7 Scheduling [10 points]

Traditionally, Unix-based operating systems implemented non-preemptive scheduling for kernel threads. In recent years, Unix kernels have increasingly implemented preemptive scheduling for kernel threads.

Part A [2 points]

Explain **two** benefits of non-preemptive scheduling over preemptive scheduling for kernel threads.

- Code is easier to write since a thread gives up the processor at well-known points (e.g, yield or sleep). In between, shared variables can be accessed without locking since the thread cannot be preempted.
- A non-preemptive system can be more efficient because context switching happens at well-known points (minimizes context switching).

Part B [2 points]

Explain **two** benefits of preemptive scheduling over non-preemptive scheduling for kernel threads.

- A single kernel thread cannot easily starve other kernel threads.
- Since kernel threads can be preempted, each thread can be more responsive, i.e., better interactive or real-time response
- A kernel developor does not have to worry that kernel threads can run for too long hampering the responsiveness of the rest of the system

Part C [2 points]

Unix systems use feedback-based scheduling for user processes. Is this scheduling preemptive or non-preemptive? Very briefly explain your answer.

Preemptive. User processes need to be scheduled preemptively or else they may never yield the CPU to other processes.

Part D [4 points]

Explain how feedback-based scheduling can help avoid starvation for both CPU and IO bound processes.

Feedback scheduling prioritizes IO-bound processes over CPU-bound processes. However, IO-bound processes cannot starve CPU-bound processes because if an IO-bound process starts using the CPU excessively then its "priority" value will increase and its priority will be reduced below a CPU-bound process that has not run for sometime. Hence the CPU-bound process will be able to run eventually.

4 Scheduling [12 points]

The table below shows the time at which 5 patients arrive at a hospital and the expected processing time needed to serve the patients. The times are shown in minutes.

Patient ID	Arrival Time	Processing Time
1	0	60
2	15	10
3	30	35
4	45	10
5	60	10

Part A [6 points]

Show the **scheduling order** for these patients by writing down the sequence of patient IDs that are served under first-come first-served (FCFS), shortest remaining time (SRT), and round-robin scheduling. For round-robin scheduling, the time quantum is 5 minutes. Use the space below to show the method you used to derive your answer. (Please read part B of this question before proceeding with your answer).

FCFS:

SRT:

Round Robin:

Question continues ...

Part B [6 points]

For each scheduling algorithm, show the **turnaround time** for each patient. Then calculate the average turnaround time for each of the algorithms.

	Turnaround Time		
Patient ID	FCFS	SRT	Round Robin
1			
2			
3			
4			
5			
Average time			

5 **Scheduling** [6 points]

Consider a Unix-style feedback scheduling algorithm that increments CPU usage by a unit amount for the currently running process every timer interrupt (T = 10ms). Then it updates the priority of all processes periodically every second. At this time, the CPU usage of each process is aged by setting it to half its existing value, and the priority value of each process is set to the sum of its new CPU usage value and an initial priority value. Then the scheduler chooses a ready process that has the smallest priority value.

Suppose there are only two processes in the system, an IO-bound process and a CPU-bound process. The IO-bound process, on average, takes 25% of the CPU. After these processes have been running for a long time, what will be the priority values of the two processes. Assume that the initial priority value of both processes is 0. Show your computation in the space provided below.

4 Scheduling [15 points]

Consider the Unix-style feedback scheduler in which the timer interrupt occurs every 10 ms and the time slice is 1 second. At each timer interrupt, the CPU usage (c) of the current process is incremented by 1. At the time slice, the priority (p) of each process is calculated as $p = p/2 + c$ and c is set to 0. Assume that the initial priority of all processes is 0. Suppose there are three processes in the system that run as follows: 1) Process P_0 runs for one second and then needs to wait for I/O for two seconds, 2) Processes P_1 and P_2 run for one second and then need to wait for I/O for one second. After waiting for I/O, each process is ready to run again. Answer the following questions assuming that the processes have been running for a long time.

Part A [5 points]

Show how the processes are scheduled in the steady state (i.e., show a schedule that will repeat over time) on the process timelines shown below. For each process, mark its states (* for Running, R for Ready, and W for Waiting) on the timeline. Make sure that the timing of scheduling events is indicated clearly (in ms or seconds).

P_0	*	W	W	
P_1	R	*	W	
P_2	W	R	*	
	1 sec	1 sec	1 sec	

Part B [2 points]

What is the CPU utilization during the steady state? What is the CPU allocation for each process?

Answer: CPU utilization is 1 or 100% (CPU is not waiting for IO at any time). CPU allocation is 1/3 (or 33.3%) for each process.

Part C [1.5 points]

What is the average response time during the steady state for each process? Recall that the response time is the time difference between when a process becomes runnable and when it starts running.

Answer: Response time (and average response time) for P_0 is 0. Response time (and average response time) for P_1 and P_2 is 1.

Part D [4.5 points]

What is highest priority value of each process in the steady state? Show your calculations.

Answer: The highest priority value P for any process will occur just after it has run. Consider Process P_2 in the diagram above. Assume that its priority at time 0 is P. At time 1, its priority will be P/2. At time 2, its priority will be P/4. At time 3, its priority will be P/8 + 100. But P/8 + 100 is also equal to P, since the cycle is repeating. Hence P/8 + 100 = P, or P = 800/7. This same argument applies to the other two processes.

Part E [2 points]

Now suppose that Process P_0 runs for one second and then needs to wait for I/O for three seconds. Show how the three processes are scheduled in steady state on a single timeline (i.e., show the times when the processes are running). Make sure that the timing of scheduling events is indicated clearly (in ms or seconds).

P_0	P_1	P_2	P_1	P_0	P_2	P_1	P_2

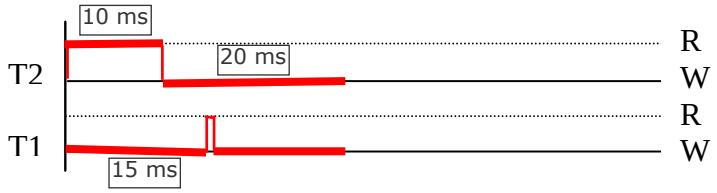
All events occur for 1 second.

4. Scheduler [10 marks]

You have been asked to implement a Unix-style feedback scheduler for 353OS. Your scheduler has a timeslice of 10 milliseconds. At each time slice, the priority (p) of each thread is calculated as p = p/2 + c, where c is the CPU usage of the thread in the last time slice.

You run 353OS on a machine with a CPU that executes 1000 MIPS (million of instructions per second) and a disk that takes 15 milliseconds to respond to each request. Assume that you only have to schedule two threads: Thread 1 has a huge memory footprint and generates a page fault every 100 instructions. Thread 2 is a cryptographic job that alternates reading a block from the disk and then computing for 10 milliseconds on the block. Answer the following questions assuming that the threads have been running for a long time. State any assumptions you make.

a) [4 points] Show how the threads are scheduled in the steady state (i.e., show a schedule that will repeat over time) on the timelines shown below. Make sure that the timing of scheduling events is indicated clearly (in ms). For each thread, show when it is running on the 'R' line and when it is waiting on the 'W' line as a square wave.



b) [2 points] What is the CPU utilization of your system? What would be the CPU utilization on a two processor system?

Utilization = 1/3
On dual processor, utilization = 1/6

c) [3 points] What is the range of the priority value for threads T1 and T2 in the steady state? Show your calculations.

T1:
c ~= 0, p = 0
T2:
When it starts running:
p = 1/4 * (p/2 + 10)
p = 20/7
When it stops running:
p = p/2 + 10 = 10/7+10 = 11 3/7

d) [1 points] Suggest a method that does not require any hardware modifications for improving the utilization of your system (in one line).

Swap out T1 and run it less often.

Question 5. Scheduling [8 MARKS]

Part (a) [3 MARKS] Explain the difference between a preemptive scheduler and a non-preemptive scheduler, and give an example of a type of system where each might be used.

A preemptive scheduler will force a context switch upon certain events (time slice expires, higher priority process becomes runnable, etc.) while a non-preemptive scheduler will wait for the running process to voluntarily yield the CPU (blocking or exiting) before scheduling a new process.

Preemptive systems: real time systems, time shared systems, interactive systems Non-preemptive systems: batch systems, single-application systems (e.g., some embedded systems) ... some real time systems are also non-preemptive, relying on cooperative scheduling to meet deadlines.

Part (b) [2 MARKS] The OS/161 scheduler runqueue is a linked list of pointers to thread structs. What would be the effect on scheduling if we put two pointers to the same thread struct into the runqueue?

A thread gets scheduled whenever its struct reaches the head of the runqueue, so this gives such threads twice as much CPU time as threads that are only in the runqueue once. (Sort of proportional share scheduling.)

Part (c) [1 MARK] What new problem could occur if we put two pointers to the same thread struct into the runqueue?

A thread could be scheduled to run while it is in the blocked state, because the second pointer to the thread struct gets to the head of the runqueue while the first is on some wait channel. [We would need to be careful to find and remove the second pointer from the runqueue when a thread blocks, or check the thread state when it is selected to run to make sure it is not blocked.]

Part (d) [2 MARKS] Briefly describe how you could achieve the same effect without the duplicate pointers.

Record the size of the scheduling quantum for each thread, so that some threads can have more CPU time (same effect as putting the pointer in the queue twice) and on timer interrupt, check if the current thread has used up its quantum, rather than giving all threads the same size of quantum.

Question 6. Round-Robin Scheduling [10 MARKS]

Measurements of a certain system have shown that the average process runs for a time **T** before blocking on I/O. A process switch requires time **S** which is effectively wasted (overhead). For round-robin scheduling with time quantum **Q**, give a formula for CPU utilization for each of the following:

Part (a) [2 MARKS] $Q = \infty$

U = T/(T+s)

Part (b) [2 MARKS] $Q > T$

U = T/(T+s)

The time quantum Q starts after a process starts running. Since Q > T, the process will block before the quantum expires, so the answer is the same Part (a).

Part (c) [2 MARKS] $S < Q < T$

U = Q/(Q+s)

Part (d) [2 MARKS] $Q = S$

U = 1/2

From the answer of Part (c), make Q=S.

Part (e) [2 MARKS] Q nearly 0

U = 0

3. Scheduling [6 marks; 3 each]

Most Unix systems provide a `setpriority` system call to set the priority of a user-level process. However, there is no guarantee about how the kernel CPU scheduler will actually use these priority values. Below, assume that there is no way for a process to get the current clock time from the OS.

a) Describe an experiment using user-level processes that would allow you to determine if the OS scheduler is using a fixed priority scheme and selecting processes according to their user-specified priority levels. Be sure to account for other factors (in addition to priority) that could affect scheduling.

Write a process that runs a tight loop and prints a message after, say, every 10000 iterations. Run two such processes at different priorities. If only the higher-priority process prints messages, then the OS scheduler is using fixed priorities. If the lower-priority process also prints messages, then the OS scheduler is not using fixed priorities.

b) Describe a similar experiment that would allow you to determine if the scheduler is dynamically adjusting priorities to prevent starvation of low-priority processes.

The same experiment above can help determine if the scheduler is dynamically adjusting priorities. The lower-priority process will print messages but should print them less frequently than the higher priority process if the scheduler adjusts priorities dynamically.

Question 3. Scheduling [10 MARKS]

There are four processes in the system. Two processes, A and B, are CPU-bound processes that are always runnable. The other two processes, X and Y, are IO-bound processes that run for 0.1 seconds and then block for 0.5 seconds. After blocking, they become runnable again.

Part (a) [5 MARKS] Show how the four processes are scheduled by a round-robin scheduler. The time slice of the scheduler is 0.1 seconds. Start by scheduling processes X, Y, A and B, as shown below. Show the process that is running in each time slice for the first 20 time slices. Assume that a process becomes runnable just before the arrival of a timer interrupt.

1	2	3	4	5	6	7	8	9	10
X	Y	A	B	A	B	A	X	B	Y

11	12	13	14	15	16	17	18	19	20
A	B	A	B	X	A	B	Y	A	B

Part (b) [5 MARKS] Show how the four processes are scheduled by a Unix-style feedback scheduler. Assume that timer interrupts arrive every 0.1 seconds and the time slice of the scheduler is 1 second. Start by scheduling processes X and Y as shown below. Assume that all processes have the same nice value.

Assuming X, Y, A, B all start at priority value = 0.

1	2	3	4	5	6	7	8	9	10
X	Y	A	A	A	A	A	A	A	A

11	12	13	14	15	16	17	18	19	20
B	B	B	B	B	B	B	B	B	B

Assuming A and B have been running for a while, their priority values will always be higher the priority values of X and Y.

X	Y	A	A	A	A	X	Y	A	A
B	B	X	Y	B	B	B	B	X	Y

Question 4. Hardware and OS Design [8 MARKS]

You plan to start a company that will design a simple, power-efficient processor for next generation smartphones. Your design must allow the mobile operating system to isolate processes from each other and protect itself from malicious processes. To do so, you are convinced that the simplest processor design requires only one privileged instruction. Convince your funding partner that your design will work.

Part (a) [2 MARKS] Describe exactly what operation the privileged instruction will perform in your processor.

The privileged instruction should be able to setup the MMU registers so that the OS can ensure memory protection.

There should be no IO instructions and all IO and interrupt access should be memory mapped.

Part (b) [4 MARKS] Explain how the OS will prevent the following:

1. A process directly modifies kernel data:

The OS uses the MMU to ensure that kernel memory is not mapped to user space, so the process cannot access kernel memory.

2. A process directly executes arbitrary kernel code:

The OS uses the MMU to ensure that kernel memory is not mapped to user space, so the process cannot access kernel memory.

3. A process writes directly to a network device:

IO access is memory mapped. This memory range lies in kernel memory.

4. A process disables interrupts:

Interrupt access is memory mapped. This memory range lies in kernel memory.

Part (c) [2 MARKS] Describe one way by which a malicious process may still be able to compromise the OS.

OS has bugs or vulnerabilities. A process can issue a system call with bad/crafted arguments, that cause the OS to crash, reveal information to user space, etc.

Question 3. Scheduling [10 MARKS]

There are three processes, A, B and C in the system. Process A is CPU-bound and always runnable. Processes B and C are IO-bound. After Process B runs for 1 unit of time, it needs to block for 1 unit of time. After Process C runs for 1 unit of time, it needs to block for 3 units of time.

Part (a) [6 MARKS] Show how the three processes are scheduled by a round-robin scheduler. The time slice of the scheduler is 1 unit of time. The first 12 time slices are shown in the three tables below. Show the process that is running in the "Running Process" table. Also, show the runnable processes in the "Ready Queue" table, and the blocked processes in the "Wait Queue" table, at each time slice.

Start by scheduling Processes A, B, and C, as shown below. Assume that a process becomes runnable just before the arrival of a timer interrupt. Also, assume that if multiple processes become runnable at the same time, then they are added to the ready queue in the order A, B, and C.

Running Process

1	2	3	4	5	6	7	8	9	10	11	12
A	B	C	A	B	A	B	C	A	B	A	B

Ready Queue

head	B	C	A	B	A		C	A	B	A		C
	C	A					A					A
tail												

Wait Queue

head			B	C	C	C		B	C	C	C	
						B					B	
tail												

If there is a repeating pattern of running processes, circle the pattern in the "Running Process" table.

Roughly, what fraction of the CPU does each process take?

A = Based on repeating pattern, over the long term, 2/5

B = 2/5

C = 1/5

We also accepted 5/12, 5/12, and 2/12 as answers based on the first 12 time units.

Part (b) [4 MARKS] Show how the three processes are scheduled by a fixed priority scheduler. Assume Process `C` runs with the highest priority, Process `B` with medium priority, and Process `A` with the lowest priority. Start by scheduling Process `C`, as shown below.

Running Process

1	2	3	4	5	6	7	8	9	10	11	12
C	B	A	B	C	B	A	B	C	B	A	B

Ready Queue

head	B	A		A	A	A		A	A	A		A
	A											
tail												

Wait Queue

head		C	C	C	B	C	C	C	B	C	C	C
			B				B				B	
tail												

If there is a repeating pattern of running processes, circle the pattern in the "Running Process" table.

Roughly, what fraction of the CPU does each process take?

A = Based on repeating pattern, over the long term, 1/4

B = 1/2

C = 1/4

These same numbers also apply for the 12 time units.

Question 5. **Scheduling** [5 MARKS]

Consider the five threads shown in the table below. Their arrival times and their processing times are known and shown below.

Thread	Arrival Time	Processing Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

A thread scheduler runs these threads in the order shown below. The scheduler has a time slice of 1 time unit. You can assume that threads arrive just before the time slice expires.

Time	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Thread	1	1	2	2	3	2	4	4	3	2	4	5	3	2	4	1	5	3	2	4

Part (a) [1 MARK] Circle either a) or b)

a) Scheduler is non-preemptive

b) Scheduler is preemptive

Part (b) [3 MARKS] Briefly describe the policy that you think is being implemented by the scheduler.

longest remaining time first, with round robin scheduling for ties

Some answers suggested highest priority, and assumed that threads were blocking in between. With the assumption that threads are blocking at various points, the scheduler could have been implementing any scheduling policy (e.g., random), and so we did not give marks for this answer.
Some answers suggested longest "job" first, which is a non-preemptive policy. We did not deduct marks in this case.

Part (c) [1 MARK] Describe one problem that can occur with this type of scheduling.

Starve short jobs

Question 4. Trick the Scheduler [12 MARKS]

A (very) clever student in your class, called Trickster, has figured out a way to guess when timer interrupts are generated by hardware (the student knows that the interrupts are generated periodically, and has somehow determined this period). Trickster knows that the UG machines run the Unix feedback scheduler. Trickster runs a program called Sneaky that repeatedly performs a computation (uses the CPU), puts itself to sleep for the short duration during which the timer interrupt fires, and then wakes up to continue performing the computation.

Part (a) [4 MARKS] Clearly explain why the Sneaky program is able to trick the feedback scheduler. How does the program benefit from this trick? Hint: It will help to think about each of the steps in the scheduler.

The feedback scheduler updates usage on each timer interrupt for the currently running thread. Sneaky will appear to not be running at all, so its usage will remain 0. This will give Sneaky the highest priority among all processes and it will get to run most of the time.

Part (b) [4 MARKS] Clearly explain how you would change the feedback scheduler so that it can still achieve its goals. Explain, as precise as possible, why your change will fix the problem above.

The problem above is that we are updating the usage value periodically. Instead, if we kept track of the time that a process actually ran, the problem above would not happen. To do so, we can track the time when a process is run (on context switch) and when it stops running, i.e., it is put back in the ready/sleep queue (again on context switch).

Part (c) [4 MARKS] Outline a way that Trickster may have been able to figure out the period of the timer interrupt. You may assume that Trickster works in the night, when no one else is running their programs. non-hint: making the program sleep will likely not be helpful.

Trickster could write a program that read the computer clock (gettimeofday) in a tight loop. It would notice that the loop executes each iteration every few nanoseconds (assuming a GHz machine), but when the timer interrupt fires, the time difference between successive calls to gettimeofday would be much higher (since interrupt processing takes significant time). Plotting these results would show clear periodic behavior.