What is Concurrent Programming?
Computer Time

time (nanoseconds) →
Human Time

time (seconds) →

0  100  200  300  400  500
Primitive, one-job-at-a-time system
Multiprogramming

• An operating system that can switch back and forth between processes to keep the CPU busy is called a multiprogramming system.

• It maintains a queue of process control blocks (PCBs).
Asynchronous interrupts

- Time outs
- I/O completions
A multiprocessing system
Observation

It is impossible for the programmer to predict the statements in the program where the process will be interrupted by the operating system.
A process is a program during execution. The state of the process is specified by:

- the program listing
- the values of all the variables
- the next instruction to execute
- program counter (PC)
8.3 Concurrent Processes

If a running process requests some input from a keyboard, in the fraction of a second that it takes the user to respond, the CPU can execute hundreds of thousands of instructions for another process. Even if the process requests input from a disk file, which is much faster than keyboard input, the CPU could still execute thousands of instructions while waiting for the information to come from the disk.

To keep from wasting CPU time, the operating system can suspend the process that makes an I/O request if it appears that the process will need to wait for the I/O to complete. It can temporarily assign the CPU to a second process with the understanding that when the I/O does complete, the first process may immediately get the CPU back. Because the second process cannot predict when the I/O device will complete the I/O operation for the first process, it cannot know when the operating system might interrupt it to give the CPU back to the first process.

An operating system that can switch back and forth between processes to keep the CPU busy is called a multiprogramming system. To implement multiprogramming, the hardware must provide connections for the I/O devices to send interrupt signals to the CPU when the devices complete their I/O operations.

Processes in the Operating System

One purpose of an operating system is to allocate the resources of the system efficiently. A multiprogramming time-sharing system allocates CPU time among the jobs in the system. The objective is to keep the CPU as busy as possible executing user jobs instead of being idle waiting for I/O. The operating system tries to be fair in scheduling CPU time so that all the jobs will be completed in a reasonable time.

At any given time, the operating system must maintain many suspended processes that are waiting their turn for CPU time. It maintains all these processes by allocating a separate PCB for each one, similar to the PCB the interrupt handler maintains in the Pep/8 system. A common practice is to link the PCBs together with pointers in a linked list called a queue. Figure 8.18 shows a queue of PCBs.

Each PCB includes copies of all the CPU register values at the time of the process's most recent interrupt. The register set must include a copy of the program counter so the process can continue executing from where it was when the interrupt occurred.

Multiprogramming

Figure 8.18

A queue of process control blocks.

<table>
<thead>
<tr>
<th>Process ID</th>
<th>CPU time</th>
<th>PC</th>
<th>SP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 8.18
The PCB contains additional information to help the operating system schedule the CPU. An example is a unique process identification number assigned by the system, labeled Process ID in Figure 8.18, that serves to reference the process. Suppose a user wants to terminate a process before it completes execution normally, and he knows the ID number is 782. He could issue a \texttt{KILL(782)} command that would cause the operating system to search through the queue of PCBs, find the PCB with ID 782, remove it from the queue, and deallocate it.

Another example of information stored in the PCB is a record of the total amount of CPU time used so far by the suspended process. If the CPU becomes available and the operating system must decide which of several suspended processes gets the CPU, it can use the recorded time to make a fair decision.

As a job progresses through the system toward completion, it passes through several states, as Figure 8.19 shows. The figure is in the form of a state transition diagram and is another example of a finite state machine. Each transition is labeled with the event that causes the change of state.

When a user submits a job for processing, the operating system creates a process for it by allocating a new PCB and attaching it to a queue of processes that are waiting for CPU time. It loads the program into main memory and sets the copy of \texttt{PC} in the PCB to the address of the first instruction of the process. That puts the job in the ready state.

Eventually, the operating system should select the job to receive some processing time. It sets the alarm clock to generate an interrupt after a quantum of time and puts the copies of the registers from the PCB into the CPU. That puts the job in the running state.

While in the running state, three things can happen: (1) The running process may time out if it is still executing when the alarm clock interrupts. If so, the operating system attaches the process's PCB to the ready queue, which puts it back in the ready state. (2) The process may complete its execution normally, in which case the last instruction it executes is an \texttt{SVC} to request that the operating system terminate it. (3) The process may need some input, in which case it executes an \texttt{SVC} for the request. The operating system would transfer the request to the appropriate I/O device and put the PCB in another queue of processes that are waiting for their I/O operations to complete. That puts the process in the waiting-for-I/O state.
Multiprocessing

- A computer system with more than one physical CPU
- Also maintains a queue of PCBs, but more than one process can be running at the same time
Input device → CPU 1 → CPU 2 → Main memory → Output device

Bus

Figure 8.20
The Concurrency Theorem

Multiprogramming and multiprocessing are logically equivalent.