Multiprogramming Systems

- Provide efficient resource utilization in a noninteractive environment
- Multiple user programs are in memory at any time
- Uses DMA mode of I/O
  - Can perform I/O operations of some program(s) while using the CPU to execute some other program
    - Makes efficient use of both the CPU and I/O devices
- Turnaround time of a program is the appropriate measure of user service in these systems

---

**Figure 3.3** Operation of a multiprogramming system: (a) program1 is in execution while program2 is performing an I/O operation; (b) program2 initiates an I/O operation, program3 is scheduled; (c) program1's I/O operation completes and it is scheduled.
Multiprogramming Systems (continued)

Table 3.4 Architectural Support for Multiprogramming

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA</td>
<td>The CPU initiates an I/O operation when an I/O instruction is executed. The DMA implements the data transfer involved in the I/O operation without involving the CPU and raises an I/O interrupt when the data transfer completes.</td>
</tr>
<tr>
<td>Memory protection</td>
<td>A program can access only the part of memory defined by contents of the base register and size register.</td>
</tr>
<tr>
<td>Kernel and user modes of CPU</td>
<td>Certain instructions, called privileged instructions, can be performed only when the CPU is in the kernel mode. A program interrupt is raised if a program tries to execute a privileged instruction when the CPU is in the user mode.</td>
</tr>
</tbody>
</table>

Multiprogramming Systems (continued)

- An appropriate measure of performance of a multiprogramming OS is *throughput*
  - Ratio of the number of programs processed and the total time taken to process them

- OS keeps enough programs in memory at all times, so that CPU and I/O devices are not idle
  - *Degree of multiprogramming*: number of programs in memory
  - Uses an appropriate *program mix of CPU-bound programs* and *I/O-bound programs*
  - Assigns appropriate priorities to CPU-bound and I/O-bound programs
Priority of Programs

Table 3.5 Techniques of Multiprogramming

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appropriate program mix</td>
<td>The kernel keeps a mix of CPU-bound and I/O-bound programs in memory.</td>
</tr>
<tr>
<td></td>
<td>• A CPU-bound program is a program involving a lot of computation and very little I/O. It uses the CPU in long bursts—that is, it uses the CPU for a long time before starting an I/O operation.</td>
</tr>
<tr>
<td></td>
<td>• An I/O-bound program involves very little computation and a lot of I/O. It uses the CPU in small bursts.</td>
</tr>
<tr>
<td>Priority-based preemptive scheduling</td>
<td>Every program is assigned a priority. The CPU is always allocated to the highest-priority program that wishes to use it. A low-priority program executing on the CPU is preempted if a higher-priority program wishes to use the CPU.</td>
</tr>
</tbody>
</table>

Definition 3.4 Priority

A tie-breaking criterion under which a scheduler decides which request should be scheduled when many requests await service.

Priority of Programs (continued)

In multiprogramming environments, an I/O-bound program should have a higher priority than a CPU-bound program.

Figure 3.4 Timing chart when I/O-bound program has higher priority.
Performance of Multiprogramming systems

• How to improve system throughput?

| Table 3.6 Effect of Increasing the Degree of Multiprogramming |
|-----------------|---------------------------|
| Action          | Effect                    |
| Add a CPU-bound program | A CPU-bound program (say, prog) can be introduced to utilize some of the CPU time that was wasted in Example 3.1 (e.g., the intervals $t_5-t_7$ and $t_8-t_9$). prog would have the lowest priority. Hence its presence would not affect the progress of prog A and prog B. |
| Add an I/O-bound program | An I/O-bound program (say, prog $i$) can be introduced. Its priority would be between the priorities of prog A and prog B. Presence of prog $i$ would improve I/O utilization. It would not affect the progress of prog A at all, since prog A has the highest priority, and it would affect the progress of prog B only marginally, since prog B does not use a significant amount of CPU time. |

Figure 3.5 Variation of throughput with degree of multiprogramming.

When an appropriate program mix is maintained, an increase in the degree of multiprogramming would result in an increase in throughput.
Time-Sharing Systems

• Provide quick response to user subrequests in an interactive computing environment
• *Round-robin scheduling with time-slicing*
  – Kernel maintains a *scheduling queue*
  – If *time slice* (δ) elapses before process completes servicing of a subrequest, kernel preempts it, moves it to end of queue, and schedules another process
  • Implemented through a timer interrupt

**Definition 3.5 Time Slice** The largest amount of CPU time any time shared process can consume when scheduled to execute on the CPU.

---

Time-Sharing Systems (continued)

![Diagram](image)

**Figure 3.6** A schematic of round-robin scheduling with time-slicing.
Time-Sharing Systems (continued)

• Response time (rt): measure of user service
  – If processing of a subrequest requires $\delta$ CPU seconds
    \[ rt = n \times (\delta + \sigma) \]
    \[ \eta = \frac{\delta}{(\delta + \sigma)} \]
    where $\eta$: CPU efficiency,
    $\sigma$: scheduling overhead,
    $n$: number of users using system,
    $\delta$: time required to complete a subrequest

• Actual response time would be different because
  – Some users may be inactive
  – Subrequests do not require exactly $\delta$ CPU seconds

<table>
<thead>
<tr>
<th>Time</th>
<th>Scheduling list</th>
<th>Scheduled program</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$P_1, P_2$</td>
<td>$P_1$</td>
<td>$P_1$ is preempted at 10 ms</td>
</tr>
<tr>
<td>10</td>
<td>$P_2, P_1$</td>
<td>$P_2$</td>
<td>$P_2$ is preempted at 20 ms</td>
</tr>
<tr>
<td>20</td>
<td>$P_1, P_2$</td>
<td>$P_1$</td>
<td>$P_1$ starts I/O at 25 ms</td>
</tr>
<tr>
<td>25</td>
<td>$P_2$</td>
<td>$P_2$</td>
<td>$P_2$ is preempted at 55 ms</td>
</tr>
<tr>
<td>35</td>
<td>$P_2$</td>
<td>$P_2$</td>
<td>$P_2$ starts I/O at 45 ms</td>
</tr>
<tr>
<td>45</td>
<td>$-$</td>
<td>$-$</td>
<td>CPU is idle</td>
</tr>
</tbody>
</table>

Figure 3.7 Operation of processes $P_1$ and $P_2$ in a time-sharing system.
Swapping of Programs

- Swapping allows the kernel to service more processes than can fit into the memory
  - Improve system throughput and response times of processes
  - Kernel performs *swap-out* and *swap-in* operations

**Figure 3.8** Swapping: (a) processes in memory between 0 and 105 ms; (b) $P_2$ is replaced by $P_3$ at 105 ms; (c) $P_4$ is replaced by $P_5$ at 125 ms; (d) $P_1$ is swapped in to service the next subrequest made to it.

**Definition 3.6 Swapping** The technique of temporarily removing a process from the memory of a computer system.