Contents

Q1.

- Short-term scheduling, also known as CPU scheduling, is the process of determining which process should be executed by the CPU next. It selects processes from the ready queue that are in main memory and allocates CPU to one of them. It is also known as CPU scheduler and invoked frequently, such as in response to clock ticks, I/O interrupts, system calls, signal sending and receiving, and activation of interactive programs.
- Medium-term scheduling, also known as swapping or memory management, is the process of moving processes in and out of main memory. It brings suspended or swapped-out processes into a pool of ready processes. The process may be suspended due to an I/O request or a system call, and it is removed from the main memory and stored in a swapped queue in the secondary memory to create space for another process. The medium-term scheduler plans the CPU scheduling for processes that have been waiting for the completion of another process or an I/O task.
- Long-term scheduling, also known as job scheduling, is the process of determining which processes should be admitted to the system for execution. It works with the batch queue and selects the next batch job to be executed, which are resource-intensive and have low priority. It selects the process from secondary storage, such as a disk and loads them into memory for execution. It is also known as job scheduler, works on a longer time frame, and aims to select a good process mix of I/O-bound and CPU-bound processes, where I/O-bound processes spend most of their time in I/O and CPU-bound processes spend most of their time in computation.

Q2.

A context switch is the process of storing the current state of a process and loading the state of another process. When a kernel performs a context switch, it saves the state of the current process, including the contents of the CPU registers and memory, and loads the state of the new process. This process can be time-consuming and can negatively impact system performance if done too frequently.

The kernel here also updates its data structures to reflect the change in process state. This includes updating the process control block (PCB) of the current process and the ready queue to reflect the new state of the system.

Q3.

Ordinary pipes are more suitable when two processes need to communicate with each other but are not part of the same parent process tree. Ordinary pipes are also used to provide unidirectional communication.

Named pipes are more suitable when the data passing between the processes is large and needs to be passed multiple times. Named pipes are typically used when the two processes are part of the same parent process tree and need to communicate with each other on the same machine. Named pipes can also be used to provide bidirectional communication.

Q4.

a)

Synchronous communication means that the sender and receiver process must be communicating at the same time, waiting for each other to finish the communication. It will block the sender process until the receiver process acknowledges the receipt of the data. The advantage of this type of communication is the simplicity of the programming, but the disadvantage is that it might cause the process to be blocked for an indefinite period if the receiver process is not responsive.

Asynchronous communication, On the other hand, allows the sender process to continue its execution after sending the data. The receiver process can pick up the data whenever it is ready. This type of communication is more suitable for real-time systems where the sender process cannot afford to wait for the receiver process. The advantage of this type of communication is that it doesn't block the sender process and provides a better throughput, but the disadvantage is that it might be more complex to program.

b)

Automatic buffering refers to the system automatically buffering data being sent or received by a process without the need for explicit intervention from the programmer. The advantage of automatic buffering is that it can be less error-prone and simpler to implement, but it can also be less efficient if the system does not correctly anticipate the needs of the program.

Explicit buffering, on the other hand, requires the programmer to explicitly set up and manage the buffer. This allows for more fine-tuned control over the buffering and can lead to better performance, but it also requires more effort and can be more prone to bugs if not implemented properly.

c)

Send by copy refers to passing a copy of the data being sent to the receiving process. The advantage of this is that it ensures that the sending process cannot modify the data after it has been sent. The disadvantage is that it can be more expensive in terms of memory and time, especially for large data sets.

Send by reference, on the other hand, refers to passing a reference to the data, rather than a copy. This allows the sending process to continue to modify the data after it has been sent, but it also means that the receiving process must be careful to not modify the data in a way that could affect the sending process.

d)

Fixed-sized messages refer to messages that have a predefined, fixed size. The advantage of this is that it makes the communication more predictable and can simplify the implementation, but it also means that the process must be able to handle the largest possible message size, even if it does not always need to.

Variable-sized messages, on the other hand, allow messages to be of varying sizes. This can be more flexible and can allow for more efficient communication, but it can also make the implementation more complex.

Q5.

If you're running a basic calculation program that takes the same amount of time no matter what, using multiple threads might slow it down. The reason is that setting up multiple threads takes more time and effort than the task itself. This is especially true for things like simple math on a list of numbers or allocating memory to a set of data variables. Even though these tasks are quick, using multiple threads to do them takes longer than just using one thread.

Another example is a program that performs many small I/O operations, such as reading small amounts of data from a file. In this case, the overhead of managing multiple threads would outweigh the benefits of parallelism.

Q6.

Yes, This is because a multiprocessor system has multiple CPUs available to execute threads, allowing for parallel execution of multiple threads. On a single processor system, the CPU must constantly switch between executing different threads, which can lead to decreased performance. However, it should be noted that just creating threads doesn't automatically guarantee a better performance, because of the overheads of creating, switching, and scheduling threads. Furthermore, the benefits of multithreading can be reduced if the threads are not designed to take advantage of multiple processors or if the threads are not properly synchronized to avoid contention for shared resources.

Q7.

a) For input and output, **only one thread**. The reason for this is that both input and output are performed at specific times during the program's execution, and both involve only one file. There is no need to create multiple threads for input and output as it would only lead to increased overhead without any significant performance benefits.

b) For the CPU-intensive portion of the application, **four threads**, one for each processor available. This will allow the application to take full advantage of the multiprocessor system by allowing the threads to be executed in parallel on different processors.

Q8.

Because I/O-bound programs have different resource requirements and behavior than CPU-bound programs. I/O-bound programs spend a significant amount of time waiting for I/O operations to complete, while CPU-bound programs spend a significant amount of time performing computations. The scheduler can use this information to better allocate resources and improve performance.

Q9.

The scheduler can assign more lottery tickets to higher-priority threads than to lower-priority threads. This will increase the probability that a higher-priority thread's lottery ticket will be chosen during the lottery, thus giving the higher-priority thread more CPU time. Additionally, the scheduler could assign different ranges of lottery tickets to different priority levels to give more chance for higher priority levels to win a lottery.

Q10.

a) The time quantum is 1ms, the CPU Utilization is calculated by dividing the time the CPU is utilized, 1ms, by the total time of a cycle, 1ms + 0.1ms (cost of context switching) and then multiplied by 100 to get the percentage. **CPU Utilization = (1 / (1 + 0.1)) * 100 = 90.9%**

b) The time quantum is 10ms, the CPU Utilization is calculated by dividing the time the CPU is utilized, 20ms (2ms (each I/O bound task) + 18ms (CPU bound task)) by the total time of a cycle,

21.1ms (10.1ms (cost of context switching) + (1.1ms*10 (I/O bound tasks))) and then multiplied by 100 to get the percentage. **CPU Utilization = (20 / (21.1)) * 100 = 94.78%**