OPERATING SYSTEMS

Razi Uddin

Lecture # 12

SHORTEST-JOB-FIRST SCHEDULING

- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.
- If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie.
- The real difficulty with the SJF algorithm is in knowing the length of the next CPU request.
- For long term scheduling in a batch system, we can use as the length the process time limit that a user specifies when he submits the job.
- For short-term CPU scheduling, there is no way to length of the next CPU burst.
- One approach is to try to approximate SJF scheduling, by assuming that the next CPU burst will be similar in length to the previous ones.

SHORTEST-JOB-FIRST SCHEDULING

- The SJF algorithm may either be preemptive or non-preemptive.
- The choice arises when a new process arrives at the ready queue while a previous process is executing.
- The new process may have a shorter next CPU burst than what is left of the currently executing process.
- A preemptive SJF algorithm preempts the currently executing process, whereas a non-preemptive SJF algorithm will allow the currently running process to finish its CPU burst.
- Preemptive SJF scheduling is sometimes called shortest remaining-time-first scheduling.

PRIORITY SCHEDULING

- SJF is a special case of the general priority-scheduling algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
- Equal priority processes are scheduled in FCFS order.
- The SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst.
- The larger the CPU burst of a process, the lower its priority, and vice versa.

PRIORITY SCHEDULING

- Priority scheduling can either be preemptive or non-preemptive.
- When a process arrives at the ready queue, its priority is compared with the priority of the currently running process.
- A preemptive priority-scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.
- A non-preemptive priority-scheduling algorithm will simply put the new process at the head of the ready queue.

PRIORITY SCHEDULING

- A major problem with priority-scheduling algorithms is indefinite blocking (or **starvation**).
- A process that is ready to run but lacking the CPU can be considered blocked-waiting for the CPU.
- A priority-scheduling algorithm can leave some low-priority processes waiting indefinitely for the CPU.

SOLUTION (AGING)

- Aging is the solution to the problem of indefinite blockage of low-priority processes.
- It involves gradually increasing the priority of processes that wait in the system for a long time.
- For example, if priority numbers range from 0 (high priority) to 127 (low priority).
- We could periodically (say, every second) increase the priority of a waiting process by 1.
- This would result in every process in the system eventually getting the highest priority in a reasonably short amount of time and being scheduled to use the CPU

SOLUTION (AGING)

Another option is to combine round-robin and priority scheduling in such a way that the system executes the highest-priority process and runs processes with the same priority using round-robin scheduling.

WHY IS SJF OPTIMAL?

SJF is an optimal algorithm because it decreases the wait times for short processes much more than it increases the wait times for long processes.

ROUND-ROBIN SCHEDULING

- The round-robin (RR) scheduling algorithm is designed especially for time-sharing systems.
- It is similar to FCFS scheduling but preemption is added to switch between processes.
- A small unit of time called a time quantum (or time slice) is defined.
- The ready queue is treated as a circular queue.
- The CPU scheduler goes around the ready queue, allocating the CPU to each process for a time interval of up to 1-time quantum.

ROUND-ROBIN SCHEDULING

- To implement RR scheduling, we keep the ready queue as a FIFO queue of processes.
- New processes are added to the tail of the ready queue.
- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1-time quantum, and then dispatches the process.
- One of the two things will then happen.
- The process may have a CPU burst of less than 1-time quantum, in which case the process itself will release the CPU voluntarily.
- The scheduler will then proceed to the next process in the ready queue.
- Otherwise, if the CPU burst of the currently running process is longer than one-time quantum, the timer will go off and will cause an interrupt to the operating system.
- A context switch will happen, the current process will be put at the tail of the ready queue, and the newly scheduled process will be given to the CPU.

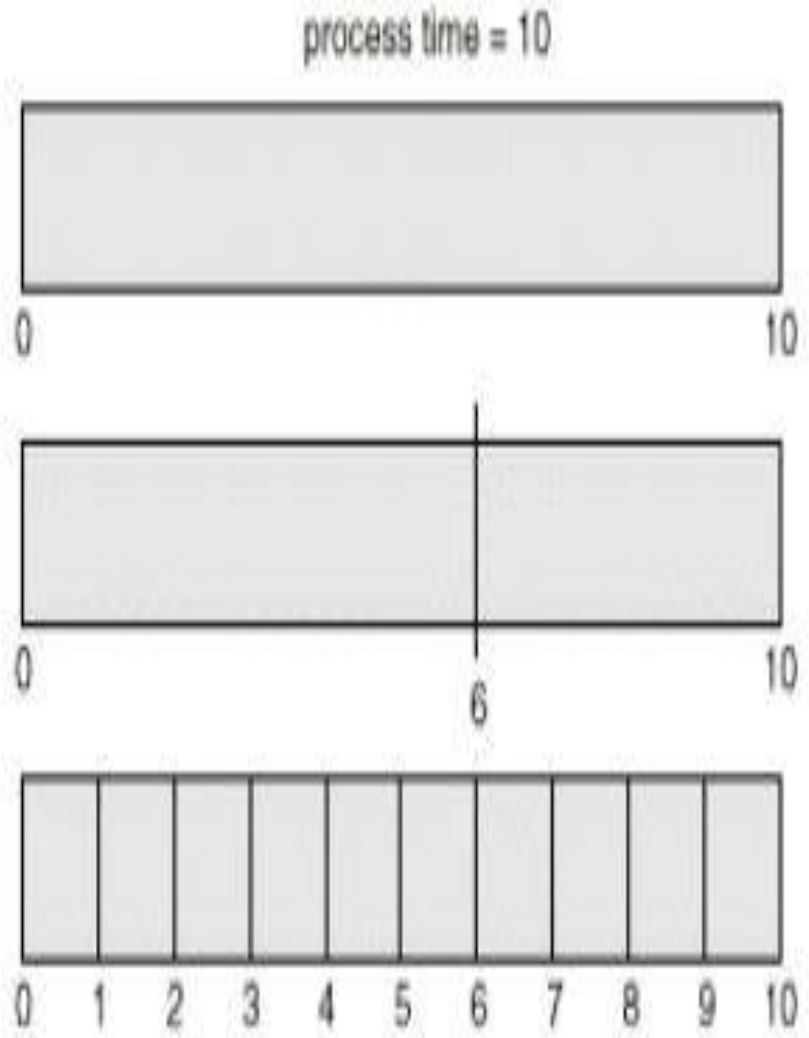
ROUND-ROBIN SCHEDULING

- The average waiting time under the RR policy however is often quite long.
- It is a preemptive scheduling algorithm.
- If there are n processes in the ready queue, context switch time is t_{cs} and the time quantum is q then each process gets 1/n of the CPU time in chunks of at most q time units.
- Each process must wait no longer than $(n-1)*(q+t_{cs})$ time units until its next time quantum.

ROUND-ROBIN SCHEDULING

- The performance of RR algorithm depends heavily on the size of the time quantum.
- If the time quantum is very large (infinite), the RR policy remains the same as the FCFS policy.
- If the time quantum is very small, the RR approach is called processor sharing and appears to the users as though each of n processes has its own processor running at 1/n the speed of the real processor (q must be large with respect to context switch, otherwise the overhead is too high).
- The drawback of small quantum is more frequent context switches.
- Since context switching is the cost of the algorithm and no useful work is done for any user process during context switching, the number of context switches should be minimized and the quantum should be chosen such that the ratio of a quantum to context switching is not less than 10:1 (i.e., context switching overhead should not be more than 10% of the time spent on doing useful work for a user process)

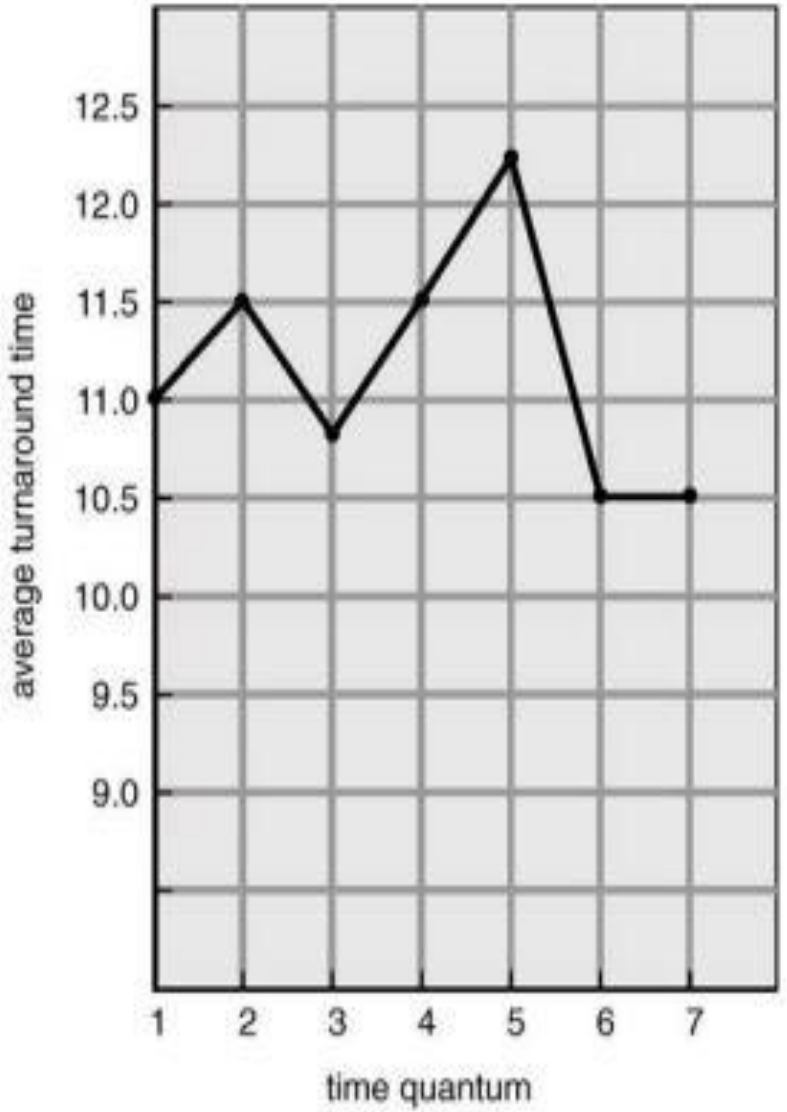
ROUND-ROBIN SCHEDULING



ROUND-ROBIN SCHEDULING

- The turnaround time of a process under round-robin also depends on the size of the time quantum.

- We can make a general statement that the round-robin algorithm gives the smallest average turnaround time when the quantum value is chosen such that most of the processes finish their next CPU bursts within the quantum.



process	time
P_1	6
P_2	3
P_3	1
P_4	7