| | | |
|---|---|---|
| 12. | Changes to the parent process do not affect child processes. | Since all threads of the same process share address space and other resources, any changes to the main thread may affect the behavior of the other threads of the process. |

## Multi-Threading Models

It is a process of multiple threads executed at same time.

Many operating systems support kernel thread and user thread in a combined way. Example of such a system is Solaris. Multi threading models are of three types.

- Many to many models.
- Many to one model.
- one to one model.

**Many to Many Model**

In this model, we have multiple user threads multiplexed to the same or lesser number of kernel level threads. Number of kernel level threads are specific to the machine, the advantage of this model is if a user thread is blocked we can schedule other user threads to another kernel thread. Thus, System doesn't block if a particular thread is blocked.

**Many to One Model**
In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks. As we have only one kernel thread and only one user thread can access the kernel at a time, so multiple threads are not able access the multiprocessor at the same time.

The thread management is done on the user level so it is more efficient.

**One to One Model**
In this model, one to one relationship between kernel and user thread. In this model multiple threads can run on multiple processors. Problem with this model is that creating a user thread requires the corresponding kernel thread.

As each user thread is connected to a different kernel , if any user thread makes a blocking system call, the other user threads won't be blocked.

## CPU SCHEDULING

CPU scheduling is the basis of Multiprogrammed operating systems. By switching the CPU among processes, the operating system can make the computer more productive.

- In a single-processor system, only one process can run at a time. Others must wait until the CPU is free and can be rescheduled.

- The CPU will sit idle and wait for a process that needs an I/O operation to complete. If the I/O operation completes then only the CPU will start executing the process. A lot of CPU time has been wasted with this procedure.
- The objective of multiprogramming is to have some process running at all times to maximize CPU utilization.
- When several processes are in main memory, if one process is waiting for I/O then the operating system takes the CPU away from that process and gives the CPU to another process. Hence there will be no wastage of CPU time.

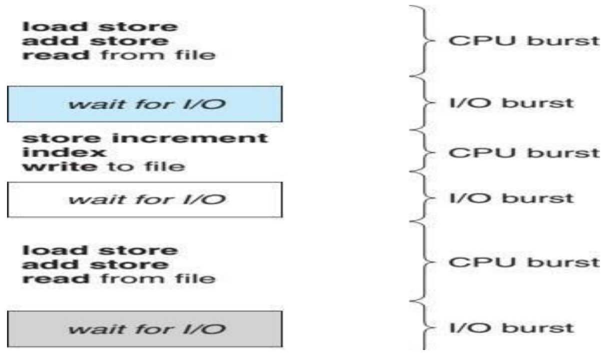
**Concepts of CPU Scheduling**

1. CPU–I/O Burst Cycle
2. CPU Scheduler
3. Preemptive Scheduling
4. Dispatcher

**CPU–I/O Burst Cycle**

Process execution consists of a **cycle** of CPU execution and I/O wait.

- Process execution begins with a **CPU burst**. That is followed by an **I/O burst.** Processes alternate between these two states.
- The final CPU burst ends with a system request to terminate execution.
- Hence the **First cycle** and **Last cycle** of execution must be CPU burst.



**CPU Scheduler**

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the **Short-Term Scheduler** or **CPU scheduler**.

**Preemptive Scheduling**

CPU-scheduling decisions may take place under the following four cases:

1. When a process switches from the running state to the waiting state.
   Example: as the result of an I/O request or an invocation of wait( ) for the termination of a child process.
2. When a process switches from the running state to the ready state. Example: when an interrupt occurs
3. When a process switches from the waiting state to the ready state.
   Example: at completion of I/O.
4. When a process terminates.

For situations 2 and 4 are considered as **Preemptive scheduling** situations. Mach OS X, WINDOWS 95 and all subsequent versions of WINDOWS are using Preemptive scheduling.

## Dispatcher

The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. Dispatcher function involves:

1. Switching context
2. Switching to user mode
3. Jumping to the proper location in the user program to restart that program.

The dispatcher should be as fast as possible, since it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another process running is known as the **Dispatch Latency**.

## SCHEDULING CRITERIA

Different CPU-scheduling algorithms have different properties and the choice of a particular algorithm may favor one class of processes over another.

Many criteria have been suggested for comparing CPU-scheduling algorithms:

- **CPU utilization:** CPU must be kept as busy as possible. CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 to 90 percent.
- **Throughput:** The number of processes that are completed per time unit.
- **Turn-Around Time:** It is the interval from the time of submission of a process to the time of completion. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.
- **Waiting time:** It is the amount of time that a process spends waiting in the ready queue.
- **Response time:** It is the time from the submission of a request until the first response is produced. Interactive systems use response time as its measure.

**Note:** It is desirable to maximize CPU utilization and Throughput and to minimize Turn-Around Time, Waiting time and Response time.

## CPU SCHEDULING ALGORITHMS

CPU scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated to the CPU. Different CPU-scheduling algorithms are:

1. First-Come, First-Served Scheduling (FCFS)
2. Shortest-Job-First Scheduling (SJF)
3. Priority Scheduling
4. Round Robin Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

**Gantt Chart** is a bar chart that is used to illustrate a particular schedule including the start and finish times of each of the participating processes.

### First-Come, First-Served Scheduling (FCFS)

In FCFS, the process that requests the CPU first is allocated the CPU first.

- FCFS scheduling algorithm is Non-preemptive.
- Once the CPU has been allocated to a process, it keeps the CPU until it releases the CPU.
- FCFS can be implemented by using FIFO queues.
- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue.