TASK SCHEDULING :

The first job of the operating system is to determine the process that runs next. The work of choosing the order of running processes is known as scheduling.

The operating system considers a process to be in one of three basic **scheduling states: waiting, ready**, or **executing.** There is at most one process executing on the CPU at any time. (If there is no useful work to be done, an idling process may be used to perform a null operation.) Any process that could execute is in the ready state; the operating system chooses among the ready processes to select the next executing process. A process may not, however, always be ready to run. For instance, a process may be waiting for data from an I/O device or another process, or it may be set to run from a timer that has not yet expired. Such processes are in the waiting state. Figure 5.2 shows the possible transitions between states available to a process. A process goes into the waiting state when it needs data that it has not yet received or when it has finished all its work for the current period. A process goes into the ready state when it receives its required data and when it enters a new period. A process can go into the executing state only when it has all its data, is ready to run, and the scheduler selects the process as the next process to run.

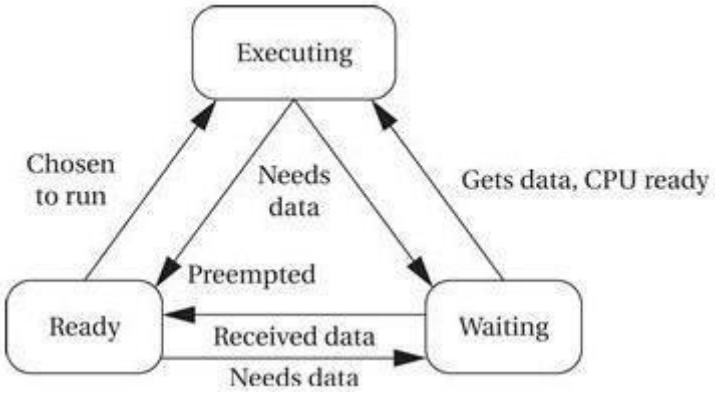


FIGURE 5.2 SCHEDULING STATES OF A PROCESS

A **scheduling policy** defines how processes are selected for promotion from the ready state to the running state. Every multitasking operating system implements some type of scheduling policy. Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, but it also has a profound influence on the CPU horsepower required to implement the system's functionality.

Scheduling policies vary widely in the generality of the timing requirements they can handle and the efficiency with which they use the CPU. Utilization is one of the key metrics in evaluating a scheduling policy. We will see that some types of timing requirements for a set of processes imply that we cannot utilize 100% of the CPU's execution time on useful work, even ignoring context switching overhead. However, some scheduling policies can deliver higher CPU utilizations than others, even for the same timing requirements. The best policy depends on the required timing characteristics of the processes being scheduled.

PRIORITY BASED SCHEDULING

The operating system's fundamental job is to allocate resources in the computing system among programs that request them. Naturally, the CPU is the scarcest resource, so scheduling the CPU is the operating system's most important job. In this section, we consider the structure of operating systems, how they schedule processes to meet performance requirements, and how they provide services beyond CPU scheduling.

ROUND-ROBIN SCHEDULING

A common scheduling algorithm in general-purpose operating systems is **round-robin**. All the processes are kept on a list and scheduled one after the other. This is generally combined with preemption so that one process does not grab all the CPU time. Round-robin scheduling provides a form of fairness in that all processes get a chance to execute.

However, it does not guarantee the completion time of any task; as the number of processes increases, the response time of all the processes increases. Real-time systems, in contrast, require their notion of fairness to include timeliness and satisfaction of deadlines.

PROCESS PRIORITIES

A common way to choose the next executing process in an RTOS is based on process priorities. Each process is assigned a priority, an integer-valued number. The next process to be chosen to execute is the process in the set of ready processes that has the highest-valued priority. Example shows how priorities can be used to schedule processes.

EXAMPLE:

For this example, we will adopt the following simple rules:

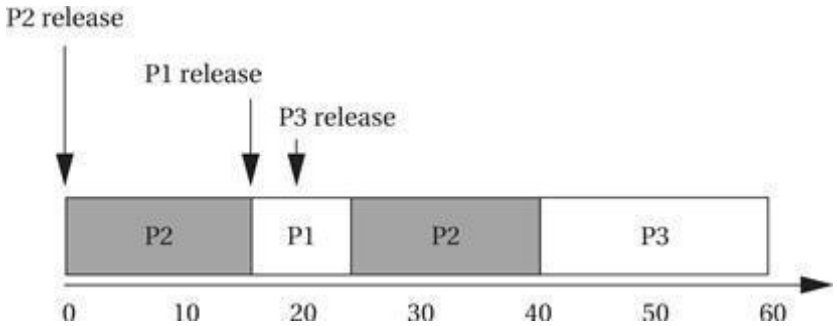
- Each process has a fixed priority that does not vary during the course of execution. (More sophisticated scheduling schemes do, in fact, change the priorities of processes to control what happens next.)
- The ready process with the highest priority (with 1 as the highest priority of all) is selected for execution.
- A process continues execution until it completes or it is preempted by a higher-priority process.

Let's define a simple system with three processes as seen below.

Process	Priority	Execution time
P1	1	10
P2	2	30
P3	3	20

In addition to describing the properties of the processes in general, we need to know the environmental setup. We assume that P2 is ready to run when the system is started, P1's data arrive at time 15, and P3's data arrive at time 18.

Once we know the process properties and the environment, we can use the priorities to determine which process is running throughout the complete execution of the system.



When the system begins execution, P2 is the only ready process, so it is selected for execution. At time 15, P1 becomes ready; it pre-empts P2 and begins execution because it has a higher priority. Because P1 is the highest-priority process in the system, it is guaranteed to execute until it finishes. P3's data arrive at time 18, but it cannot preempt P1. Even when P1 finishes, P3 is not allowed to run. P2 is still ready and has higher priority than P3. Only after both P1 and P2 finish can P3 execute.