin ④Priority

A) ①②
B) ②④
C) ①③
D) ③④
Answer: _B_

7) CPU scheduling is the basis of _____
A) multiprocessor systems
B) multiprogramming operating systems
C) larger memory sized systems
D) none of the mentioned.
Answer: _B_

8) One of the disadvantages of the priority scheduling algorithm is that:
A) it schedules in a very complex manner
B) its scheduling takes up a lot of time
C) it can lead to some low priority process waiting indefinitely for the CPU
D) none of the mentioned
Answer: _C_

9) LWP is _____.

A) short for lightweight processor
B) placed between user and kernel threads
C) placed between system and kernel threads
D) common in systems implementing one-to-one multithreading models
Answer: _B_

10) With round robin scheduling algorithm in a time shared system,
A) using very large time slices degenerates it to First Come First Served scheduling algorithm
B) using very small time slices degenerates it to First Come First Served scheduling algorithm
C) using extremely small time slices increases performance
D) using very small time slices degenerates it to Shortest Job First algorithm
Answer: _A_

2. [20 points] Multithread Process

1) What set of resources are needed or used when a thread within a process is created? How do they differ from those used when a process is created? (4 points)

i. A thread ID, its own PC, register set, stack and priority,its TCB

ii. When creating a process, the OS kernel need to create and maintain a PCB, which also includes a memory map, open files, process ID, and possible environment variables.

2) Linux does not distinguish between processes and threads. Instead, it treats them in the same way by allowing a task to be more akin to a process or a thread depending on the set of flags passed to the clone() system call. Please compare and contrast clone() and fork() in Linux. (4 points)

i. Clone() system call creates a child which shares a part of resources with its parent. Clone() receives a pointer to a function, which the child process starts to execute, and terminates when finishing the function. The recources shared can be specified by the parent using parameter(flags). When more recources are shared, the child process becomes more like a typical thread, and the less, more like an independent process.

ii. Fork() system call creates a child which shares no part of resources with parents(except the fact that the resources are the same). Fork() takes no parameters and the child process continues executing from the fork() funcion. The parent cannot set the resources shared by the child using fork().

3) Consider a multicore system and a multithreaded program using *many-to-many threading model*. Let the number of user-level threads in the program be much

greater than the number of processing cores in the system. How many kernel threads should the OS allocate in order to increase the utilization of the multiprocessor system, and why ? (4 points)

Hint: Consider the number of kernel threads allocated to the program is less than, equal to or is greater than the number of processing cores.

i. More than the number of processing cores.

ii. If the number of kernel threads is less than the processing cores, some processing cores become idle(for the scheduler only schedule kernel threads not user-level threads), and thus more computational power can be get when more kernel threads are allocated.

iii. If the number of kernel threads is equal to the processing cores. Consider a exterme case when all the kernel threads are preforming I/O instrutions, and are waiting for them, thus some CPU cores may idle. By allocating more kernel threads, we can use the time when the processing cores are blocking.

iv. When more kernel threads are allocated than the number of processors, a kernel thread may execute on the same core when one thread bokcks. So the utilization of CPU may be increased.

v. Given the fact that the number of user-level threads is much greater than the number of processign cores, there will not be more waste of time when a kernel thread waiting for an availble processing core, for the corresponding user-level thread will be waiting anyway, even if less kernel threads are allocated(in which case it will waiting for an availble kernel thread/LWP).

vi. In conclusion, the number of kernel thread should be somewhere between the number of processing cores and the number of user-level threads.

4) Using Amdahl's Law, list the speedup gain of an application in a table or figure that has a 60 percent parallel component for systems with one, two, three, … and eight processing cores. (8 points)

1	2	3	4	5	6	7	8
1	1.4286	1.6667	1.8182	1.9231	2	2.0588	2.1053

3. [20 points] CPU scheduling

1) Distinguish between PCS and SCS scheduling. Under what threading model, PCS and SCS are the same? (5 pints)

i. PCS schedules the user level therads to LWPs. It is usually done by user-level thread library, and take place among the threads belonging to the same process.

ii. SCS schedules kernel threads to availble CPUs among all the threads in the system.

iii. Under one-to-one mapping threading model, these two scheduling are the same.

2) Can FCFS be considered as one type of priority scheduling? Please be careful in justifying your answer. (4 points)

i. Yes

ii. The priority here is defined to be the arrival time of the process, which is smaller(higher in priority) for early arrived process. Thus FCFS can be considered as a nonpreemptive priority scheduling, for new proess will not preempt the executing process.

iii. When multiple processes arrive at the same time, no mattter which order are they being executed, it does not conflict with the fact that process with higher priority is executed first.

3) What is the difference between response time and turnaround time? How do they conflict with each other? (5 points)

i. Response time is th amount of time it takes until the first response is produced from the request is submitted.

ii. The turnaround time is the amount of time to execute a particualr process(from the time of submission to the time of completion).

iii. The (average) turnaround time is minimized when small jobs are executed first and less context switch happens. But (average) response time is then increased for longer jobs would waiting for their turn to be executed.

iv. The average response time can be minimized by scheduling jobs in turn, but then the turnaround time will increase for the time waiting for other jobs and time doing context switch.

4) CPU scheduling often needs to balance CPU-bound and I/O-bound programs. Suppose we design a simple CPU scheduling algorithm that favors those processes that have used the least CPU time in the recent past. Why will this algorithm favor I/O-bound programs and yet not permanently starve CPU-bound programs? Can this also improve I/O device utilization? (6 points)

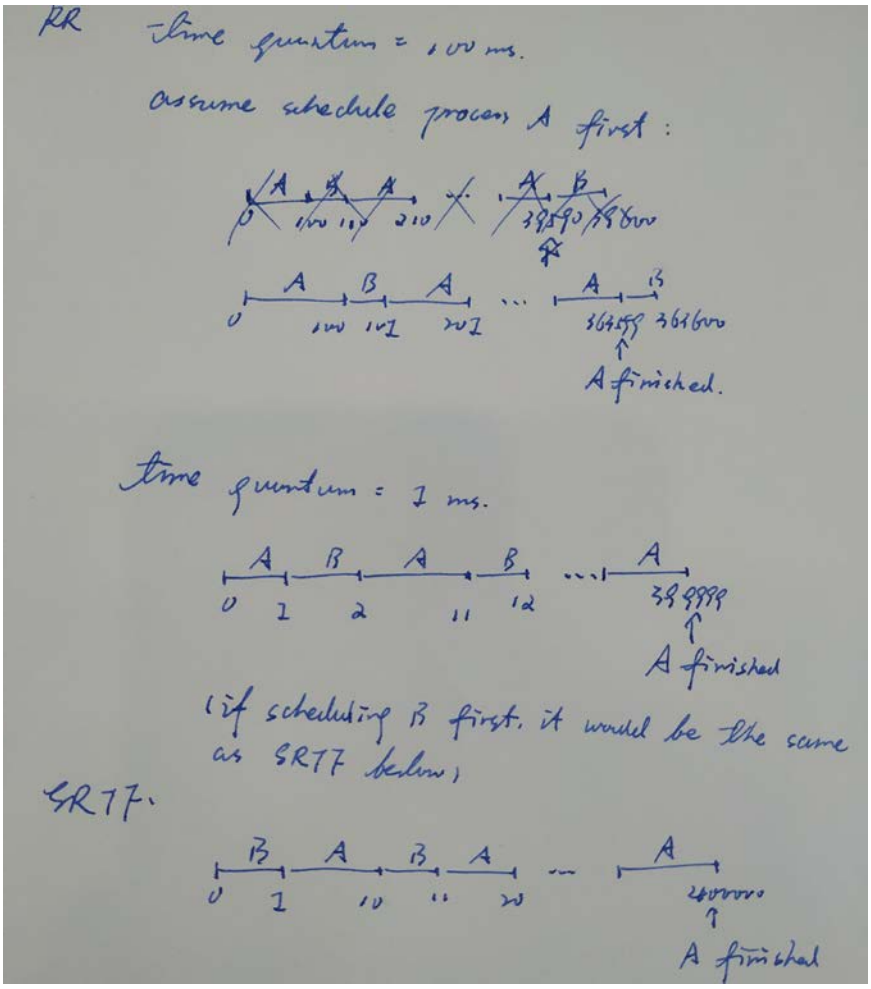
i. For I/O-bound programs, the CPU time in the recent past would be relatively smaller than CPU-bound process, so they are more likely to be scheduled to execute on processors.

ii. CPU-bound programs will not *permanently* be starved, for eventually the CPU time in the recent past for this CPU-bound program will be small, or even 0, so it will be scheduled to execute, thus no starvation will occur. It is also because the I/O bounded process will release the CPU quite often to wait for I/O, thus give a chance for CPU-bound process to execute.

iii. Yes, because this algorithm favors I/O-bound programs, more I/O instrutions will be executed, thus improve I/O device utilization.

4. [40 points] Scheduling algorithms

1) Consider two processes, process A is CPU-bound with CPU burst time one hour, and process B is I/O-bound repeating CPU burst time 1 ms and disk I/O 9 ms. Please illustrate using Gnatt charts the round-robin scheduling with a time quantum respectively setting to 100 ms and 1 ms, and SRTF scheduling. Please discuss which one performs the best (10 points)? Hint: consider disk utilization and number of context switch.



i. Above is the Gnatt charts showing of RR(with time quantum being 100ms and 1ms) and SRTF.

ii. With respect to disk utilization, the RR with time quantum 1ms and SRTF(which are almost the same in this case) are the best, because roughly speaking, these algorithms both achieves that disk IO is performing in 90% of the time, while RR with time quantum 100ms only spends (9 / 101) * 100% time performing IO.

iii. But with respect to the number of context switch, the RR with time quantum 100ms is better for context switch occurred less frequently.

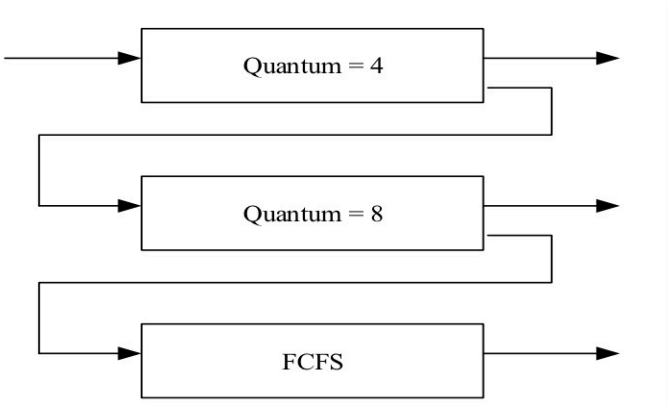
iv. The best algorithm thus depends: if process A is more important(or the overhead of context switch is large), then RR(100ms) would perform best, otherwise the other two algorithm would perform best.

2) Given the arrival time and CPU-burst times of 6 processes shown in the following diagram:

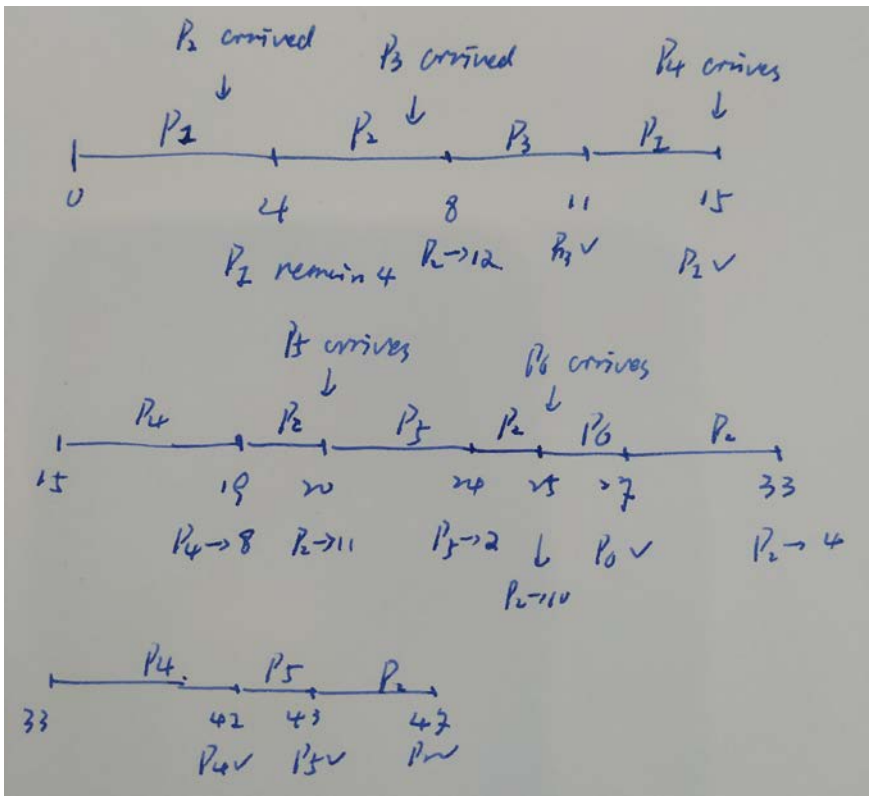
Process	Arrival Time (ms)	Burst Time (ms)
P1	0	8
P2	3	16
P3	7	3
P4	15	12
P5	20	6
P6	5	2

Suppose the OS uses a 3-level feedback queue to schedule the above 6 processes. Round-Robin scheduling strategy is used for the queue with the highest priority and the queue with the second highest priority, but the time quantum used in these two queues is different. First-come-first-serve scheduling strategy is used for the queue with the lowest priority. The scheduling is **preemptive**.

(Note: In this scenario, the scheduling is preemptive, which means that the execution of the current job may be preempted by another job with **higher** priority. a newly arriving job can preempt a job currently running only if its priority is **higher** than this job)



Construct a Gantt chart depicting the scheduling for the set of processes specified in the above diagram using this 3-level feedback queue, and compute the average waiting time for all processes (10 points)

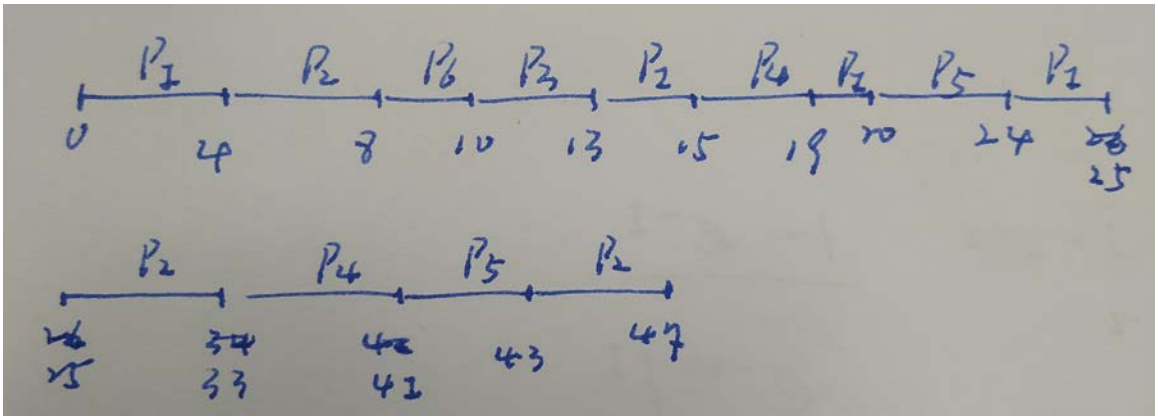


i. Assume the arrival time of P6 is 25, the Gnatt chart is given above, if the arrive time of P6 is 5, **the solution is shown below**.

ii. The turn around time of each process is: 15, 44, 4, 26, 23, 2. The waiting time of each process is: 7, 28, 1, 14, 17, 0

iii. Thus, the average waiting time is 11.1667.

iv. The solution that P6 arrives at 5 is shown below:



v. The waiting time for each process is then 17, 28, 3, 14, 17, 3

vi. The average waiting time would be 13.6667

3) (20 points) Consider the following set of processes, with the length of the CPU burst time given in milliseconds:

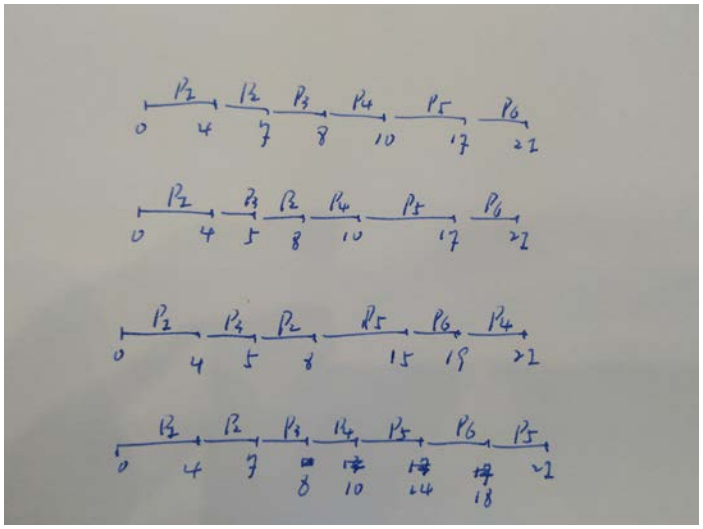
Process	Arrival Time(ms)	Burst Time(ms)
P1	0	4
P2	2	3
P3	3	1
P4	6	2
P5	7	7
P6	11	4

a) (10 points) Draw four Gantt charts that illustrate the execution of these processes using the scheduling algorithms listed below:

(i) FCFS
(ii) SJF
(iii) Non-Preemptive priority (a smaller priority number implies a higher priority), with the priorities listed here:

Process	Priority
P1	1
P2	4
P3	2
P4	5
P5	1
P6	3

(iv) RR (quantum = 4)



b) (4 points) What is the turnaround time of each process for each of the scheduling algorithms in part a?

Turnaround time	P1	P2	P3	P4	P5	P6
FCFS	4	5	5	4	10	10
SJF	4	6	2	4	10	10
Non-Preemptive priority	4	6	2	15	8	8
RR	4	5	5	4	14	7

c) (4 points) What is the waiting time of each process for each of these scheduling algorithms in part a?

Waiting time	P1	P2	P3	P4	P5	P6
FCFS	0	2	4	2	3	6
SJF	0	3	1	2	3	6
Non-Preemptive priority	0	3	1	13	1	4
RR	0	2	4	2	7	3

d) (2 points) Which of the algorithms results in the minimum average waiting time (over all processes) mentioned above?

i. SJF