Real-Time Systems
(0630581)

Lecture (8)

Real-Time Operating Systems

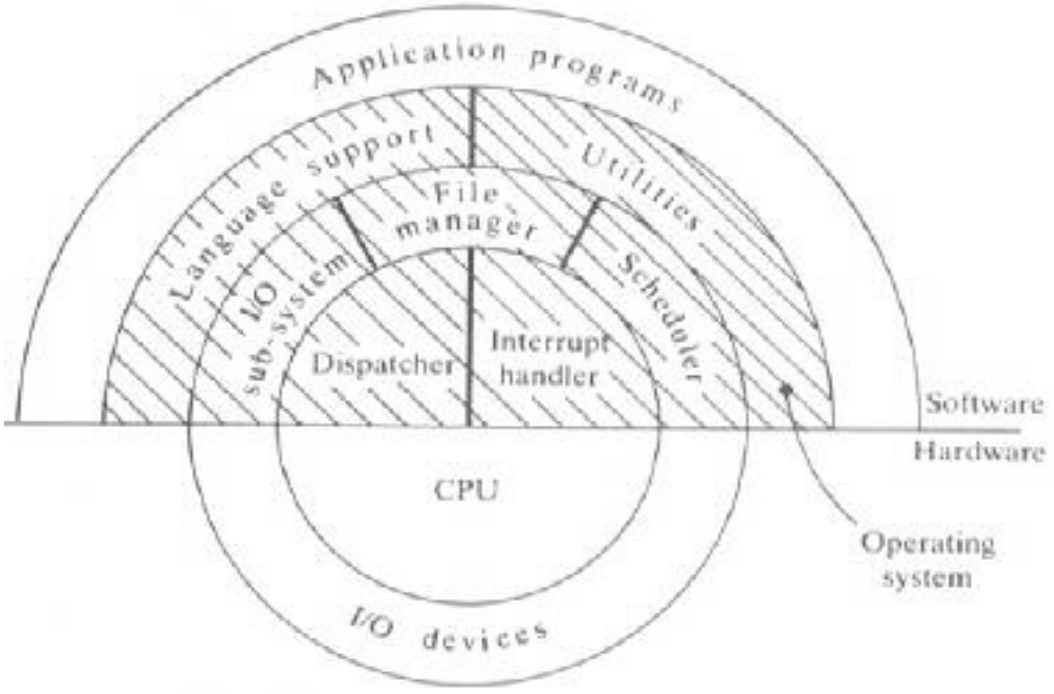
Prof. Kasim M. Al-Aubidy
Computer Engineering Department
Philadelphia University
Summer Semester, 2011

Lecture Outline:

- Explain components of a simple operating system.
- Describe types of operating systems.
- Why we use a RTOS?
- What an RTOS does? How it works?
- Benefits and drawbacks of an RTOS.
- Describe and explain by examples the basic task synchronization mechanisms.

General Purpose Operating System:

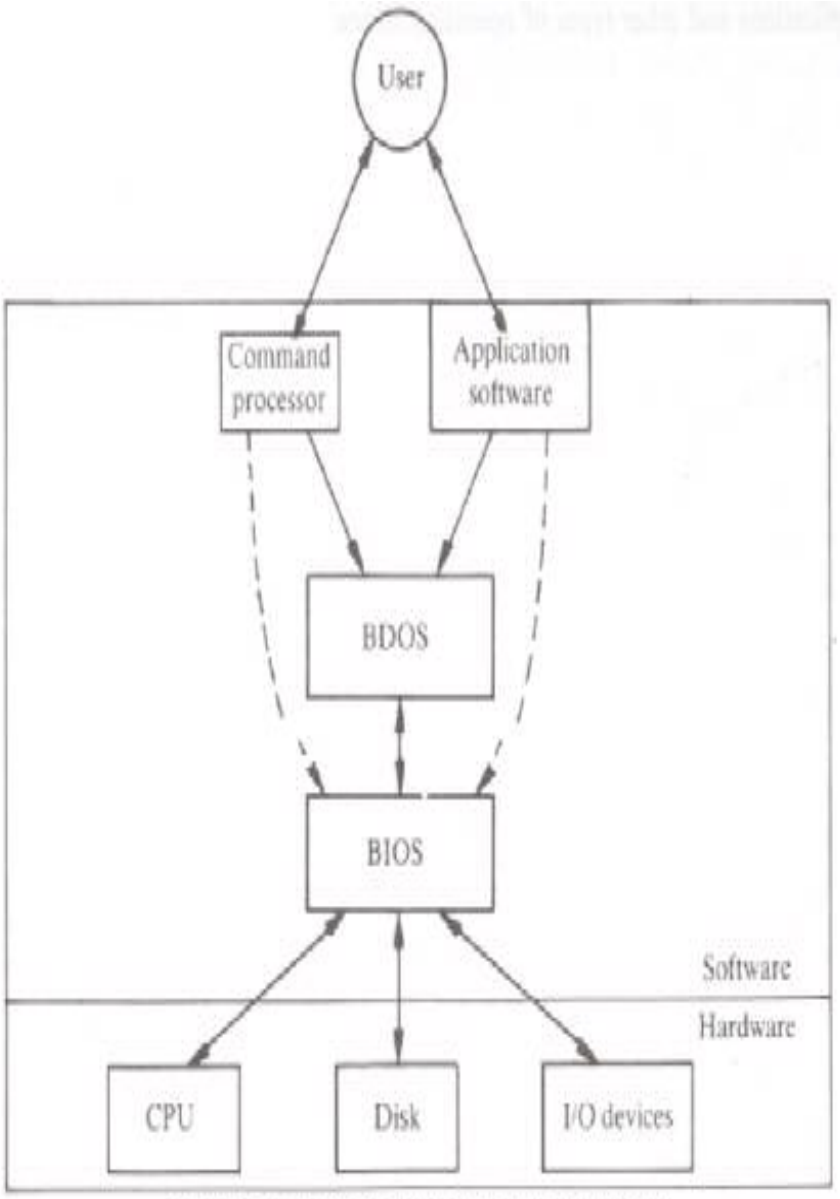
- Access to the hardware of the system and to the I/O devices is through the operating system (OS).
- In many real-time and multiprogramming systems restriction of access is enforced by hardware and software traps.
- A general purpose operating system will provide some facilities that are not required in a particular application.
- Recently, operating systems which provide only a minimum kernel have become popular, additional features can be added.



General purpose operating system

General Structure of a Simple OS:

- The command processor provides a means by which the user can communicate with the OS.
- The actual processing of the user commands is done by BDOS, which also handles the I/O and the file operations on the disks.
- The BDOS makes the actual management of the file and I/O operations transparent to the user.
- Application programs will normally communicate with the hardware of the system through *system calls* which are processed by the BDOS.
- The BIOS contains the various device drivers which manipulate the physical devices and OS.
- Devices are treated as *logical* or *physical* units. Logical devices are software constructs used to simplify the user interface. User programs perform I/O to logical devices and the BDOS connects the logical devices to the physical device.



General structure of a simple operating system

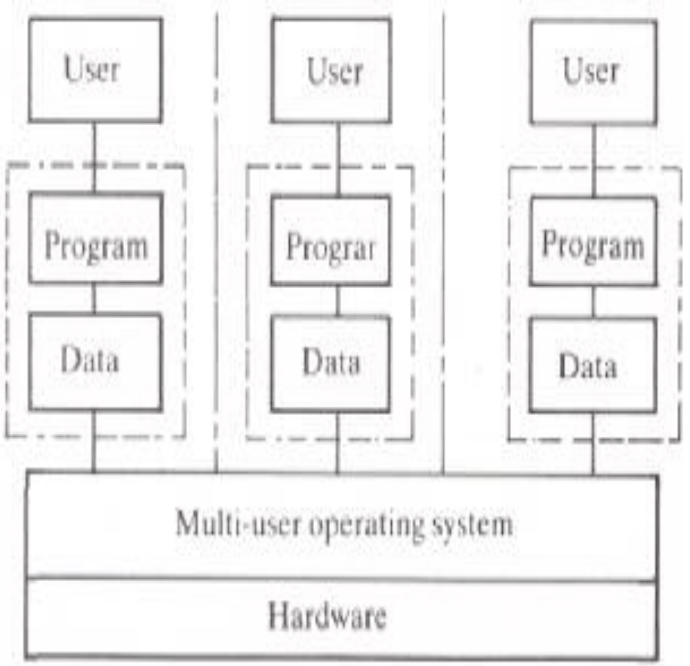
Types of Operating Systems:

There are different types of OSs:

- Single-user and Multi-user operating systems.
- Single-task and Multi-tasking operating systems.
- Real-time operating systems.

Multi-User Operating Systems:

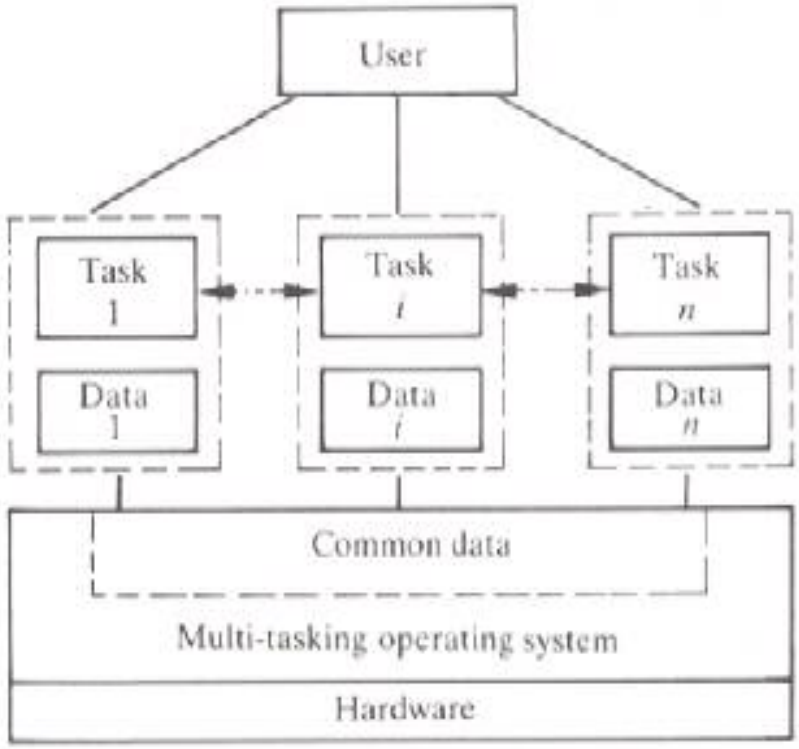
- The OS ensures that each user can run a single program as if the had the whole computer system.
- At any given instance, it is not possible to predict which user will have the use of the CPU.
- The OS ensures that one user program cannot interfere with the operation of another user program. Each user program runs in its own protected environment.



Multi-user operating system

Multi-Tasking Operating Systems:

- In a multi-tasking operating system, it is assumed that there is a single user and that the various tasks co-operate to serve the requirements of the user.
- Co-operation requires that all tasks communicate with each other and share common data.
- Task communication and data sharing will be regulated so that the OS is able to prevent inadvertent communication or data access, and hence protect data which is private to a task.

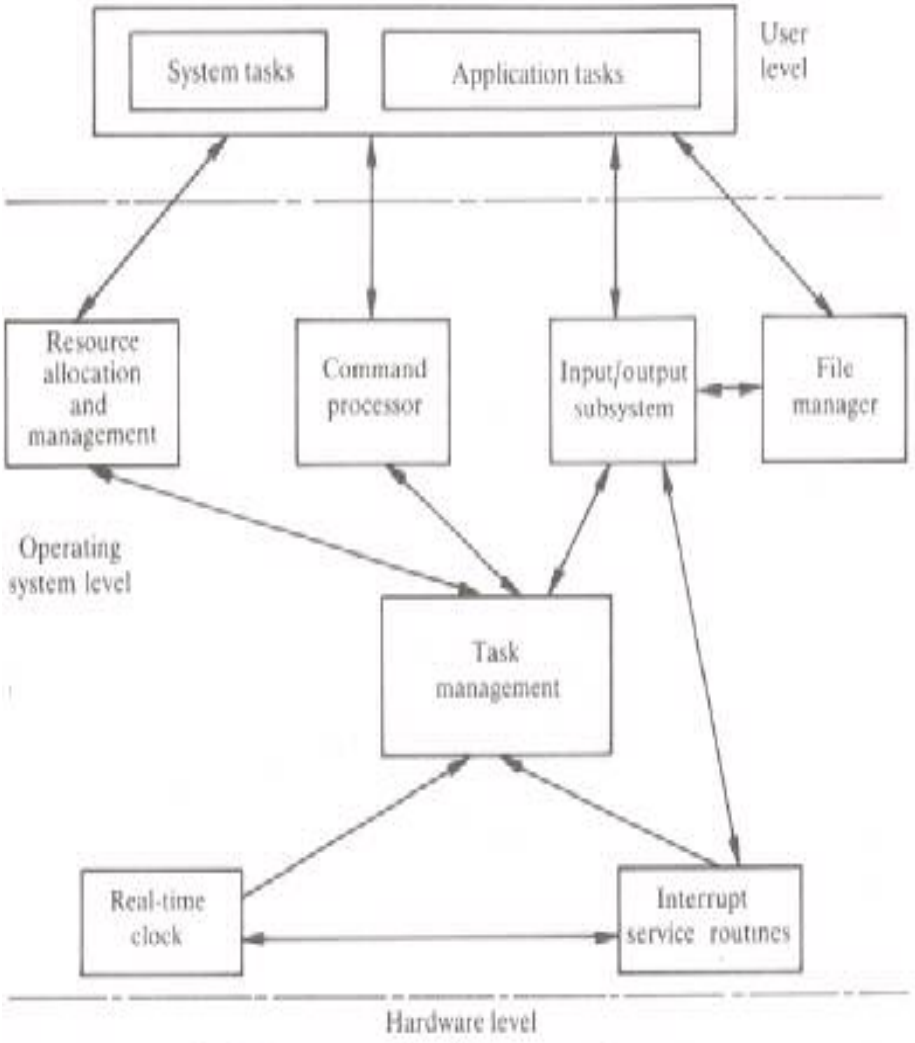


Multi-tasking operating system

Real-Time Operating System (RTOS):

- A fundamental requirement of an operating system is to allocate the resources of the computer to the various activities which have to be performed. In a RTOS this allocation procedure is complicated by the fact that some of the activities are time critical and hence have a higher priority than others. Therefore, there must be some means of allocating priorities to tasks and of scheduling allocation of CPU time to the tasks according to some priority scheme.
- A task may use another task, thus tasks may need to communicate with each other. The OS must have some means of enabling tasks either to share memory for the exchange of data or to provide a mechanism by which tasks can send messages to each other.
- Tasks may need to be invoked by external events and hence the OS must support the use on interrupts.
- Tasks may need to share data and they may require access to various hardware and software components, hence there has to be a mechanism for preventing two tasks from attempting to use the same resource at the same time.

- A real-time multi-tasking operating system has to support the resource sharing and the timing requirements of the tasks an the functions can be divided as follows:
  - **Task Management:** the allocation of memory and processor time (scheduling) to tasks.
  - **Memory Management:** control of memory allocation.
  - **Intertask Communication & Synchronization:** provision of support mechanisms to provide safe communication between tasks and to enable tasks to synchronies their activities.



Typical structure of a real-time operating system

Scheduling Strategies:

There are two basic strategies for the scheduling of time allocation on a single CPU, these are:

1. **Cyclic Strategy:**
   - The task uses the CPU for as long as it wishes.
   - It is a very simple strategy which is highly efficient in that it minimizes the time lost in switching between tasks.
   - It is an efficient strategy for small embedded systems for which the execution times for each task run are carefully calculated and for which the software is carefully divided into appropriate task segments.
   - This approach is too restrictive since it requires that the task units have similar execution times. It is difficult to deal with random events using this approach.
2. **Pre-emptive Strategies:**
   - There are many pre-emptive strategies, all involve the possibility that a task will be interrupted before it has completed a particular invocation.
   - The simplest form of pre-emptive scheduling is to use a time slicing approach. Using this strategy each task is allocated a fixed amount of CPU time (number of clock ticks), and at the end of this time it is stopped and the next task in the list is run. If a task completes before the end of its time slice, the next task in the list is run immediately.

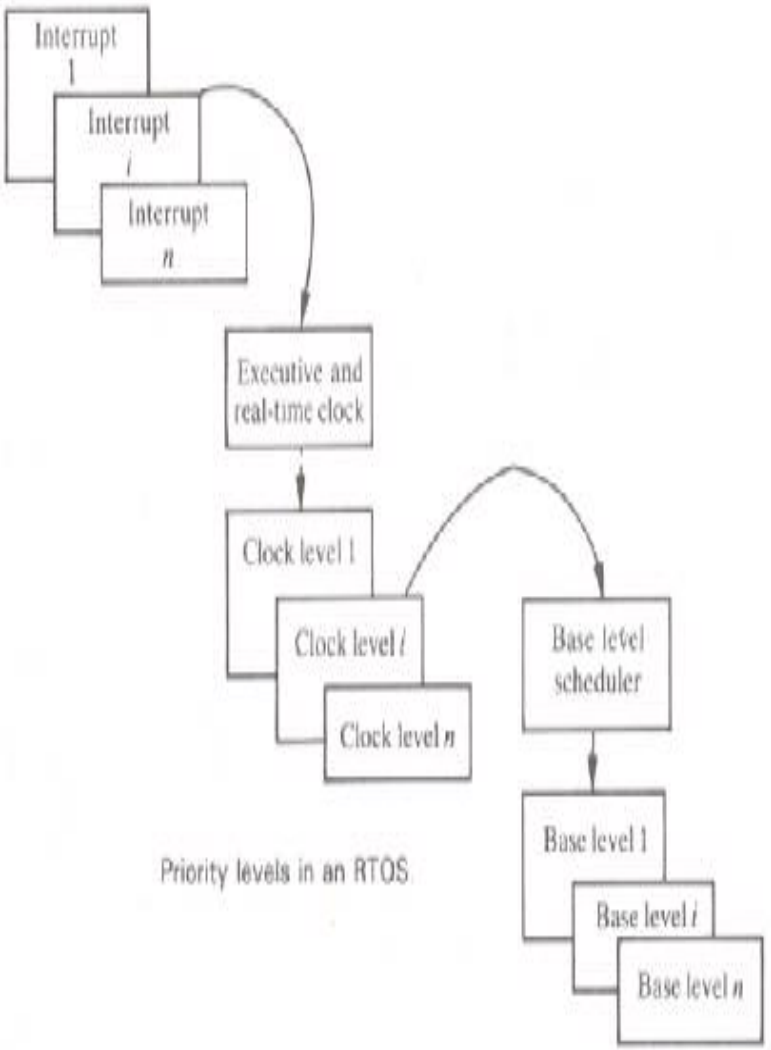
Priority Scheduling Mechanism:

- Tasks are allocated a priority level and at the end of a predetermined time slice, the task with highest priority of those ready to run is chosen and is given control of the CPU.
- Task priorities may be fixed (static priority system) or may be changed during system execution (dynamic priority system).
- Dynamic priority schemes can increase the flexibility of the system.
- Changing priorities is risky as it makes it much harder to predict and test the behavior of the system.
- The task management system has to deal with the handling of interrupts. These may be hardware interrupts caused by external events, or software interrupts generated by a running task.

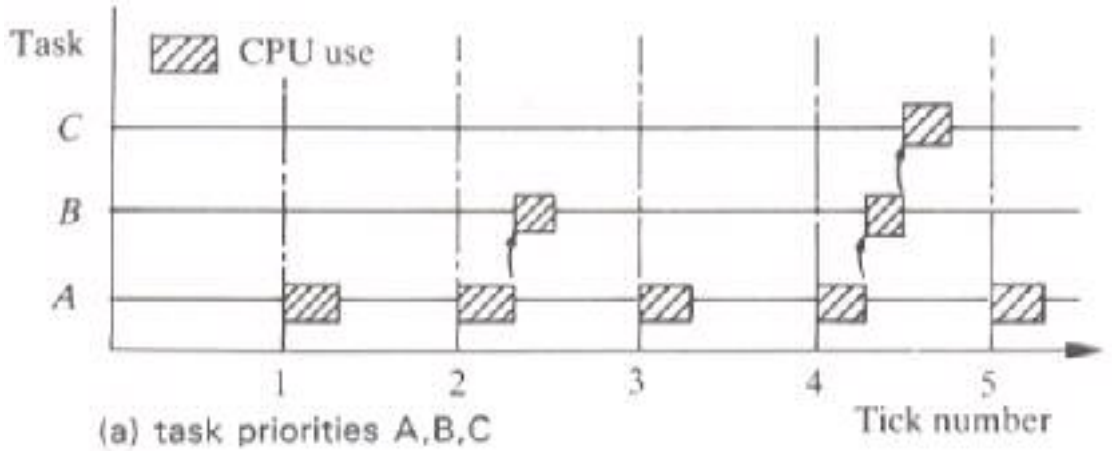
Priority Structures:

- In a real-time system the designer has to assign priorities to the tasks in the system.
- The priority will depend on how quickly a task will have to respond to a particular event.
- Most RTOSs provide facilities such that tasks can be divided into three board levels:

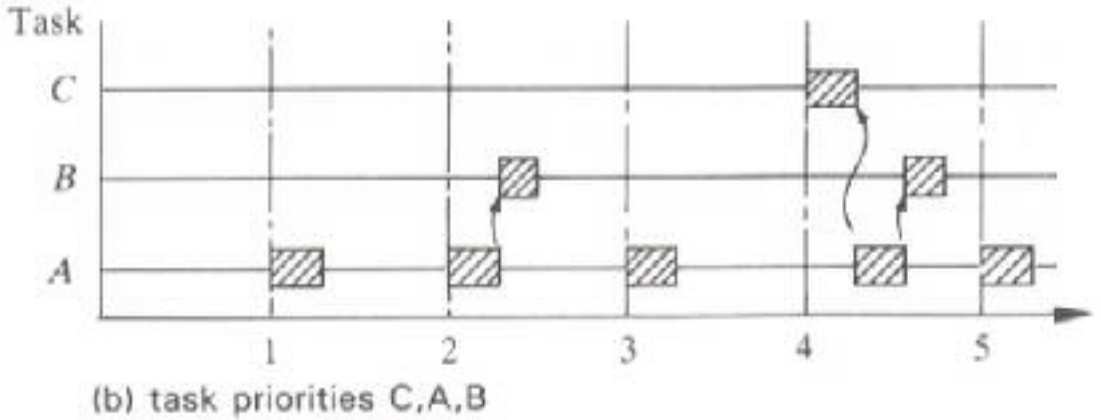
1. **Interrupt Level:** at this level are the service routines for the tasks and devices which require very fast response (measured in msec.) Example: real-time clock task.
2. **Clock Level:** at this level are the tasks which require accurate timing and repetitive processing, such as the sampling and control tasks.
3. **Base Level:** tasks at this level are of low priority and either have no deadlines to meet or are allowed a wide margin of error in their timing. Tasks at this level may be allocated priorities or may all run at a single priority level.



Priority levels in an RTOS



(a) task priorities A,B,C



(b) task priorities C,A,B

Clock Level:

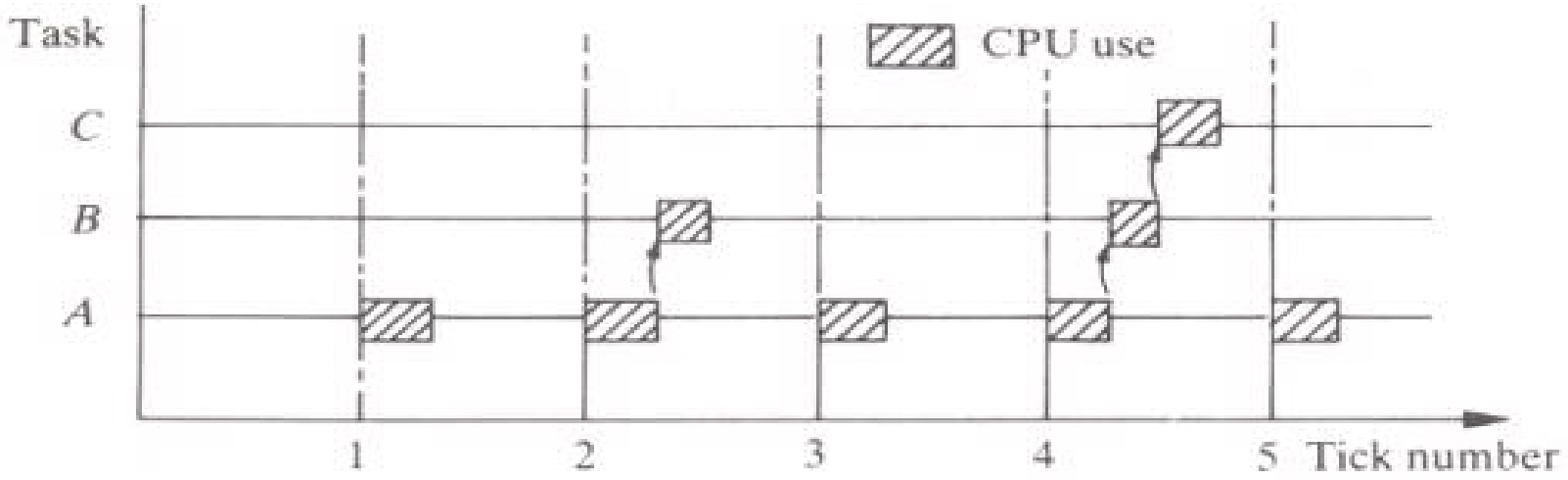
- One interrupt level task will be the real-time clock.
- Typical values 1-200 msec.
- Each clock interrupt is known as a tick and represents the smallest time interval in the system.
- The function of the clock interrupt handling routine is to update the time of day clock in the system and to transfer control to dispatcher.
- The scheduler selects which task is to run at a particular clock rate.
- Clock level tasks divided into two categories;
  - **Cyclic:** these are tasks which require accurate synchronization with outside world.
  - **Delay:** these tasks simply wish to have a fixed delay between successive repetitions or to delay their activities for a given period of time.

Cyclic Tasks:

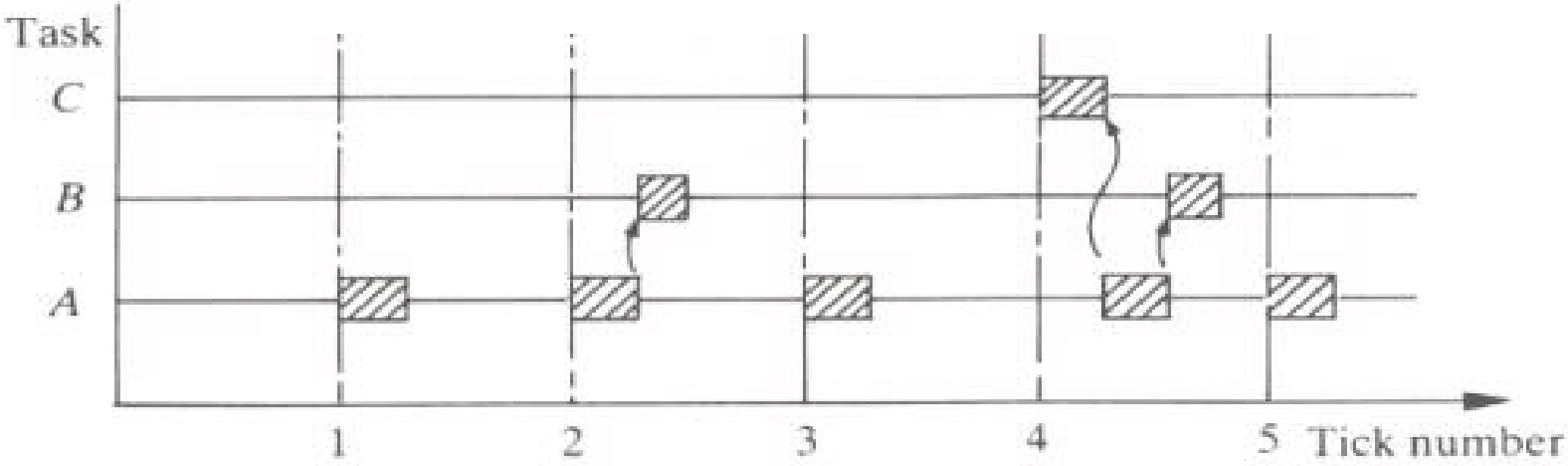
- Cyclic tasks are ordered in a priority which reflects the accuracy of timing required for the task, those which require high accuracy being given the highest priority
- Tasks of lower priority within clock level will have some jitter since they will have to await completion of the higher-level tasks.

Example:

- Three tasks A, B, and C are required to run at 20 msec, 40 msec and 80 msec intervals. If the clock interrupt rate is set at 20 msec. if the task priority order is set as A,B,and C with A as the highest priority.
- The following slid shows task activation diagram for this example in two cases;
- Case (a): Task priorities are: A, B, then C.
- Case (b): Task priorities are: C, A, then B.



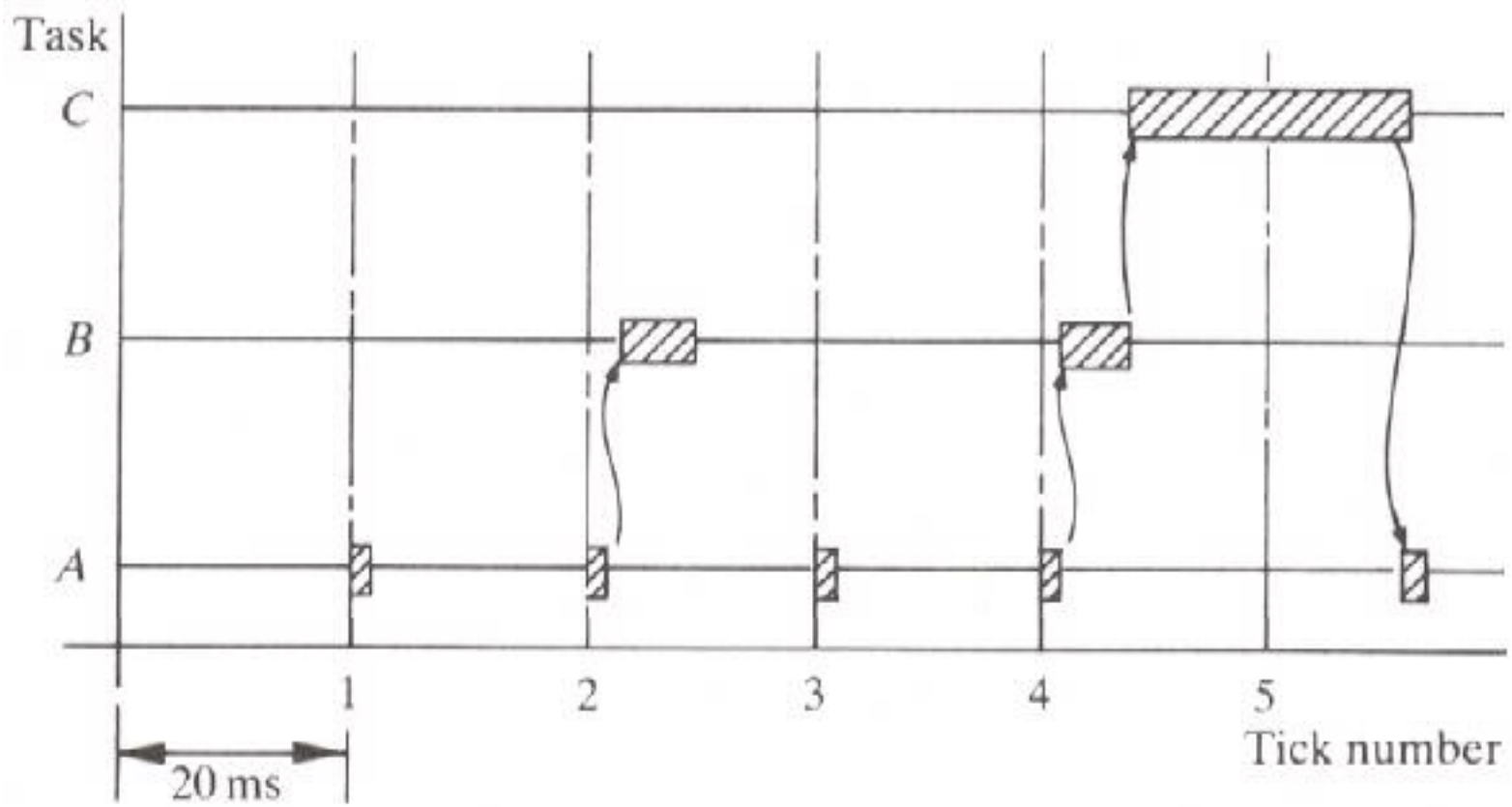
Task activation diagram for task priorities A,B,C.



Task activation diagram for task priorities C,A,B.

Example:

- Now assume that task C takes 25 msec to complete, task A takes 1 msec and task B takes 6 msec. if task C is allowed to run until completion then the activity diagram is given bellow.
- Task A will be delayed by 11 msec at every fourth invocation.

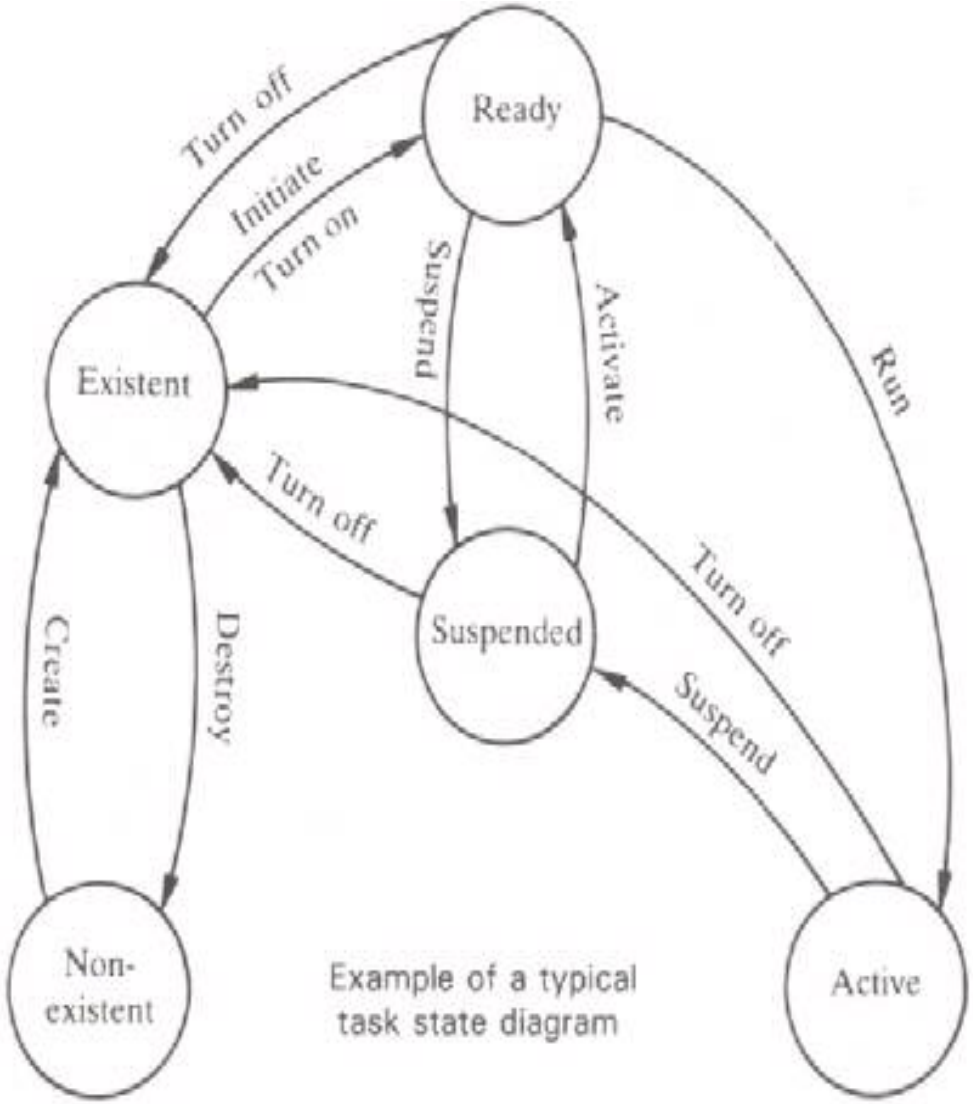


Task States:

Tasks are in one of four states:

1. Running
2. Ready to Run (but not running)
3. Waiting (for something other than the CPU
4. Inactive

- Only one task can be Running at a time (unless we are using a "multi-core" CPU).
- A task which is waiting for the CPU is Ready. When a task has requested I/O or put itself to sleep, it is Waiting.
- An Inactive task is waiting to be allowed into the schedule. It is like Microsoft Word when you are NOT running it.



Example of a typical task state diagram

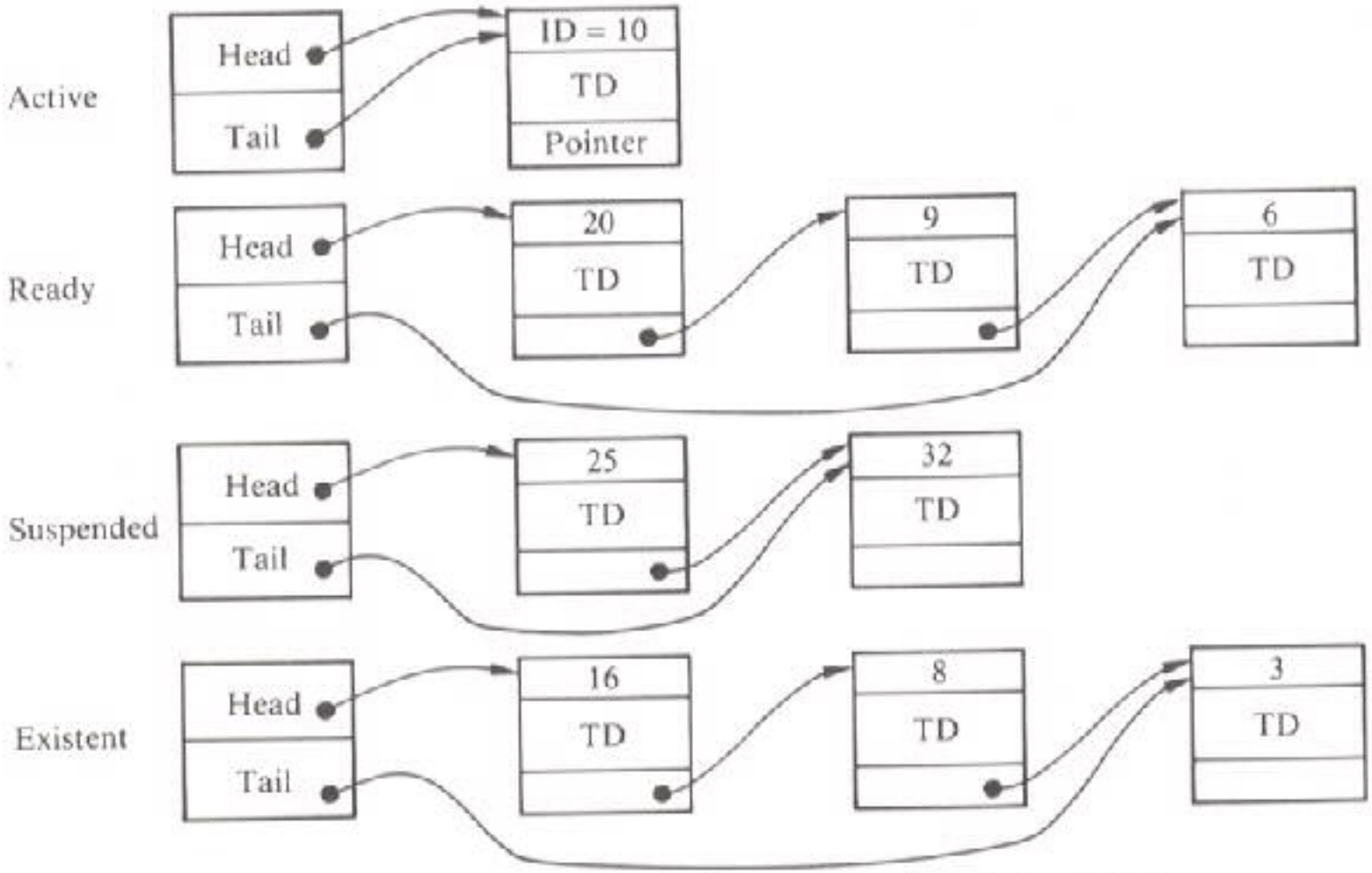
Task Descriptor:

- Information about the status of each task is held in a block of memory by the RTOS. This block is called Task Descriptor (TD), or Task Control Block (TCB) or Task Data Control (TDC).
- The information is held in the TD will vary from system to system, but will typically consist of the following:
  - Task Identification.
  - Task Priority.
  - Current state of task.
  - Area to store volatile environment (or a pointer to an area for storing the volatile environment).
  - Pointer to next task in list.

Example:

The next slide shows list structure for holding task state information:

- There is one active task (task ID=10).
- There are three tasks ready to run (ID=20, ID=9 and ID=6). The entry held in the executive for the ready queue head points to task 20, which in tern points to task 9 and so on.
- The advantage of the list structure is that the actual TD can be located anywhere in the memory and hence the OS is not restricted to a fixed number of tasks as the case in older OSs which used fixed length tables to hold task state information.

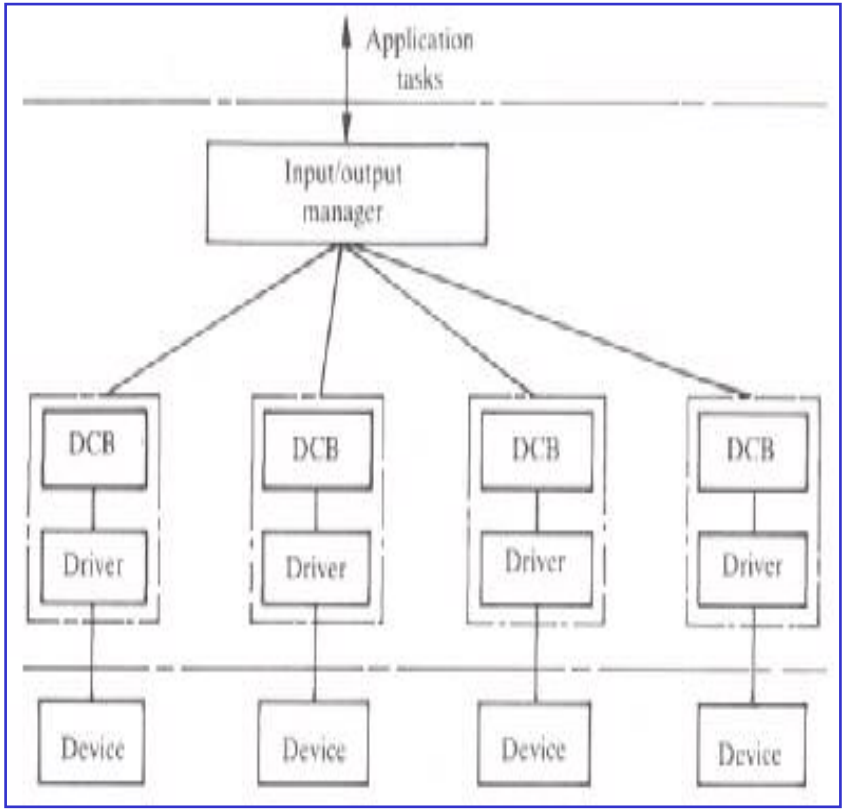


List structure for holding task state information

Resource Control:

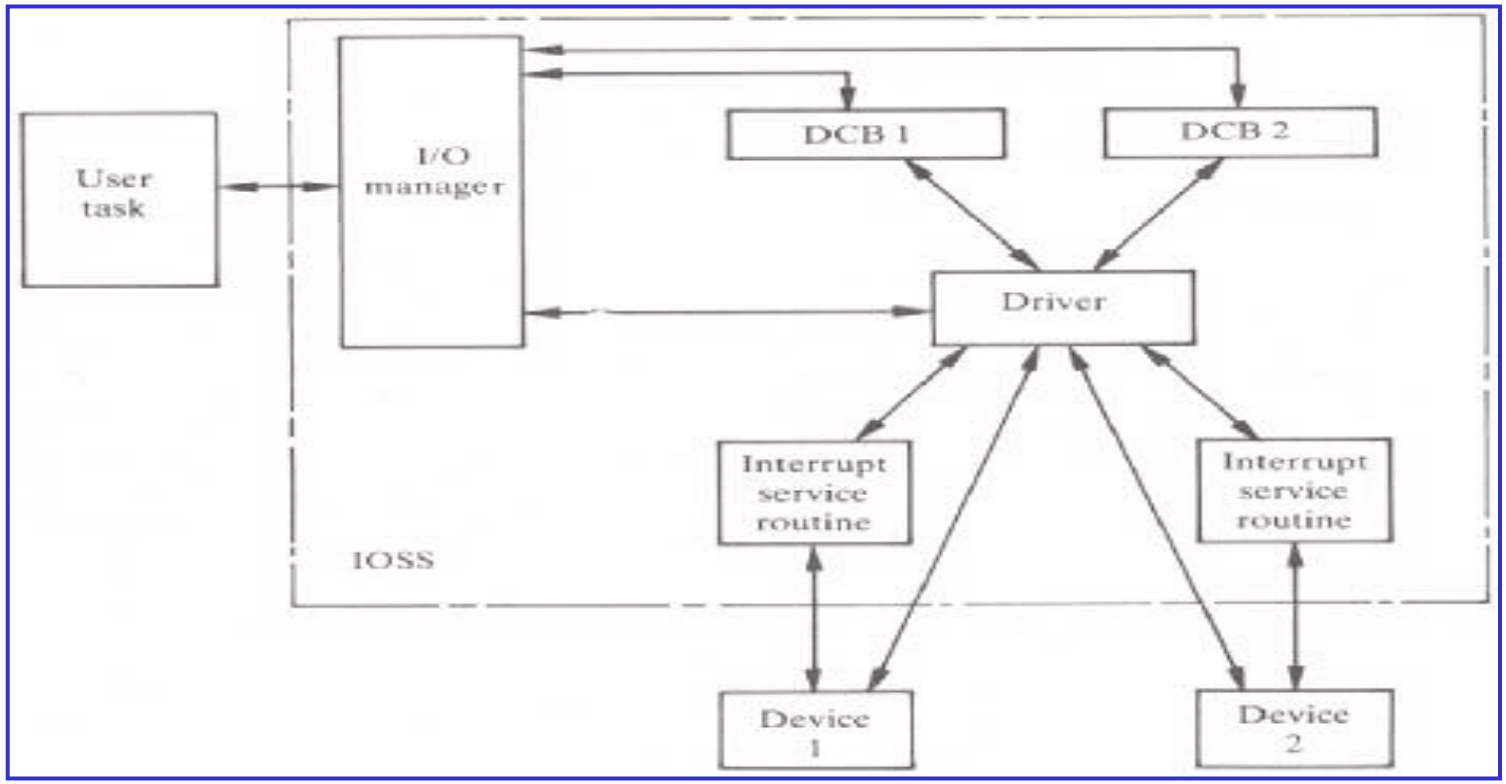
- One of the most difficult areas of programming is the transfer of information to and from external devices. The availability of a well-designed and implemented I/O subsystem (IOSS) in an OS is essential for efficient programming. This enables programmer to perform input output by means of system calls either from a HLL or from the assembler. The IOSS handles all the details of the devices.

- A typical IOSS will be divided into two levels.
- The **I/O manager** accepts the system calls from the user tasks and transfers the information contained in the calls to the **device control block (DCB)** for the particular device.
- The information supplied in the call by the user task will be;
  - the location of a buffer area in which the data to be transferred is stored (o/p) or is to be stored (i/p),
  - the amount of data to be transferred,
  - type of data,
  - direction of transfer, and
  - the device to be used.



Detailed arrangement of IOSS:

- The actual transfer of the data between the user task and the device will be carried out by the device driver and this segment of code will make use of other information stored in the DCB.
- A separate device driver may be provided for each device.
- A single driver may be shared between several devices, however, each device will require its own DCB.
- The OS will normally be supplied with DCBs for the more common devices.



For more information:

1. http://www.cs.ou.edu/~fagg/umass/classes/377f02/lectures.html

2. http://www.cs.umbc.edu/~younis/Real-Time/CMSC691S.htm#D