Process Concept & Scheduling

Swati Mali

Nirmala Shinde Baloorkar

Assistant Professor

Department of Computer Engineering

Outline

- Basic Terminology
- Process

Basic Terminologies & Definitions

Task

- A task is a unit of assigned work. It can also be defined as the unit of programming controlled by an operating system (OS). Depending on the OS design, the task may involve one or more processes.
- E.g.
  - Download a file from the internet
  - Bake a cake

Program

- A program is a **sequence of instructions** written to accomplish a task.
- A program may comprise **one or more processes** depending on the statement being executed.
- It is generally referred to as a **passive entity** that does not perform any action until it is executed.
- E.g.
  - A particular recipe given in a book
  - A web application developed

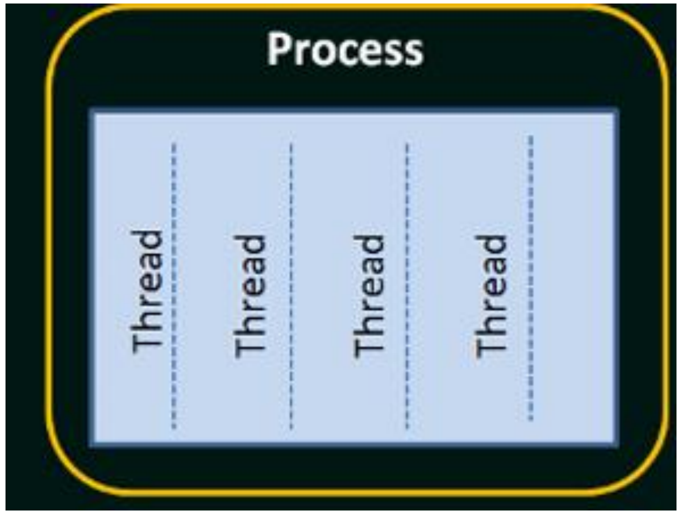
Job

- A job is a unit of work submitted by a user to the system.
- A job may be interactive or a batch job, which may in turn consist of one or more processes.
- E.g.
  - A user submitting a print job to a printer
  - Executing a scheduled task to back up files

Process and Thread

- A process is an instance of a program in execution.
- It is a basic unit of work that can be scheduled and executed by the operating system.

- A thread is the unit of execution within a process.
- A process can have anywhere from just one thread to many threads.



Process Mode

A process can execute in either user mode or kernel mode, with the processor switching between these modes depending on the code being executed.

- User Mode: The process runs with limited privileges, typically to prevent it from performing harmful operations.
- Kernel Mode: The process runs with full access to all system resources and hardware, necessary for performing critical tasks.
- E.g.: Typing / Editing document

Process Context

- When a running process is taken away from the processor, certain information about its state needs to be saved to allow it to resume execution later.
- The process context includes:
  - address space,
  - stack space,
  - virtual address space,
  - register set image i.e. Program Counter, Stack Pointer, Program Status Word, Instruction Register and other general processor registers,
  - accounting information,
  - associated kernel data structures and
  - current state of the process (waiting, ready, etc).

Event

- An activity that is happens or is expected to happen.
- Generally this a software message exchanged when the activity occurs.
- In operating systems, the events may cause the processes to change their state.
    - e.g. mouse click, file lock reset, etc.

- Check- event viewer in windows OS

Process Concept

- Process priority:
  - In a multiprogramming system, the processes are assigned numerical privileged values indicating their relative importance and/or urgency and/or value.
- Preemptive:
  - Preemption is the ability of the system to take over a currently executing process by another one (possibly with high privileged one) with an intention to resume the preempted process later on.
- Non-preemptive:
  - Non-preemptive entities are the one those cannot be taken over by other entities

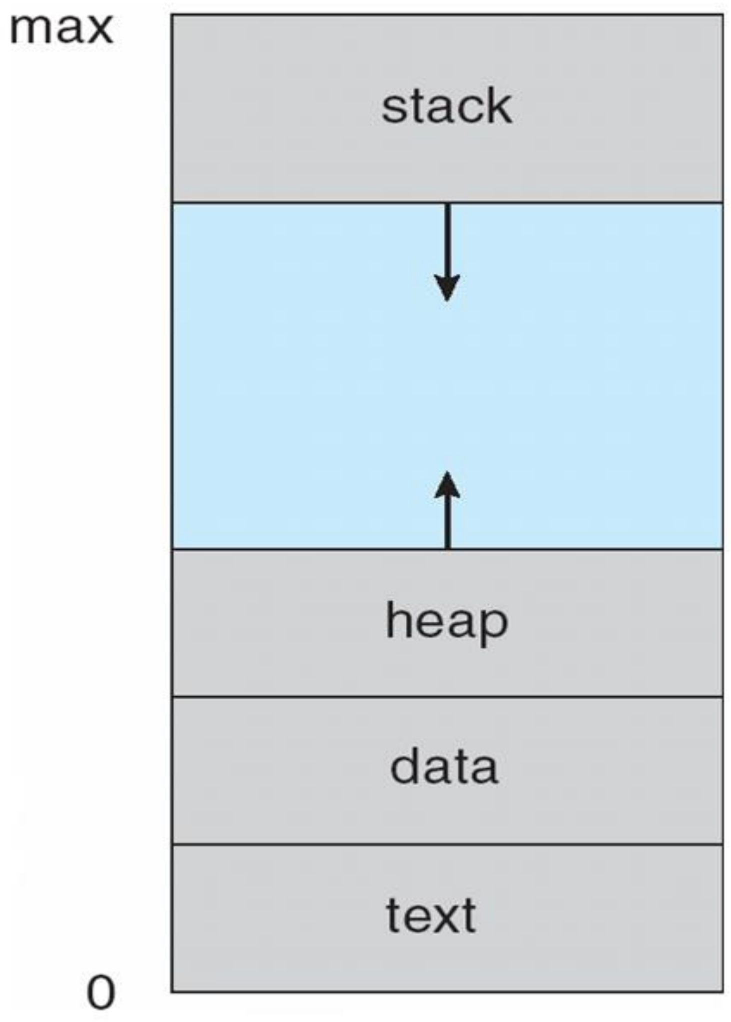
Process

Process Component

- The program code, also called **text section**
- Current activity including **program counter**, registers
- **Stack** containing temporary data
  - Function parameters, return addresses, local variables
- **Data section** containing global variables
- **Heap** containing memory dynamically allocated during run time

Process in Memory

- Program is passive entity, process is active
  - Program becomes process when executable file loaded into memory

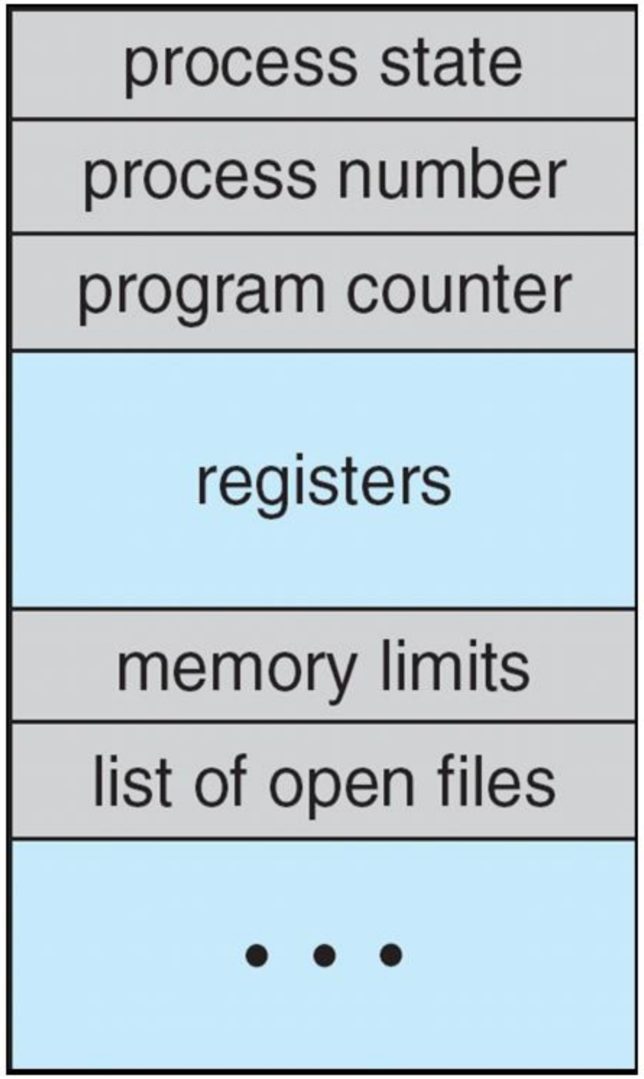


Process Control Block (PCB)

- **Information associated with each process (also called task control block)**
- Repository of any information related to Process.
- PCB components:
    - Identifiers
    - Processor State Information
    - Process Control Information

Process Control Block (PCB)

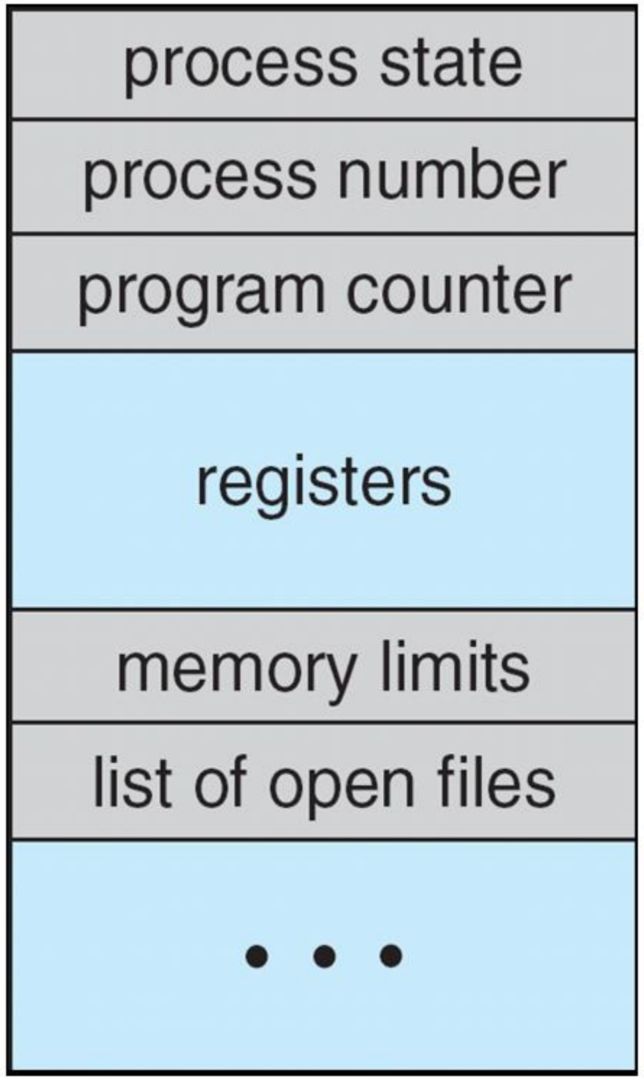
- **Process Identifier/Number -** An identifier that helps us in identifying and locating a process.
- **Process state** –It identifies the state that the process is currently in. It could be running, waiting, etc
- **Program counter** – address of the next instruction to be executed for this process.



Process Control Block (PCB)

CPU registers –

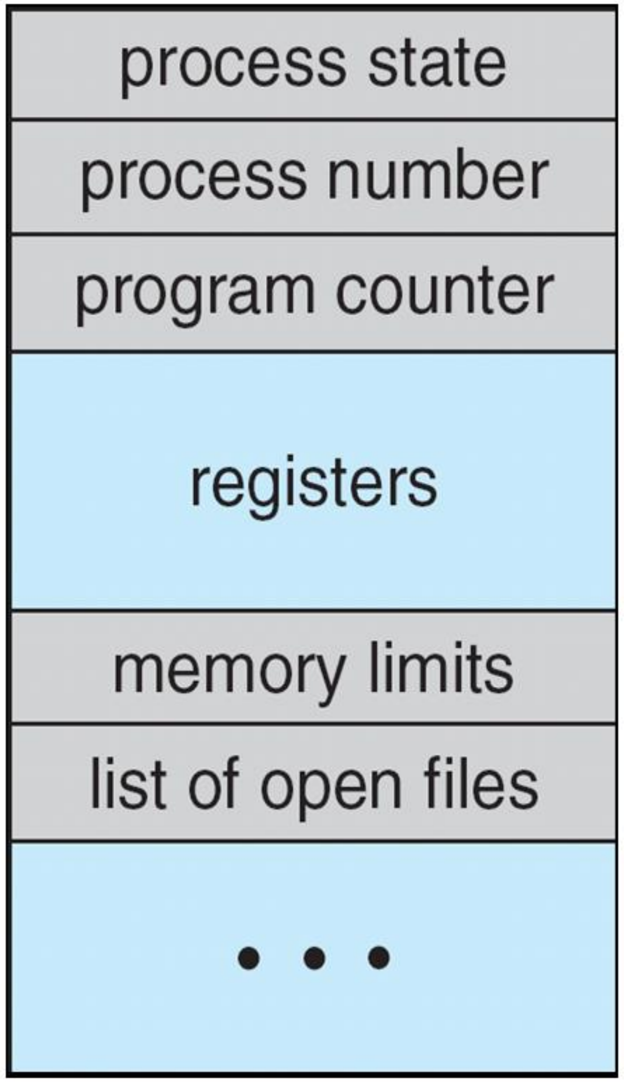
- contents of all process-centric registers,
- **Registers may vary in number and type depending on system architecture.**



Process Control Block (PCB)

CPU scheduling information-

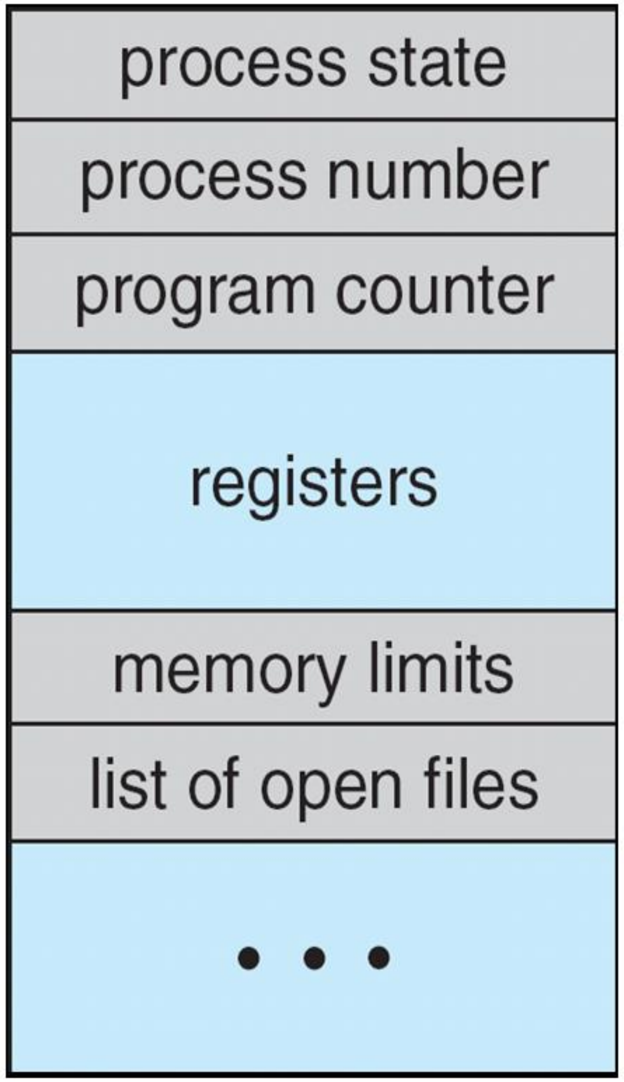
- Process priorities,
- scheduling queue pointers,
- any other scheduling parameters.



Process Control Block (PCB)

Memory-management information –

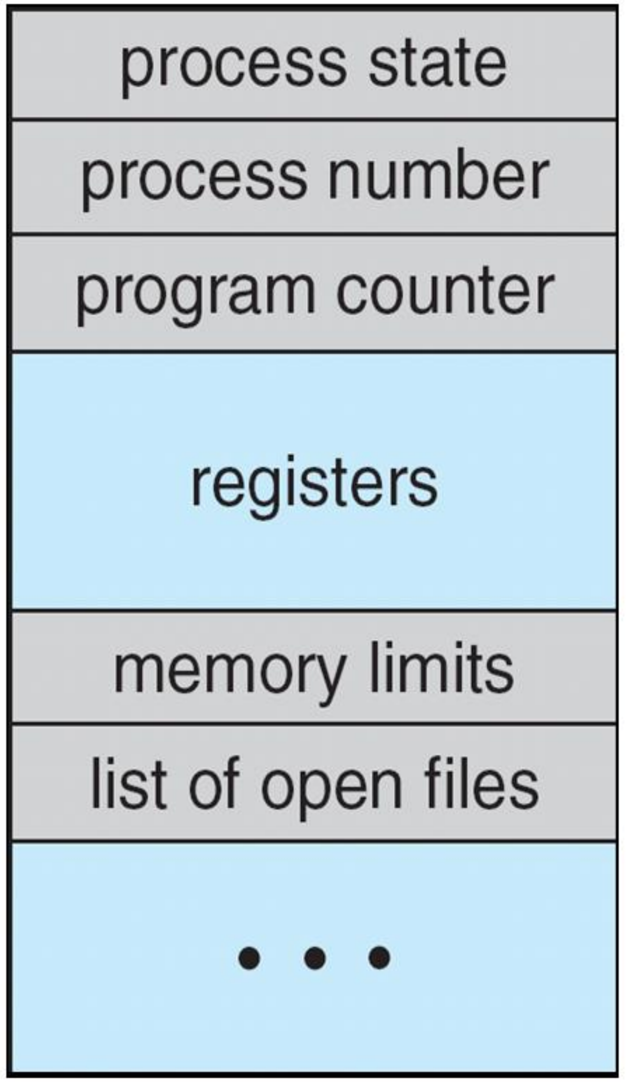
- memory allocated to the process,
- value of the Base and limit registers,
- the page tables or segment tables.



Process Control Block (PCB)

Accounting information –

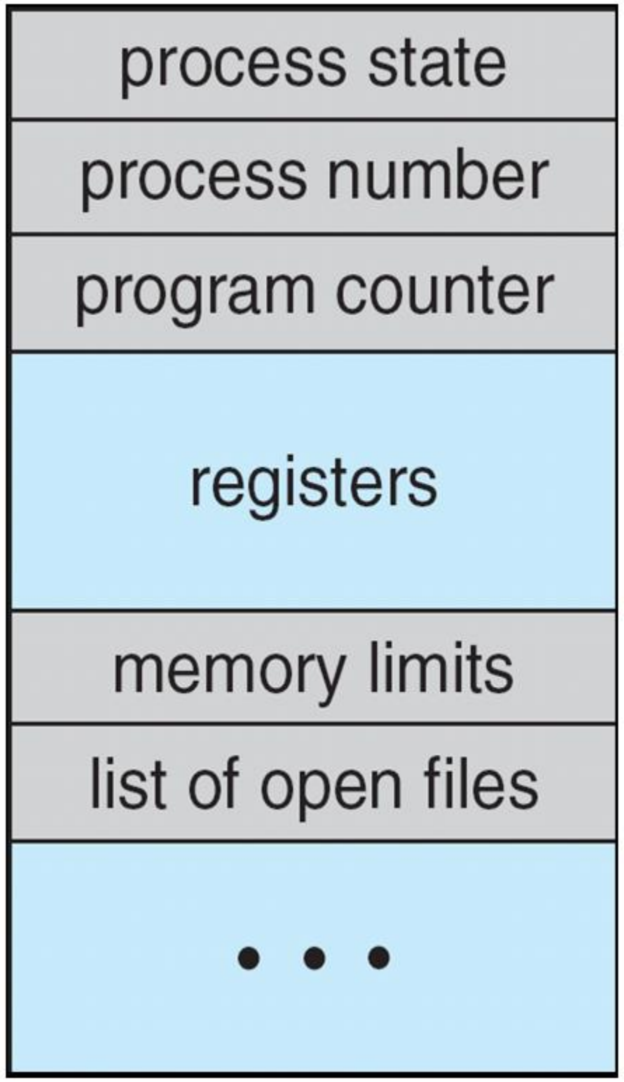
- CPU used,
- clock time elapsed since start, time limits,
- job or process numbers



Process Control Block (PCB)

I/O status information –

- list of I/O devices allocated to process,
- list of open files



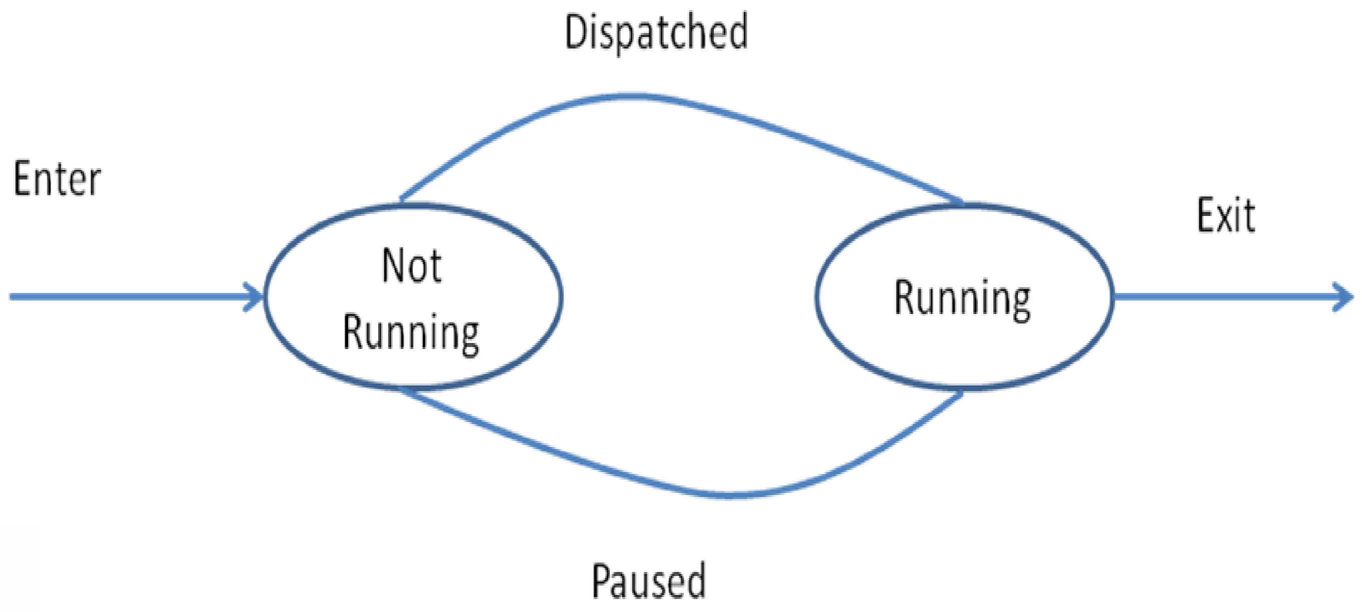
Process State

As a process executes, it changes *state*

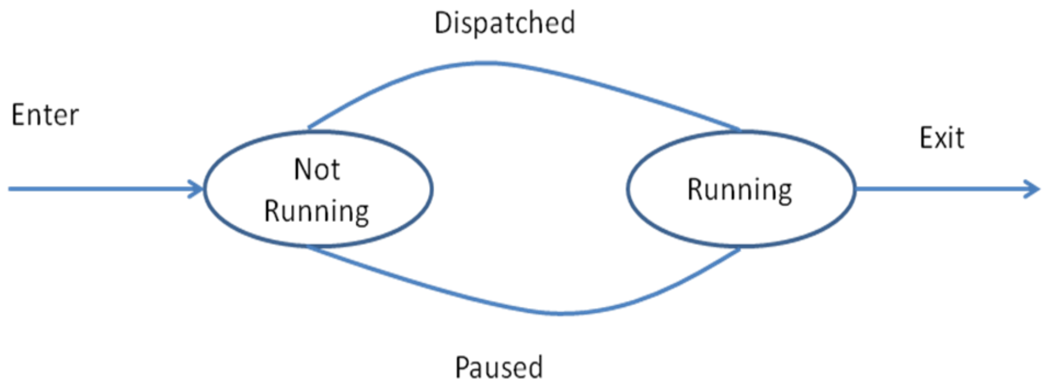
- New
  - When process creation is taking place, the process is in a new state.
- Ready
  - During this state, the process is loaded into the main memory and will be placed in the queue of processes **which are waiting for the CPU allocation**.
- Running
  - Instruction is being executed.
- waiting
  - The process is waiting for some event to occur (such as an I/O completion)
- Terminated
  - The process has finished execution.

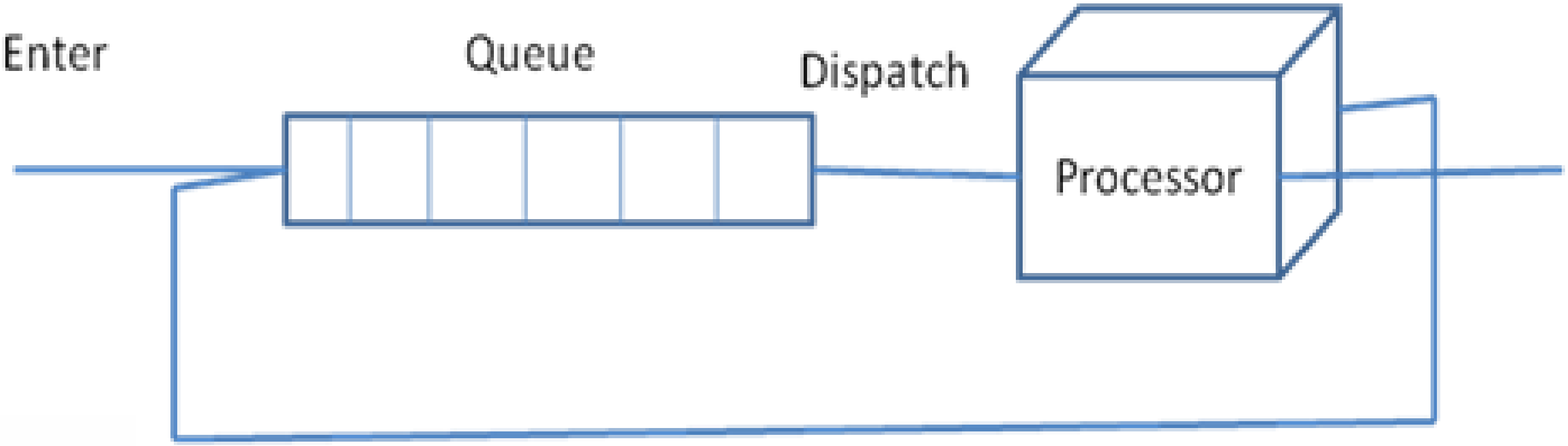
Process state transition

- A state transition for process is defined as a change in its state.
- The state transition occurs in response to some event in the computing system.
- E.g. Two State Model - State transition diagram

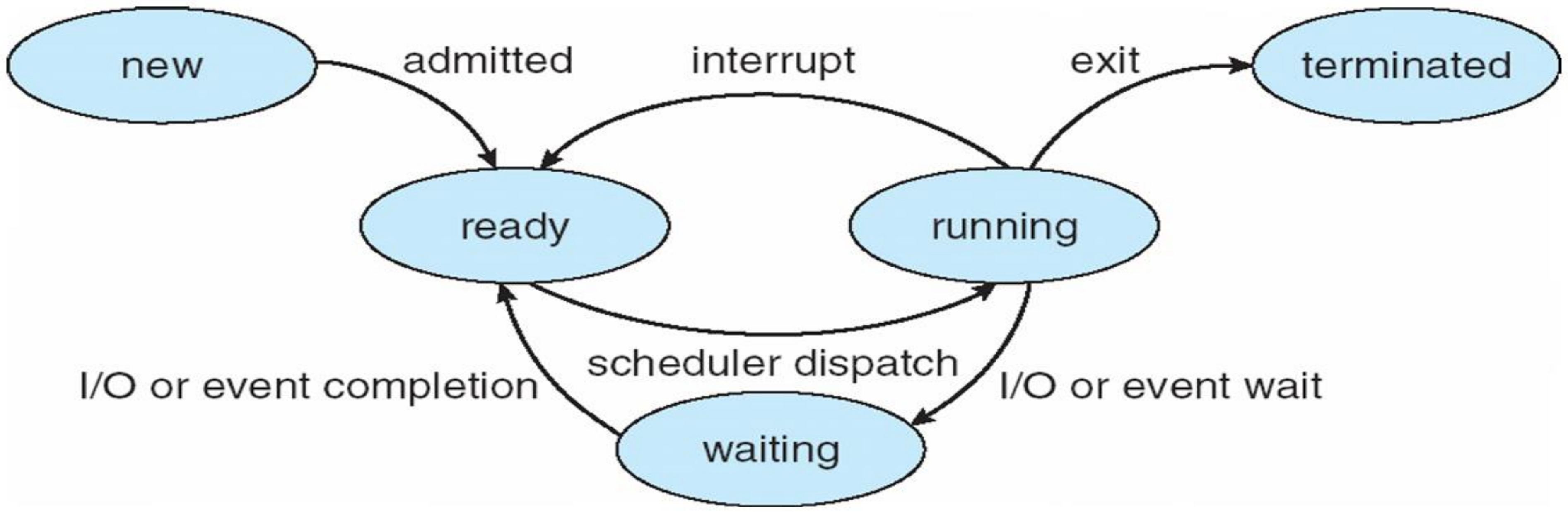


Two State Model – Queuing Diagram

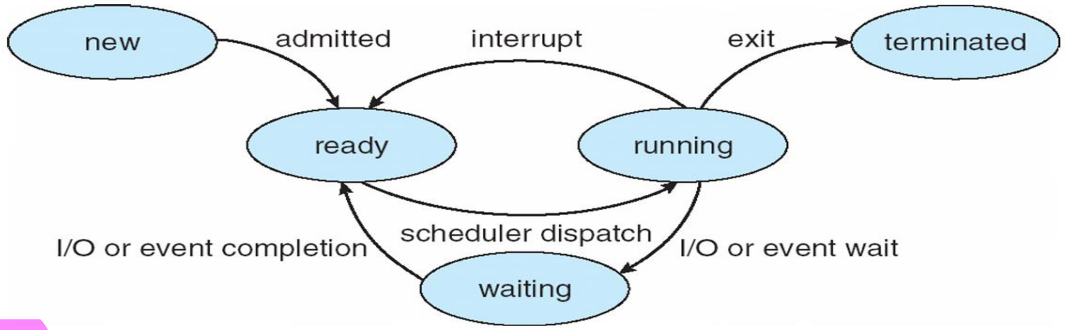




Five State Model

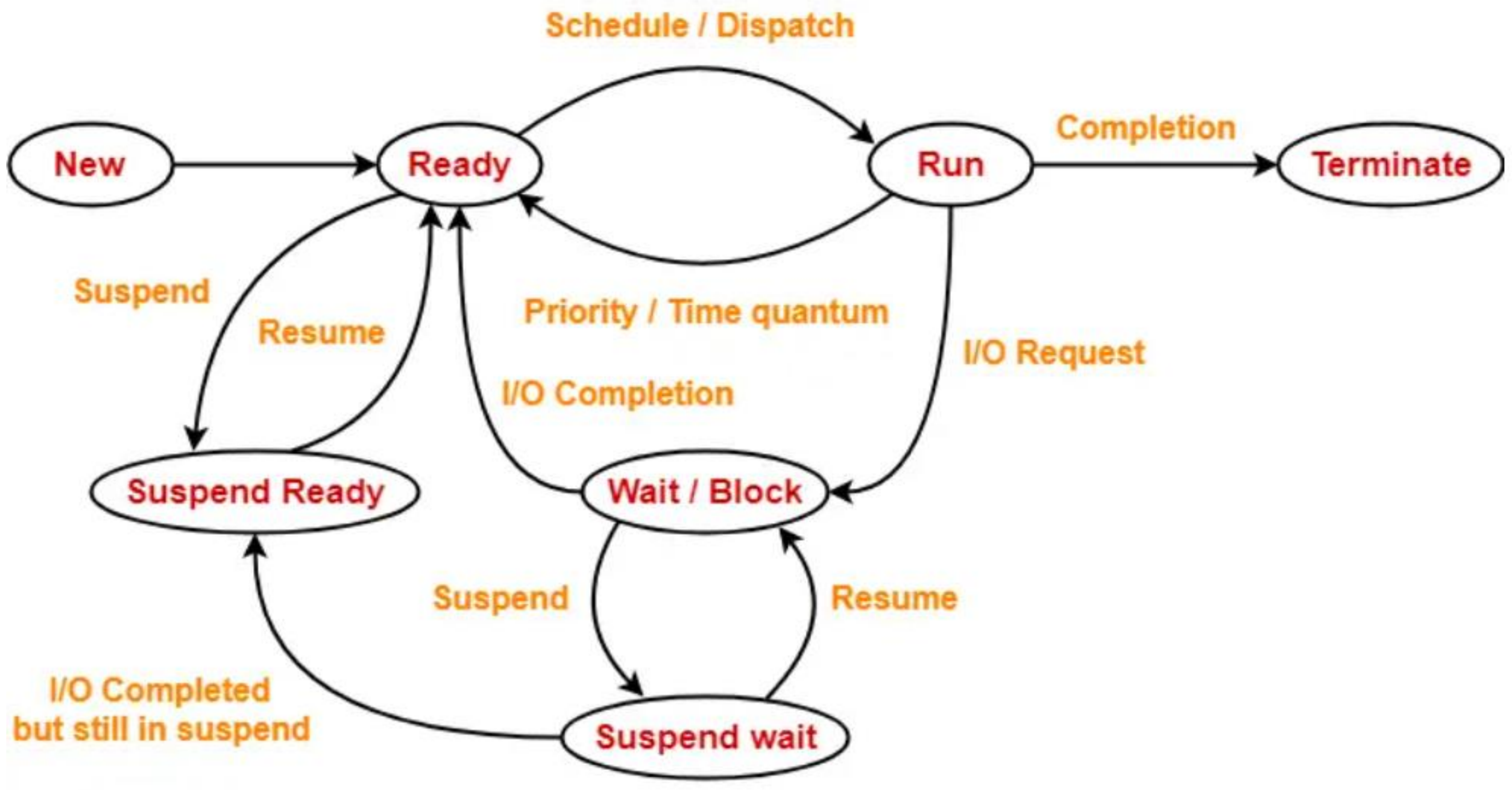


Five State Model



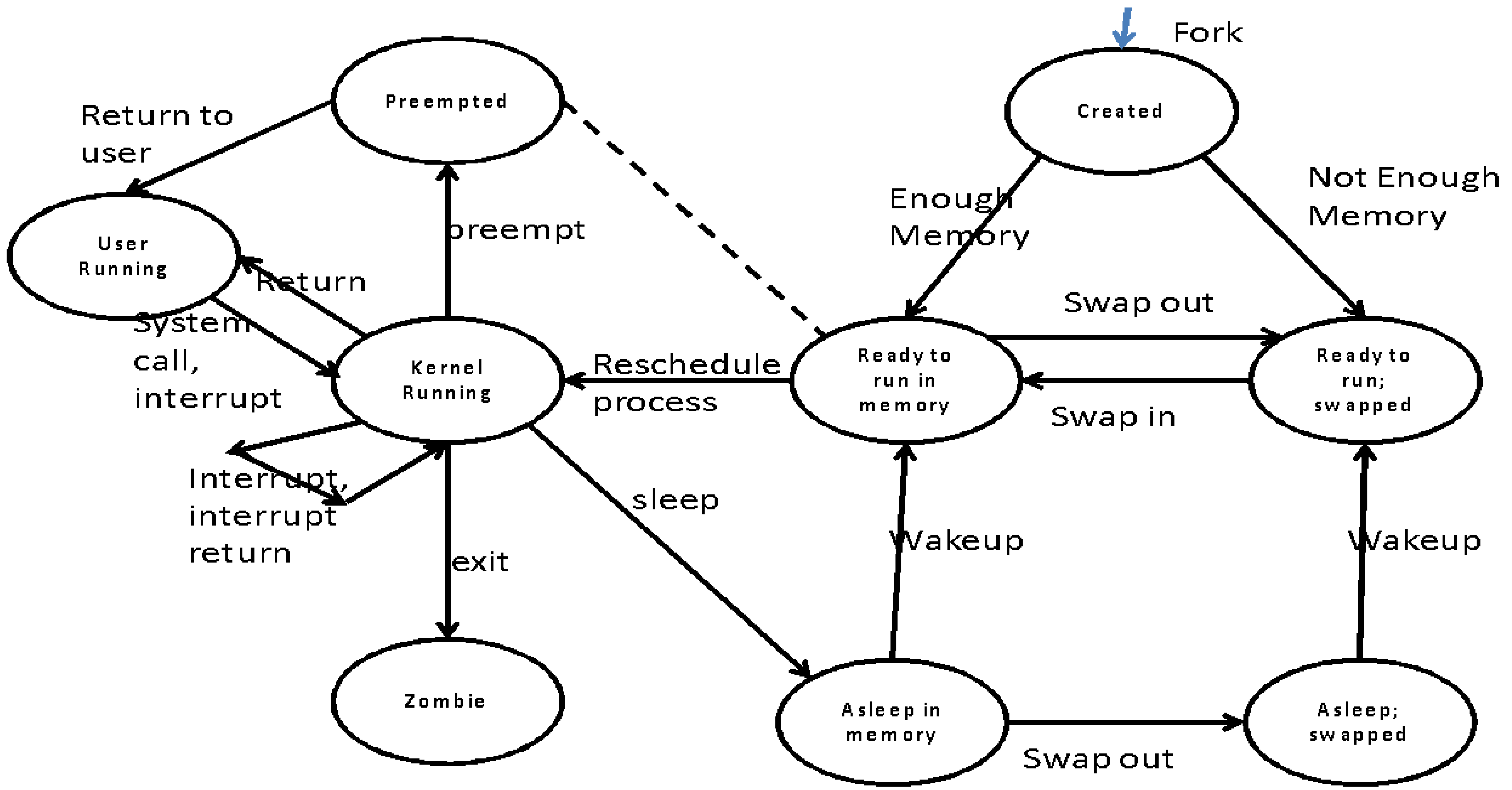
- Need of swapping (Suspend State)
  - because blocked processes hog up the main memory.
  - Swap them out from main memory to disk.
- There are two independent concepts
  - whether a process is waiting on an event (blocked or not),
  - whether a process has been swapped out of main memory (suspended or not)

7-state process model



Process State Diagram

9-state process model



State Transition

State transition	Description
Null□ New	A new process is created to execute a program.
New□ ready	The OS may move a process from New to Ready state depending on the predefined maximum number of processes allowed (Degree of multiprogramming).
Ready□ Running	The process is scheduled by dispatcher. The CPU starts or resumes execution of the instruction codes
Blocked □ Ready	The request initiated by process is satisfied or the event on which it is waiting occurs.

State Transition (cont...)

State transition	Description
Running Ready	The process is preempted by the OS decides to execute some other process. This transition takes place may be because of expiration of time quantum or arrival of high priority process.
Running Blocked	The running process makes request for resource(s) or needs some event to occur to proceed further. The process then calls for a system call to indicate its wish to wait till the resource or the event becomes available.
Running Termination	The program execution is completed or terminated.

State Transition (cont…)

State transition	Description
Running □ Blocked	The running process makes request for resource(s) or needs some event to occur to proceed further. The process then calls for a system call to indicate its wish to wait till the resource or the event becomes available.
Running □ Termination	The program execution is completed or terminated.

Causes of process initiation

- **Interactive logon:** when a user logs into the system, a new process is created.
- **Created by OS to provide some service:** The OS initiates a process to perform the service requested by user directly or indirectly, without making the user to wait.
- **Spawned by an existing process**: To support Modularity and/or parallelism, a user program can create some number of new processes.
- **Program Execution**: Whenever you open an application, the OS creates a process to run it

How the OS creates a process?

1. Create a process
2. Assign a unique process ID to newly created process
3. Allocate the memory and create its process image
4. Initialize process control block
5. Set the appropriate linkages to the different data structures such as ready queue etc.
6. Create or expand the other data structures if required

Causes of process blocking

- Process requests an I/O operation
- Process requests memory or some other resource
- Process wishes to wait for a specific interval of time
- Process waits for message from some other process
- Process wishes to wait for some action to be performed by another process.

Causes of process termination

- **Normal Completion:** The process executes an OS system call to intimate that it has completed its execution.
- **Self termination** (e.g. incorrect file access privileges, inconsistent data)
- **Termination by the parent process:** a parent process calls a system call to kill/terminate its child process when the execution of child process is no longer necessary.
- **Exceeding resource utilization:** An OS may terminate a process if it is holding resources more than it is allowed to. This step can also be taken as part of deadlock recovery procedure.

Causes of process termination (cont...)

- **Abnormal conditions during execution:** the OS may terminate a process if an abnormal condition occurs during the program execution. (e.g. memory protection violation, arithmetic overflow etc)
- **Deadlock detection and recovery**

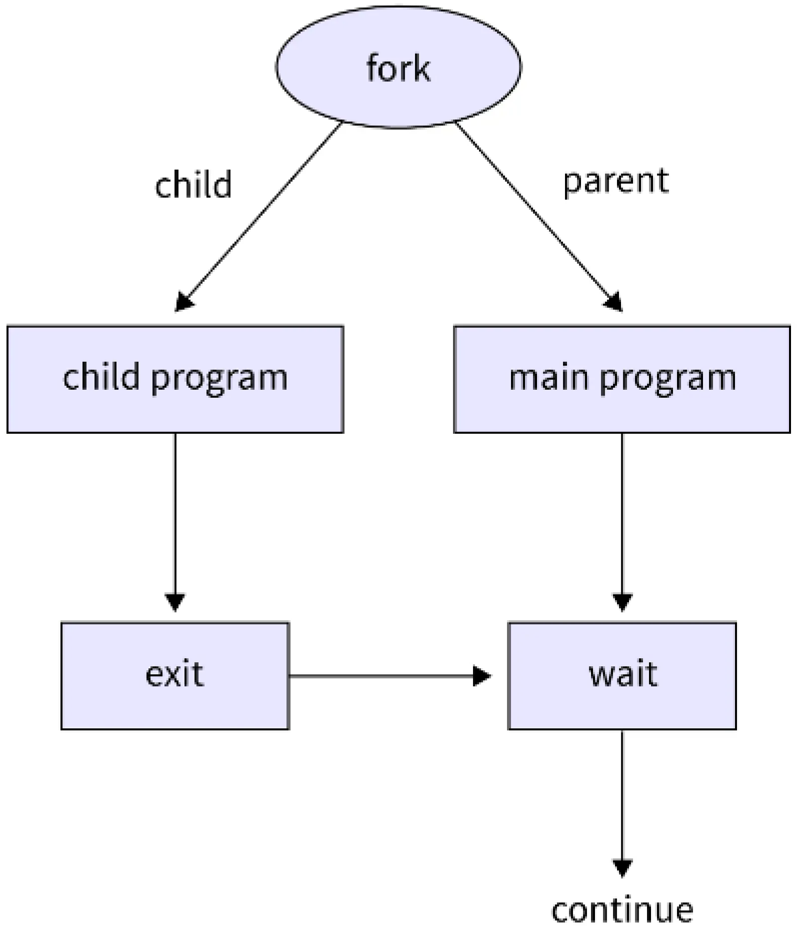
Process Creation in UNIX

- One process can create another process, perhaps to do some work for it.
  - The original process is called the ***parent***
  - The new process is called the ***child***
  - The child is an (almost) identical **copy** of parent (same code, same data, etc.)
  - The parent can either wait for the child to complete, or continue executing in parallel (concurrently) with the child

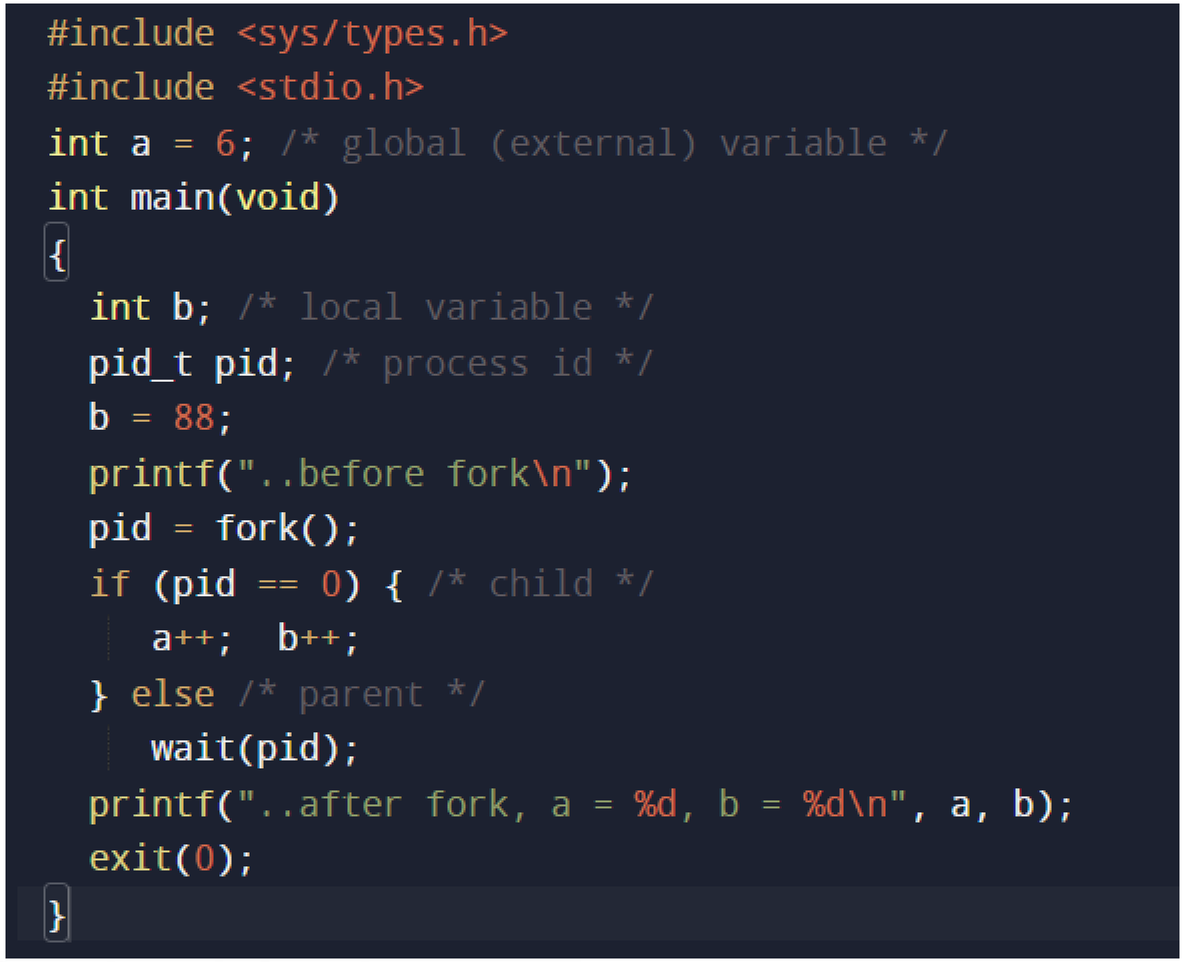
Process Creation in UNIX (cont...)

- In UNIX, a process creates a child process using the system call fork( )
  - **Negative:** A child process could not be successfully created if the fork() returns a **negative** value.
  - **Zero:** A new child process is successfully created if the fork() returns a **zero**.
  - **Positive:** The positive value is the process ID of a child's process to the parent. The process ID is the type of pid_t that is defined in OS or sys/types.h.
- Child often uses exec( ) to start another completely different program

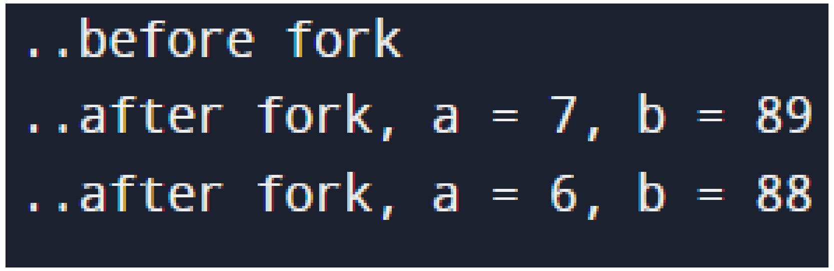
Fork()


fork
child
parent
child program
main program
exit
wait
continue

Example - Process Creation



```c
#include <sys/types.h>
#include <stdio.h>
int a = 6; /* global (external) variable */
int main(void)
{
  int b; /* local variable */
  pid_t pid; /* process id */
  b = 88;
  printf("..before fork\n");
  pid = fork();
  if (pid == 0) { /* child */
     a++;  b++;
  } else /* parent */
     wait(pid);
  printf("..after fork, a = %d, b = %d\n", a, b);
  exit(0);
}
```



```
..before fork
..after fork, a = 7, b = 89
..after fork, a = 6, b = 88
```

System Calls for Process Management

Sr No	System Call	Description
1	fork()	This system calls creates a new process.
2	exec()	This call is used to execute a new program on a process.
3	wait()	This call makes a process wait until some event occurs.
4	exit()	This call makes a process to terminate
5	getpid()	This system call helps to get the identifier associated with the process.
6	getppid()	This system call helps to get the identifier associated with the parent process.
7	nice()	The current process priority can be changed with execution of this system call.
8	brk()	This call helps to increase or decrease the data segment size of the process.
9	Kill()	The forced termination of any process can be executed with this system call.
10	Signal()	This system call is invoked for sending and receiving software interrupts

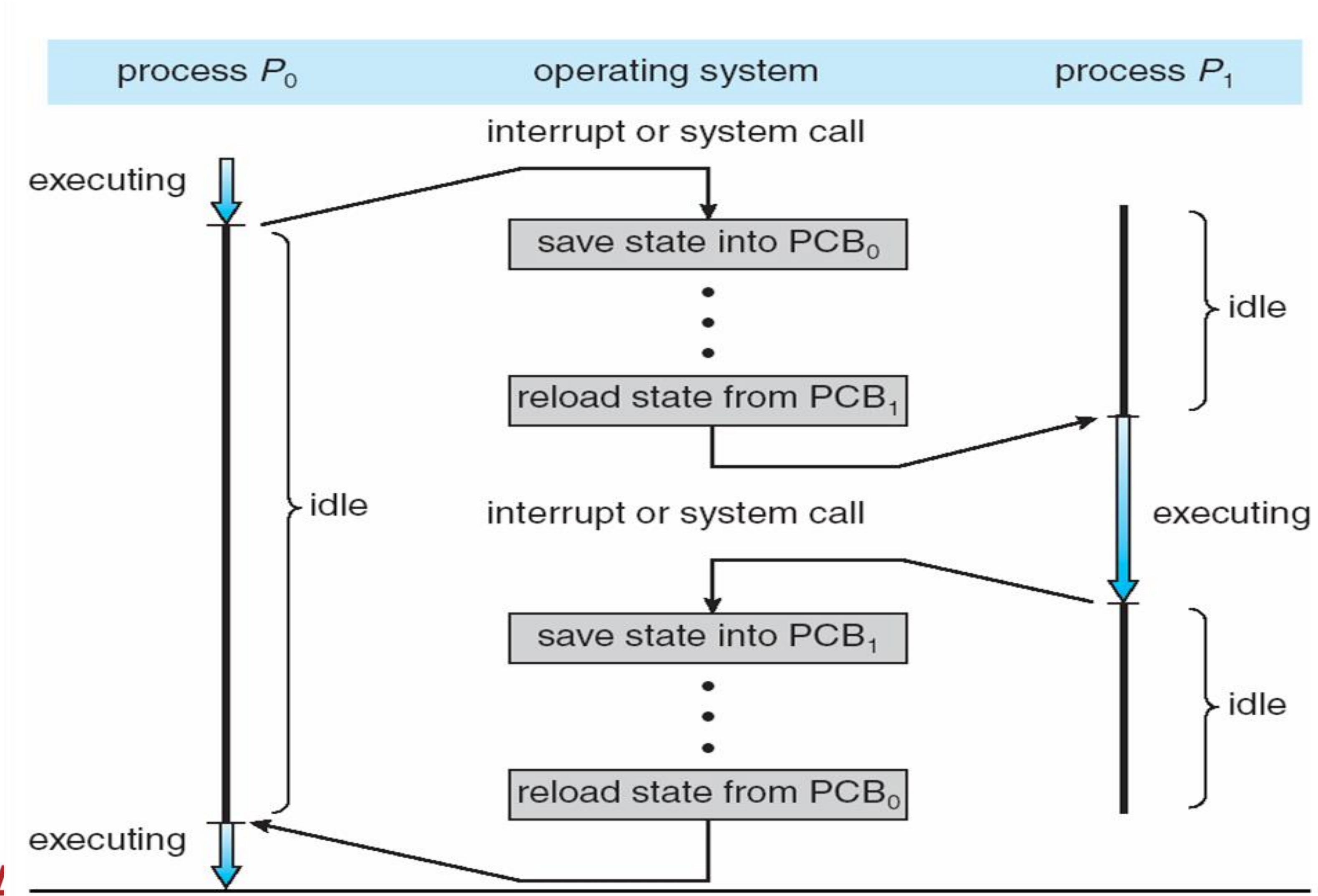
Context Switch

- Stopping one process and starting another is called a context switch
- When the OS stops a process, it stores the hardware registers (PC, SP, etc.) and any other state information in that process' PCB
- When OS is ready to execute a waiting process, it loads the hardware registers (PC, SP, etc.) with the values stored in the new process' PCB, and restores any other state information

Process context switch Vs mode switch

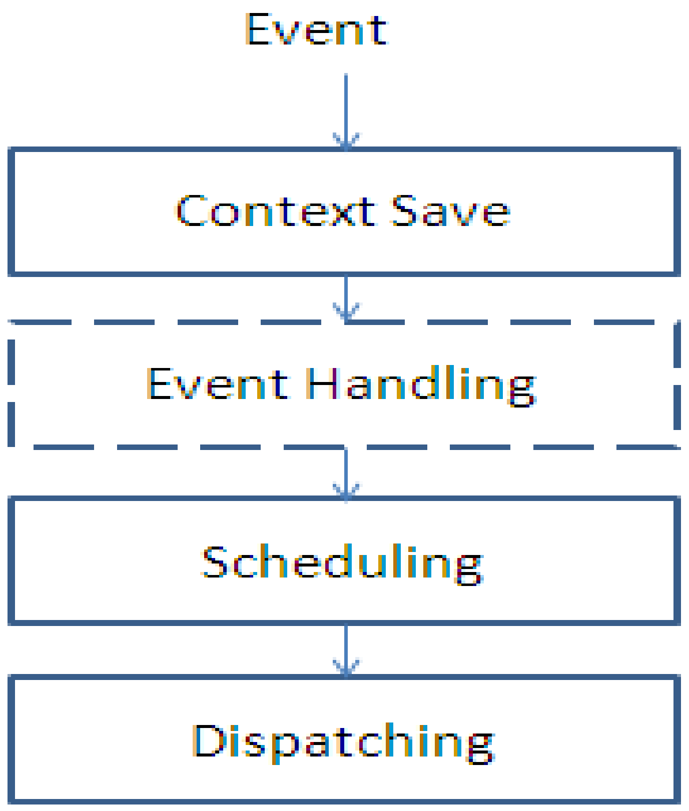
- Context Switch:
  - Execution of a process is stopped to respond an interrupt.
  - Needs to save Process Image of one process and load process image of the new process loaded.
  - The processes are switched and processes keep on changing their status as Running and Not running.
- Mode switch:
  - Every process may switch in between a low privileged user mode and high privileged kernel mode in its lifetime.
  - Process continues to execute even after mode switches.

CPU Switch From Process to Process



Fundamental kernel functions of process control

- Context save: Save information concerning an executing process when its execution gets suspended
- Scheduling: Choose the process as per the scheduling policy to be executed next on the CPU.
- Dispatching: Set up execution of the chosen process on the CPU.



Fundamental kernel functions of process control

- Occurrence of the event calls the context save functionality and an appropriate event handling procedure.
- Event handling may initiate some processes, hence the scheduling function gets invoked to choose the process and in turn,
- The dispatching function transfers control to the new process.

Control/Data structures maintained by OS to manage processes

- Memory Tables
- I/O Tables
- File Tables
- Process table

Control structures maintained by OS to manage processes

- **Memory Tables:**
  - Memory tables keep track of both main and secondary memory.
  - Active processes are stored in main memory and when required, they are moved to secondary memory through the mechanism called 'swapping'.
  - The memory tables maintain the following information:
    - The main memory allocation to all processes in system
    - The secondary memory allocation to all processes in system
    - Shared memory regions in main and virtual memory and their attributes
    - Miscellaneous information required to manage virtual memory.

Control structures maintained by OS to manage processes

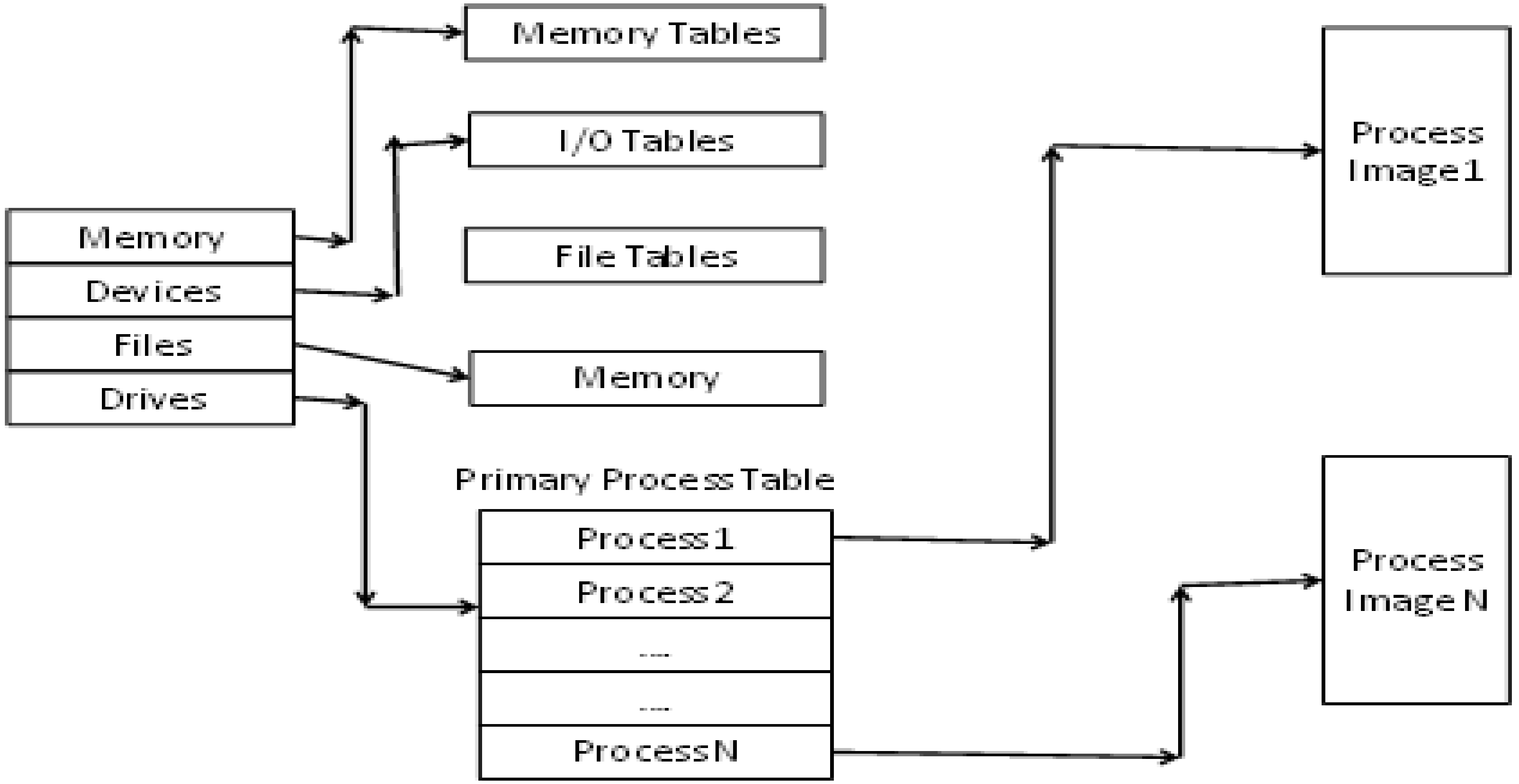
- **I/O Tables:**
  - I/O tables keep track of I/O devices and channels in the computing system.
  - The I/O devices are also resources required by processes.
  - So at any given instance, I/O devices may be available or allocated to a particular process.

Control structures maintained by OS to manage processes

- **File Tables:**
  - File tables keeps track of;
    - all files,
    - their locations on secondary memory,
    - their current statuses and
    - other attributes such security, sharing, etc.
  - Most of the operating systems, this information is maintained by a module called File Management System.

Control structures maintained by OS to manage processes

- **Process table:**
  - Process tables manage processes.
  - They maintain information of:
    - processes,
    - their child process references,
    - statuses,
    - allocated resources,
    - process contexts,
    - information required for process synchronization and so on.
  - These pieces of information are stored in process images.


Memory Tables
I/O Tables
File Tables
Memory
Memory
Devices
Files
Drives
Primary Process Table
Process 1
Process 2
....
....
Process N
Process Image 1
Process Image N

Different interaction mechanisms used by processes

Interaction Mechanism	Description
Data Sharing	The processes interact with each other by altering data values. If more than one processes update the data the same time, they may leave the shared in inconsistent state. So, shared data items are protected against simultaneous access to avoid such situation.
Message Passing	In this mechanism, the processes exchange information by sending messages to each other.
Synchronization	In certain computing environments, the processes are required to execute their actions in some particular order. To help this happen, the processes synchronize with each other to maintain their relative timings and execute in the desired sequence.
Signals	The processes may wait for events to occur. It can be intimated to processes through the signaling mechanism.

Question ?