**Unit 2 Process Management**

**2.1 Process Concepts**

**2.1.1 Definitions of Process**

A process is an instance of a program in execution. A program by itself is not a process; a program is a ***passive entity***, such as a file containing a list of instructions stored on disks (often called an executable file), whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources. A program becomes a process when an executable file is loaded into memory.

***Definition:*** A process is defined as an entity which represents the basic unit of work to be

implemented in the system.

**2.1.2 Process states**

*A process goes through a series of discrete process states.*

**Running state:**

-Process executing on CPU. -Only one process at a time.

**Ready state:**

-Process that is not allowed to CPU but is ready to run.

-a list of processes ordered based on priority.

**Blocked state:**

-Process that is waiting for some event to happen(e.g. I/O completion events).

-a list of processes (no clear priority).

**2.1.3 Process state transition**

*When a job is admitted to the system, a corresponding process is created and normally inserted at the back of the ready list.*

***dispatch(process name): ready -> running:***When the CPU becomes available, the process is said to make a state transition from ready to running.

***Time run out(process name): running -> ready:***To prevent any one process from monopolizing the CPU, OS specify a time period (quantum) for the process. When the quantum expires makes state transition running to ready.

***block(process name): running -> block.:***When the process requires an I/O operation before quantum expire, the process voluntarily relinquishes the CPU and changed to the blocked state.

***wakeup(process name): blocked -> ready* :** When an I/O operation completes. The process make the transition from block state to ready state.*.*



***Figure: Process states transition diagram***

**2.1.4 Process Control Block (PCB)**

Each process is represented in the operating system by a process control block (PCB) also called a task control block. PCB is the data structure used by the operating system. Operating system groups all information that needs about particular process.



***Figure: PCB***

PCB contains many pieces of information associated with a specific process which is described below.

|  |  |
| --- | --- |
| **S.N.** | **Information & Description** |
| 1 | **Pointer**  Pointer points to another process control block. Pointer is used for maintaining the scheduling list. |
| 2 | **Process State** Process state may be new, ready, running, waiting and so on. |
| 3 | **Program Counter** Program Counter indicates the address of the next instruction to be executed for this process. |
| 4 | **CPU registers**  CPU registers include general purpose register, stack pointers, index registers and accumulators etc. number of register and type of register totally depends upon the computer architecture. |
| 5 | **Memory management information**  This information may include the value of base and limit registers, the page tables, or the segment tables depending on the memory system used by the operating system. This information is useful for de-allocating the memory when the process terminates. |
| 6 | **Accounting information**  This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc. |

Process control block includes CPU scheduling, I/O resource management, file management information etc. The PCB serves as the repository for any information which can vary from process to process. Loader/linker sets flags and registers when a process is created. If that process gets suspended, the contents of the registers are saved on a stack and the pointer to the particular stack frame is stored in the PCB. By this technique, the hardware state can be restored so that the process can be scheduled to run again.

**2.1.5 Operations on Processes**

The processes in the system can execute concurrently, and they must be created and deleted dynamically. OS provide the mechanism for process *creation* and *termination*.

**Process Creation:** There are four principal events that cause the process to be created:

» System initialization.

» Execution of a process creation system call.

» User request to create a new process.

» Initiation of a batch job.

*A process may create several new processes during the course of execution. The creating process is called a parent process, where as the new processes are called children of that process.*

***Two ways to create a new process***

***1. Build a new one***

-Load specified code and data into memory.

-Create and initialize PCB.

-Put processes on the ready list.

***2. Clone an existing one (e.g. Unix fork() syscall)***

-Stop the current process and save its state.

-Make copy of code, data, stack, and PCB.

-Add new process PCB to ready list.

**Process Termination:** Process are terminated on the following conditions.

1. Normal exit.

2. Error exit.

3. Fatal error.

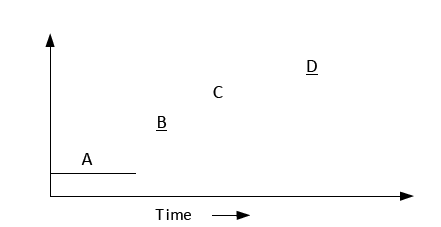
4. Killed by another process.

***Example:*** In Unix the normal exit is done by calling a *exit* system call. The process return data (output) to its parent process via the *wait* system call. *Kill* system call is used to kill other process.

**\*\*\* The process Model (Not in Syllabus but Required)**

1. **Uni-programming:**

* Only one process at a time.
* Examples: Older systems
* Advantages: Easier for OS designer
* Disadvantages: Not convenient for user and poor performance

****

***Figure: Uni-processing***

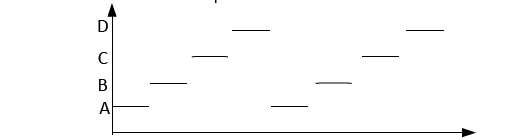
1. **Multiprogramming:**

* Multiple processes at a time.
* OS requirements for multiprogramming:

*Policy:* to determine which process is to schedule.

*Mechanism:* To switch between the processes .

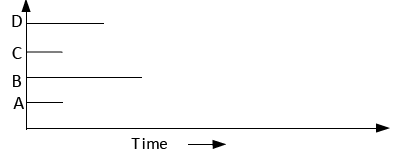
* *Examples:* Unix ,Windows NT

****

***Figure: Multiprogramming***

1. **Multiprocessing:**

* System with multiple processors.

**

*Figure: Multiprocessing*

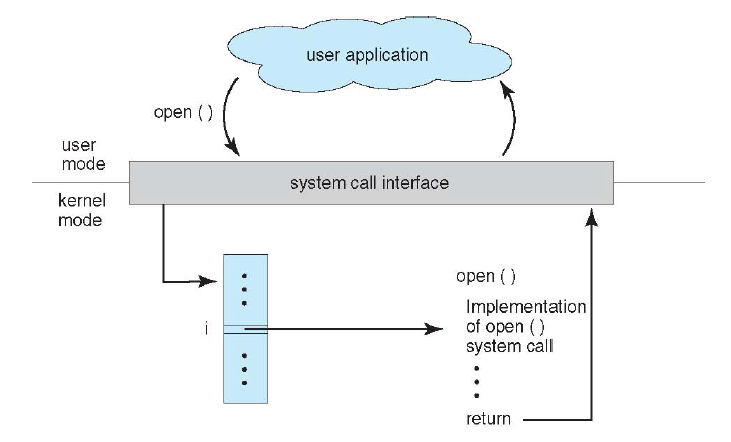
**Cooperating process**

The processes executing in the O.S. may be independent or cooperating processes. An independent process is such which does not get affected by other processes. Generally these type of process does not share resources with other processes.

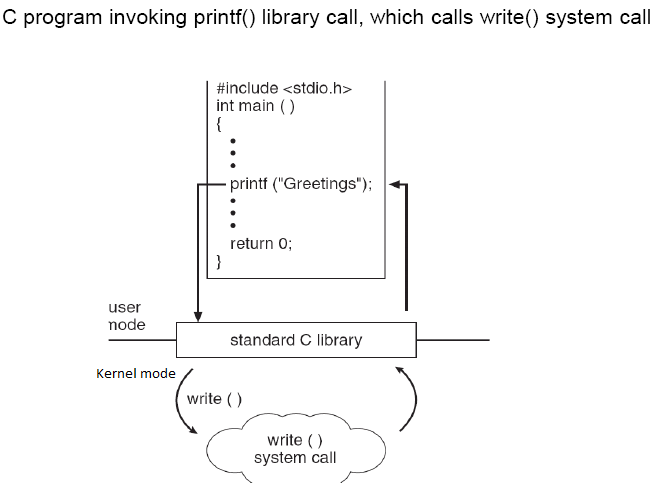
Cooperating processes are those which can affect or be affected by other executing processes in the system. Generally, these type of processes share resources with other processes. Such processes also sometime use the result of other processes as an input.

**System calls**

* System calls provide the interface between a process and the operating system.
* Generally available as assembly-language instructions. Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)



***Figure: API system call***

******

**\*\*\*\*Types of System calls(Not in Syllabus but important)**



