SVKM'S NMIMS, School of Technology Management & Engineering | Navi-Mumbai

MBA-Tech (A.Y. 2019-20)

Lecture Notes

(I/O and multi-processor organization Unit)

Input/Output Subsystem:

The I/O subsystem of a computer provides an efficient mode of communication between the central system and the outside environment. It handles all the input-output operations of the computer system.

Peripheral Devices

Input or output devices that are connected to computer are called peripheral devices. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be the part of computer system. These devices are also called peripherals.

For example: Keyboards, display units and printers are common peripheral devices.

There are three types of peripherals:

1. Input peripherals: Allows user input, from the outside world to the computer. Example: Keyboard, Mouse etc.
2. Output peripherals: Allows information output, from the computer to the outside world. Example: Printer, Monitor etc
3. Input-Output peripherals: Allows both input (from outside world to computer) as well as, output (from computer to the outside world). Example: Touch screen etc.

Interfaces:

Interface is a shared boundary between two separate components of the computer system which can be used to attach two or more components to the system for communication purposes.

Input-Output Interface:

Peripherals connected to a computer need special communication links for interfacing with CPU. In computer system, there are special hardware components between the CPU and peripherals to control or manage the input-output transfers. These

components are called input-output interface units because they provide communication links between processor bus and peripherals. They provide a method for transferring information between internal system and input-output devices.

Modes of I/O Data Transfer:

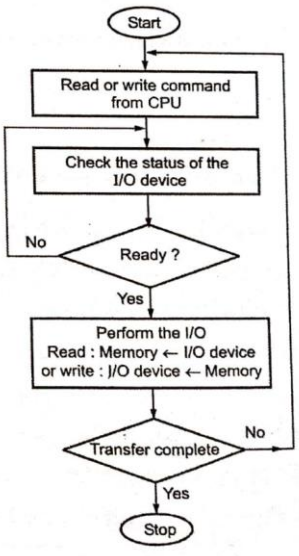
Data transfer between the central unit and I/O devices can be handled in generally three types of modes which are given below:

- Programmed I/O
- Interrupt Initiated I/O
- Direct Memory Access

Programmed I/O:

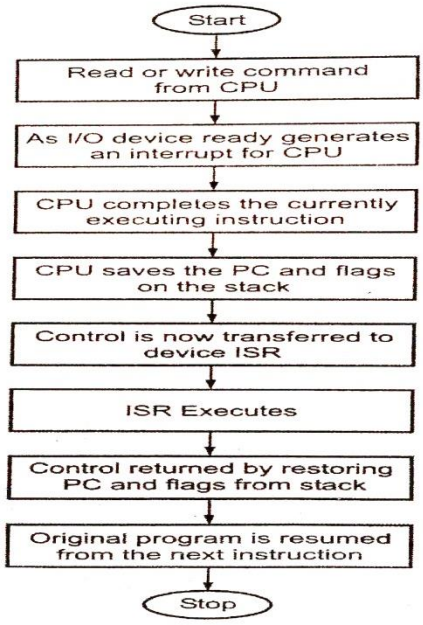
- Programmed I/0 is the simplest technique of performing input/output data transfer and it is a very effective method when the number of devices in the system is less in number.
- It is best on the concept of a program controlling the input / output, data transfer and the flowchart in Fig. provides sequence of steps carried out in programmed I/O.

1. CPU generates the desired I/0 command I. e. Either Read or write.
2. CPU checks the status of I/O device whether it is ready to perform the specified Read or Write operation.
3. If device is not ready CPU remains "Busy Waiting" I. e. It keeps on checking the status of the device repeatedly and continuously.
4. If the device is ready, the specified Read or Write I/0 operation is completed.
5. It is checked whether any more I/O transfers are pending. If pending, the complete is sequence is repeated.



Interrupt Driven I/O:

- Programmed I/0 is not an efficient technique due to the fact that the CPU consumes substantial amount of line in checking the devices repeatedly turn by turn.
- Furthermore the devices are also required to wait for their turn. The better technique is Interrupt driver Input / Output. In this method the CPU is not sequencing the Input / Output device on its own.
- When the device is required to perform Input / output data transfers it intimates so the CPU by activating the hardware line which goes to CPU as an input signal. This is line is called as an Interrupt and therefore this I/0 is termed as interrupt driven Input / Output.
- The mechanism of Input / Output data transfer using Interrupt driven Input'/ output is shown is Fig. It is explained by following sequence of steps:
  1. CPU generates instructions for Read or Write operation from / to Input/Output device.
  2. When the input / output device is ready to perform l/0 data transfer, it generates an Interrupt to the CPU.
  3. CPU completes the instruction currently in execution and then recognizes the interrupt from Input / Output device.
  4. It saves the context of currently executing program on the top of the stack.
  5. Then the control is transferred to the start of interrupt service routine (ISR).
  6. Now the CPU services the Interrupting I/O device performing the under lying I/O data transfer.
  7. Then the CPU comes back, retrieves the context from the stack and continues the original execution executing program resuming it from the next instruction onwards.



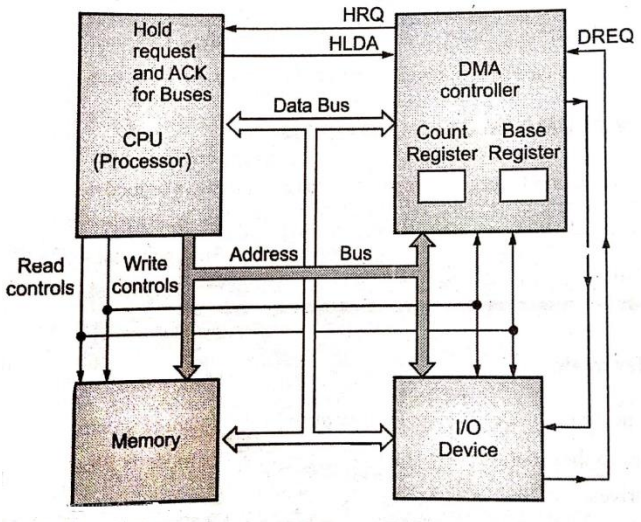
DMA Controlled I/O:

Need of Direct Memory Access (DMA):

- Data Transfers between Memory and I/O device have their source and destination, respectively in memory and I/O devices respectively. Theoretically, they do not require the Processor or CPU as it is not involved in such transfers.
- Therefore, involving processor or CPU for such transfers, is an inefficient process and consumes the CPU time unnecessarily. If it is made possible to do Memory and I/O data transfers without involving the CPU, it would be highly advantageous.
- This requirement necessitates the need of Direct Memory Access (DMA) which is the technique that achieves these data transfers without using or involving the CPU.
- The use of DMA, therefore, is for performing data transfers from memory to I/O device, I/O device to memory and from one memory to another memory; without the involvement of CPU.

Direct Memory Access (DMA):

- The technique of accessing memory directly without the involvement of the CPU for performing Input / output data transfers; is called as direct memory access or DMA.
- The data transfers involving DMA technique are the transfers from memory to I/O device, I/O device to memory and from one memory to another memory. These data transfers can be performed without the direct involvement of the CPU.
- However, they do require the data bus, the address bus and control bus. By default, these three buses are in control of the CPU. Therefore whenever the DMA operation is to be performed without involving CPU, there is a need to transfer the buses.
- For this purpose, a separate hardware unit called as DMA controller becomes necessary. The DMA controller is used for taking over the control of the buses from the CPU completing the DMA operation and then returning the buses back to the CPU. The typical implementation of DMA based I/O using the DMA controller is as shown in Fig.

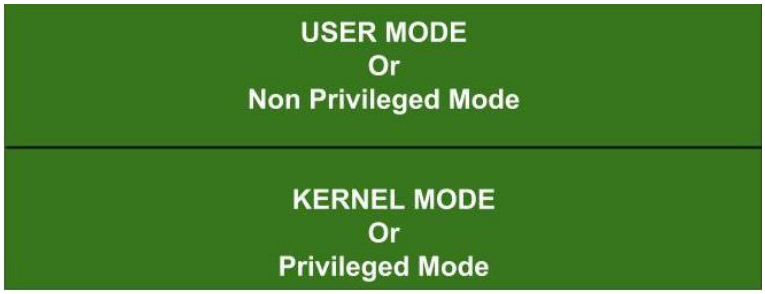


- The sequence of steps for performing I/O data transfers using DMA controller are as follows:

1. CPU programs the internal registers (Base Address Register and Count Register)of DMA Controller and prepares it for the DMA activity.
2. When the Device is ready for DMA transfers, it activates DREQ signal to the DMA Controller. In turn, DMA controller activates HOLD signal to the CPU.
3. CPU completes the current machine cycle and then tri-states the system buses and stops driving them. It indicates so, by activating the HLDA signal to DMA controller which in turn activates #DACK signal to the device.
4. DMA Controller takes over the control of buses. It issues appropriate addresses and control signals to perform DMA based data transfers. After every byte transfer, Base address and Count is modified. The activity can continue till Count becomes Zero.
5. As DMA activity is over, Device deactivates DREQ signal to DMA Controller and it turn gets #DACK signal also being deactivated.
6. DMA Controller stops driving the buses and deactivates HOLD signal to the CPU.
7. CPU takes back the control of buses and indicates so by deactivating HLDA signal. Subsequently, CPU continues its work normally.

Privileged and Non-Privileged Instructions in Operating System:

In any Operating System, it is necessary to have Dual Mode Operation to ensure protection and security of the System from unauthorized or errant user's. This Dual Mode separates the User Mode from the System Mode or Kernel Mode.



What are Privileged Instructions?

The Instructions that can run only in Kernel Mode are called Privileged Instructions. Privileged Instructions possess the following characteristics:

(i) If any attempt is made to execute a Privileged Instruction in User Mode, then it will not be executed and treated as an illegal instruction. The Hardware traps it to the Operating System.

(ii) Before transferring the control to any User Program, it is the responsibility of the Operating System to ensure that the Timer is set to interrupt. Thus, if the timer interrupts then the Operating System regains the control.

Thus, any instruction which can modify the contents of the Timer is a Privileged Instruction.

(iii) Privileged Instructions are used by the Operating System in order to achieve correct operation.

(iv) Various examples of Privileged Instructions include:

- I/O instructions and Halt instructions
- Turn off all Interrupts
- Set the Timer
- Context Switching
- Clear the Memory or Remove a process from the Memory
- Modify entries in Device-status table

What are Non-Privileged Instructions?

The Instructions that can run only in User Mode are called Non-Privileged Instructions. Various examples of Non-Privileged Instructions include:

- Reading the status of Processor
- Reading the System Time
- Generate any Trap Instruction
- Sending the final printout of Printer

Also, it is important to note that in order to change the mode from Privileged to Non-Privileged, we require a Non-privileged Instruction that does not generate any interrupt.

Interrupts and Exceptions:

Exceptions and interrupts are unexpected events which will disrupt the normal flow of execution of instruction (that is currently executing by processor). An exception is an unexpected event from within the processor. Interrupt is an unexpected event from outside the process.

Whenever an exception or interrupt occurs, the hardware starts executing the code that performs an action in response to the exception. This action may involve killing a process, outputting an error message, communicating with an external device, or horribly crashing the entire computer system by initiating a “Blue Screen of Death” and halting the CPU. The instructions responsible for this action reside in the operating system kernel, and the code that performs this action is called the interrupt handler code. Now, we can think of handler code as an operating system subroutine. Then, After

the handler code is executed, it may be possible to continue execution after the instruction where the execution or interrupt occurred.

Exception and Interrupt Handling:

Whenever an exception or interrupt occurs, execution transition from user mode to kernel mode where the exception or interrupt is handled. In detail, the following steps must be taken to handle an exception or interrupts.

While entering the kernel, the context (values of all CPU registers) of the currently executing process must first be saved to memory. The kernel is now ready to handle the exception/interrupt.

1. Determine the cause of the exception/interrupt.
2. Handle the exception/interrupt.

When the exception/interrupt have been handled the kernel performs the following steps:

1. Select a process to restore and resume.
2. Restore the context of the selected process.
3. Resume execution of the selected process.

At any point in time, the values of all the registers in the CPU defines the context of the CPU. Another name used for CPU context is CPU state.

The exception/interrupt handler uses the same CPU as the currently executing process. When entering the exception/interrupt handler, the values in all CPU registers to be used by the exception/interrupt handler must be saved to memory. The saved register values can later restore before resuming execution of the process.

The handler may have been invoked for a number of reasons. The handler thus needs to determine the cause of the exception or interrupt. Information about what caused the exception or interrupt can be stored in dedicated registers or at predefined addresses in memory.

Next, the exception or interrupt needs to be serviced. For instance, if it was a keyboard interrupt, then the key code of the key press is obtained and stored somewhere or some other appropriate action is taken. If it was an arithmetic overflow exception, an error message may be printed or the program may be terminated.

The exception/interrupt have now been handled and the kernel. The kernel may choose to resume the same process that was executing prior to handling the exception/interrupt or resume execution of any other process currently in memory.

The context of the CPU can now be restored for the chosen process by reading and restoring all register values from memory.

The process selected to be resumed must be resumed at the same point it was stopped. The address of this instruction was saved by the machine when the interrupt occurred, so it is simply a matter of getting this address and make the CPU continue to execute at this address.

- **Role of interrupt in process state transition:**

Here is process state diagram from Modern Operating Systems. Transition from running to ready happens when the scheduler picks another process.

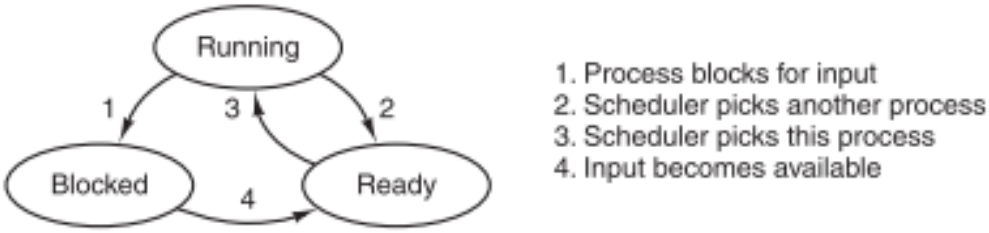


Figure 2-2. A process can be in running, blocked, or ready state. Transitions between these states are as shown.

Here is process state diagram from Operating System Concepts. What does "Interrupt" mean for transition from running to ready? Is it the same as "the scheduler picks another process" in the above?

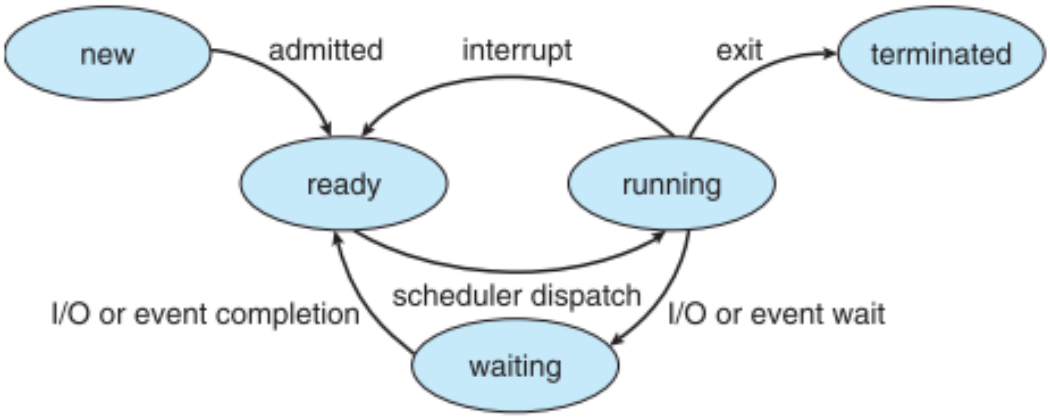
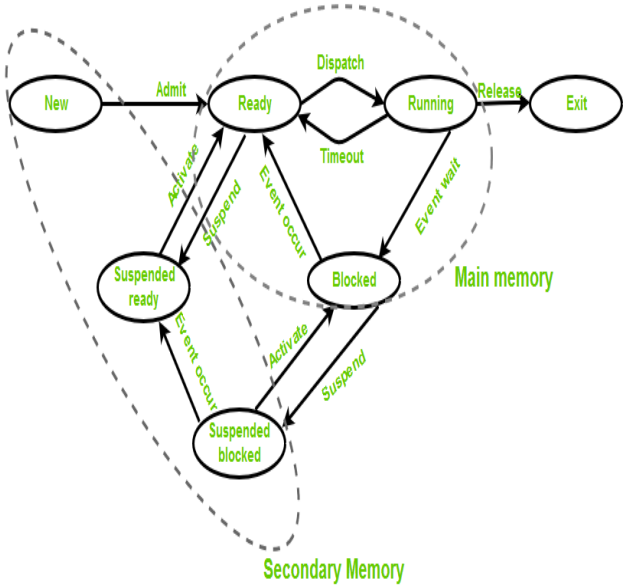


Figure 3.2 Diagram of process state.

States of a process are as following:



- **New (Create) –** In this step, the process is about to be created but not yet created, it is the program which is present in secondary memory that will be picked up by OS to create the process.
- **Ready –** New -> Ready to run. After the creation of a process, the process enters the ready state i.e. the process is loaded into the main memory. The process here is ready to run and is waiting to get the CPU time for its execution. Processes that are ready for execution by the CPU are maintained in a queue for ready processes.
- **Run –** The process is chosen by CPU for execution and the instructions within the process are executed by any one of the available CPU cores.
- **Blocked or wait –** Whenever the process requests access to I/O or needs input from the user or needs access to a critical region (the lock for which is already acquired) it enters the blocked or wait state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state.
- **Terminated or completed –** Process is killed as well as PCB is deleted.
- **Suspend ready –** Process that was initially in the ready state but were swapped out of main memory (refer Virtual Memory topic) and placed onto external storage by scheduler are said to be in suspend ready state. The process will transition back to ready state whenever the process is again brought onto the main memory.
- **Suspend wait or suspend blocked –** Similar to suspend ready but uses the process which was performing I/O operation and lack of main memory caused them to move to secondary memory. When work is finished it may go to suspend ready.