# CHP#05: CPU Scheduling

**5.5 The following processes are being scheduled using a preemptive, roundrobin scheduling algorithm.**

a. The scheduling order of the processes using a Gantt chart is as follows:

```
0   P1  10  P1  20  P2  30  P3  40  P2  50  P3  60
P4  70  P4  80  P2  90  P3 100 P5 110 P6 120
```

b. The turnaround time for each process is:

- P1: `20 - 0 = 20`
- P2: `90 - 25 = 65`
- P3: `100 - 30 = 70`
- P4: `80 - 60 = 20`
- P5: `110 -100 =10`
- P6: `120 -105 =15`

c. The waiting time for each process is:

- P1: `20 -20 =0`
- P2: `65 -25 =40`
- P3: `70 -25 =45`
- P4: `20 -15 =5`
- P5: `10 -10 =0`
- P6: `15 -10 =5`

d. The CPU utilization rate is `(20+25+25+15+10+10)/120 *100% =95.83%`

**5.6 What advantage is there in having different time-quantum sizes at different levels of a multilevel queueing system?**

1. Allows the system to prioritize certain types of processes over others.
2. Ensures that interactive or time-sensitive processes receive more frequent CPU time and have lower response times.
3. Allows for more efficient processing of batch or compute-intensive jobs.
4. Helps prevent starvation of lower-priority processes by giving them a longer time-quantum to make more progress before being preempted.
5. Provides more fine-grained control over the allocation of CPU resources.
6. Can improve the performance and responsiveness of the system.

**5.7 Many CPU-scheduling algorithms are parameterized. For example, the RR algorithm requires a parameter to indicate the time slice. Multilevel feedback queues require parameters to define the number of queues, the scheduling algorithms for each queue, the criteria used to move processes between queues, and so on. These algorithms are thus really sets of algorithms (for example, the**

**set of RR algorithms for all time slices, and so on). One set of algorithms may include another (for example, the FCFS algorithm is the RR algorithm with an infinite time quantum). What (if any) relation holds between the following pairs of algorithm sets?**

**a. Priority and SJF**

**b. Multilevel feedback queues and FCFS**

**c. Priority and FCFS**

**d. RR and SJF**

a. Priority and SJF: Shortest Job First (SJF) can be seen as a special case of priority scheduling where the priority of a process is determined by the length of its CPU burst. In other words, SJF is a priority scheduling algorithm where the priority is the inverse of the next CPU burst.

b. Multilevel feedback queues and FCFS: Multilevel feedback queues can be configured to behave like First-Come-First-Serve (FCFS) scheduling by setting up a single queue with FCFS scheduling and not allowing processes to move between queues.

c. Priority and FCFS: There is no direct relation between priority scheduling and FCFS scheduling. However, if all processes have the same priority in a priority scheduling algorithm, it will behave like FCFS scheduling.

d. RR and SJF: There is no direct relation between Round-Robin (RR) scheduling and SJF scheduling. However, if the time quantum in RR scheduling is set to infinity, it will behave like FCFS scheduling which is similar to SJF when all processes have the same length of CPU burst.

**5.11 Of these two types of programs: a. I/O-bound b. CPU-bound which is more likely to have voluntary context switches, and which is more likely to have nonvoluntary context switches? Explain your answer**

a. I/O-bound programs are more likely to have voluntary context switches. This is because I/O-bound programs spend a significant amount of time waiting for I/O operations to complete. During this time, the program may voluntarily yield the CPU to allow other processes to run while it waits for the I/O operation to complete.

b. CPU-bound programs are more likely to have nonvoluntary context switches. This is because CPU-bound programs spend most of their time using the CPU and may not voluntarily yield the CPU. As a result, the operating system may need to preempt the program and perform a nonvoluntary context switch to allow other processes to run.

**5.12 Discuss how the following pairs of scheduling criteria conflict in certain settings.**

**a. CPU utilization and response time**

**b. Average turnaround time and maximum waiting time**

**c. I/O device utilization and CPU utilization**

a. CPU utilization and response time: Maximizing CPU utilization can lead to longer response times for interactive processes. This is because keeping the CPU busy with batch or compute-intensive processes can result in interactive processes having to wait longer for their turn to use the CPU.

b. Average turnaround time and maximum waiting time: Minimizing average turnaround time can lead to longer maximum waiting times. This is because scheduling algorithms that prioritize shorter jobs can result in longer jobs being delayed and experiencing longer waiting times.

c. I/O device utilization and CPU utilization: Maximizing I/O device utilization can lead to lower CPU utilization. This is because keeping I/O devices busy with I/O-bound processes can result in the CPU being idle while it waits for I/O operations to complete.

**5.20 Which of the following scheduling algorithms could result in starvation? a. First-come, first-served b. Shortest job first c. Round robin d. Priority**

Starvation occurs when a process is perpetually denied access to the CPU and is unable to make progress. Of the scheduling algorithms listed, Shortest Job First (SJF) and Priority scheduling could result in starvation.

a. First-come, first-served (FCFS) scheduling does not result in starvation because processes are scheduled in the order they arrive, so every process will eventually get a chance to run.

b. Shortest Job First (SJF) scheduling could result in starvation if there are always shorter jobs arriving. In this case, longer jobs may be perpetually delayed and never get a chance to run.

c. Round Robin (RR) scheduling does not result in starvation because each process is given a time quantum to run before being preempted. This ensures that every process will eventually get a chance to run.

d. Priority scheduling could result in starvation if there are always higher-priority processes arriving. In this case, lower-priority processes may be perpetually delayed and never get a chance to run

**5.22 Consider a system running ten I/O-bound tasks and one CPU-bound task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context-switching overhead is 0.1 millisecond and that all processes are long-running tasks. Describe the CPU utilization for a round-robin scheduler when:**

**a. The time quantum is 1 millisecond**

**b. The time quantum is 10 milliseconds**

a. The time quantum is 1 millisecond: In this case, each I/O-bound task will issue an I/O operation after using its 1 millisecond time quantum. The I/O operation will take 10 milliseconds to complete, during which time the CPU can switch to another task. However, since there are ten I/O-bound tasks, by the time the first I/O-bound task completes its I/O operation, the other nine I/O-bound tasks will have used their time quantum and issued their own I/O operations. This means that the CPU will have to wait for the first I/O-bound task to complete its I/O operation before it can switch back to it.
The total time to complete one round of execution for all ten I/O-bound tasks and one CPU-bound task is `10*(1+0.1) + 1 + 0.1 = 12 milliseconds`. During this time, the CPU is busy for `10*1 + 1 = 11 milliseconds` and idle for `1 millisecond` while waiting for the first I/O-bound task to complete its I/O operation. Therefore, the CPU utilization in this case is `11/12 * 100% = 91.67%`.

b. The time quantum is 10 milliseconds: In this case, each I/O-bound task will issue an I/O operation after using its 10 millisecond time quantum. The I/O operation will take 10 milliseconds to complete, during which time the CPU can switch to another task.
The total time to complete one round of execution for all ten I/O-bound tasks and one CPU-bound task is `10*(10+0.1) + 10 + 0.1 = 111 milliseconds`. During this time, the CPU is busy for `10*10 + 10 = 110 milliseconds` and idle for `1 millisecond` while waiting for the first I/O-bound task to complete its I/O operation. Therefore, the CPU utilization in this case is `110/111 * 100% = 99.10%`.

**5.25 Explain the how the following scheduling algorithms discriminate either in favor of or against short processes: a. FCFS b. RR c. Multilevel feedback queues**

a. FCFS: First-Come-First-Serve (FCFS) scheduling does not discriminate in favor of or against short processes. Processes are scheduled in the order they arrive, regardless of their length.

b. RR: Round-Robin (RR) scheduling does not directly discriminate in favor of or against short processes. However, since each process is given a time quantum to run before being preempted, shorter processes may complete within their time quantum and not need to be preempted. This can result in shorter processes having lower response times and turnaround times compared to longer processes.

c. Multilevel feedback queues: Multilevel feedback queues can be configured to discriminate in favor of short processes. This can be achieved by setting up multiple queues with different priorities and scheduling algorithms. For example, a higher-priority queue could use Shortest Job First (SJF) scheduling to prioritize shorter processes. Processes that do not complete within their time quantum could be moved to a lower-priority queue with a longer time quantum or a different scheduling algorithm.

**5.36 Explain why interrupt and dispatch latency times must be bounded in a hard real-time system.**

Interrupt latency is the time it takes for the system to respond to an interrupt and start executing the interrupt handler. Dispatch latency is the time it takes for the system to stop executing one task and start executing another. These latencies must be bounded in a hard real-time system to ensure that tasks can be executed in a timely manner and meet their deadlines.

If interrupt and dispatch latency times are not bounded, they can become unpredictable and result in tasks missing their deadlines.

**#Global Addition**

**5.1. List four design issues for which the concept of concurrency is relevant.**

1. Resource allocation: managing access to shared resources among concurrent processes.
2. Synchronization: coordinating the execution of concurrent processes to ensure correct behavior.
3. Deadlock prevention: avoiding situations where concurrent processes are blocked and unable to make progress.
4. Process scheduling: determining the order in which concurrent processes are executed.

**5.2. What are three contexts in which concurrency arises?**

1. Multiple processes executing on a single CPU.
2. Multiple threads executing within a single process.
3. Distributed systems where multiple processes execute on different machines.

**5.3. What is a race condition?**

A race condition is a situation where the behavior of a system depends on the relative timing of events, such as the order in which concurrent processes access shared data. Race conditions can result in unpredictable and undesirable behavior.

**5.4. List three degrees of awareness between processes and briefly define each.**

1. Unaware: Processes are unaware of each other and do not communicate or coordinate their actions.
2. Indirectly aware: Processes communicate indirectly through shared resources or message passing but do not coordinate their actions directly.
3. Directly aware: Processes communicate and coordinate their actions directly through message passing or other synchronization mechanisms.

**5.5. What is the distinction between competing processes and cooperating processes?**

The distinction between competing processes and cooperating processes is that competing processes contend for shared resources and may interfere with each other's execution, while cooperating processes work together towards a common goal and coordinate their actions to achieve that goal.

**5.6. List the three control problems associated with competing processes, and briefly define each.**

1. Mutual exclusion: ensuring that only one process can access a shared resource at a time.
2. Deadlock: avoiding situations where processes are blocked and unable to make progress because they are waiting for each other to release resources.
3. Starvation: preventing situations where a process is perpetually denied access to a shared resource

**5.7. What is starvation with respect to concurrency control by mutual exclusion?**

Starvation with respect to concurrency control by mutual exclusion refers to a situation where a process is perpetually denied access to a shared resource because other processes are always accessing it.

**5.8. What operations can be performed on a semaphore?**

The operations that can be performed on a semaphore are wait (also known as P or acquire) and signal (also known as V or release). The wait operation decrements the semaphore value and blocks the calling process if the value becomes negative. The signal operation increments the semaphore value and unblocks a waiting process if one exists.

**5.9. What is the difference between binary and general semaphores?**

The difference between binary and general semaphores is that binary semaphores can only have values of 0 or 1, while general semaphores can have any non-negative integer value.

**5.10. What is the key difference between a mutex and a binary semaphore?**

The key difference between a mutex and a binary semaphore is that a mutex has an owner, meaning that only the process that acquired the mutex can release it, while a binary semaphore has no owner, meaning that any process can signal it.

**5.11. Which characteristics of monitors mark them as high-level synchronization tools?**

The characteristics of monitors that mark them as high-level synchronization tools are their encapsulation of shared data and synchronization mechanisms within a single module, their use of condition variables to allow processes to wait for specific conditions, and their support for structured programming using procedures.

**5.12. Compare direct and indirect addressing with respect to message passing.**

Direct addressing with respect to message passing refers to specifying the recipient of a message by its unique identifier, while indirect addressing refers to specifying the recipient by its role or by using an intermediary such as a mailbox or port.

**5.13. What conditions are generally associated with the readers/writers problem?**

The conditions generally associated with the readers/writer's problem are:

1. Any number of readers can read from the shared resource simultaneously.
2. Only one writer can write to the shared resource at any given time.
3. If a writer is writing to the resource, no reader may read it.