Process Concept & Scheduling

Nirmala Shinde Baloorkar
Assistant Professor
Department of Computer Engineering

Concept Covered

2.1 Process: Concept of a Process, Process States, Process Description, Process Control Block, Operations on Processes.

2.2 Scheduling: Uniprocessor Scheduling - Types of Scheduling: Preemptive and, Non-preemptive, Scheduling Algorithms: FCFS, SJF, SRTN, Priority based, Round Robin, Multilevel Queue scheduling

2.3 Thread: Introduction to thread, Multithreading models

Outline

- Introduction to process
- Process state
- Process control block
- Operation on process

Task Manager

Type a name, publisher, or PID to search

Processes

Run new task | End task | Efficiency mode

Name	Type	Status	5% CPU	42% Memory	0% Disk	0% Network
Google Chrome (14)	App		1.3%	550.5 MB	0.1 MB/s	0 Mbps
Task Manager	App		1.9%	66.4 MB	0.1 MB/s	0 Mbps
Start (2)	Background process		0%	48.3 MB	0 MB/s	0 Mbps
Runtime Broker	Background process		0%	4.2 MB	0 MB/s	0 Mbps
Phone Link	Background process		0%	41.3 MB	0 MB/s	0 Mbps
Microsoft Edge (12)	Background process		0%	191.5 MB	0 MB/s	0 Mbps
Microsoft Windows Search Filt...	Background process		0%	1.3 MB	0 MB/s	0 Mbps
Microsoft Windows Search Pr...	Background process		0%	1.9 MB	0 MB/s	0 Mbps
System Settings Broker	Background process		0%	7.0 MB	0 MB/s	0 Mbps
Runtime Broker	Background process		0%	4.0 MB	0 MB/s	0 Mbps
Adobe Acrobat SpeedLaunche...	Background process		0%	1.3 MB	0 MB/s	0 Mbps
Windows Driver Foundation - ...	Background process		0%	1.3 MB	0 MB/s	0 Mbps
Windows Security notification ...	Background process		0%	0.9 MB	0 MB/s	0 Mbps
COM Surrogate	Background process		0%	2.6 MB	0 MB/s	0 Mbps
SECOCL.exe	Background process		0%	4.2 MB	0 MB/s	0 Mbps
Microsoft Edge Update (32 bit)	Background process		0%	0.6 MB	0 MB/s	0 Mbps
HP Hotkey UWP Service	Background process		0%	1.4 MB	0 MB/s	0 Mbps
Intel(R) Innovation Platform Fr...	Background process		0%	1.2 MB	0 MB/s	0 Mbps
Usermode Font Driver Host	Background process		0%	1.4 MB	0 MB/s	0 Mbps

Top Stories
Marco Rubio Se...
Search
ENG IN
09:39
22-01-2025

Process Concept

- In nearly every computer, the resource that is most often requested is the CPU or processor. Many computers have only one processor, so this processor must be shared among all the programs that need to execute on the computer.
- An operating system executes a variety of programs:
  - Batch system – jobs
  - Time-shared systems – user programs or tasks
- The term "process" was first used by the designers of the MULTICS in 1960's.
- Since then, the term process, used somewhat interchangeably with 'task' or 'job'.

Process Concept (continued)

```
void X (int b) {
    if(b == 1) {
…
int main() {
  int a = 2;
  X(a);}
```

- Imagine we've written a program called **abc.c** in C.
- Initially, this program is just a script, a static text file with no dynamic behavior.
- It can't process input or produce output on its own.
- However, once we compile the program and execute it.
- The script is transformed into an active entity that can engage the processor for computations and perform input/output operations.
- Essentially, a program is a passive script until it is compiled and executed, at which point it becomes an active process capable of performing its intended tasks.

Process Concept (continued)

```
void X (int b) {
   if(b == 1) {
…
int main() {
  int a = 2;
  X(a);}
```

Program in C

Compiler

a.out

- A program is representation of an algorithm in some programming language; i.e. it is **static**.
- A process refers to the activity performed by a computer when executing program; i.e. it is **dynamic**

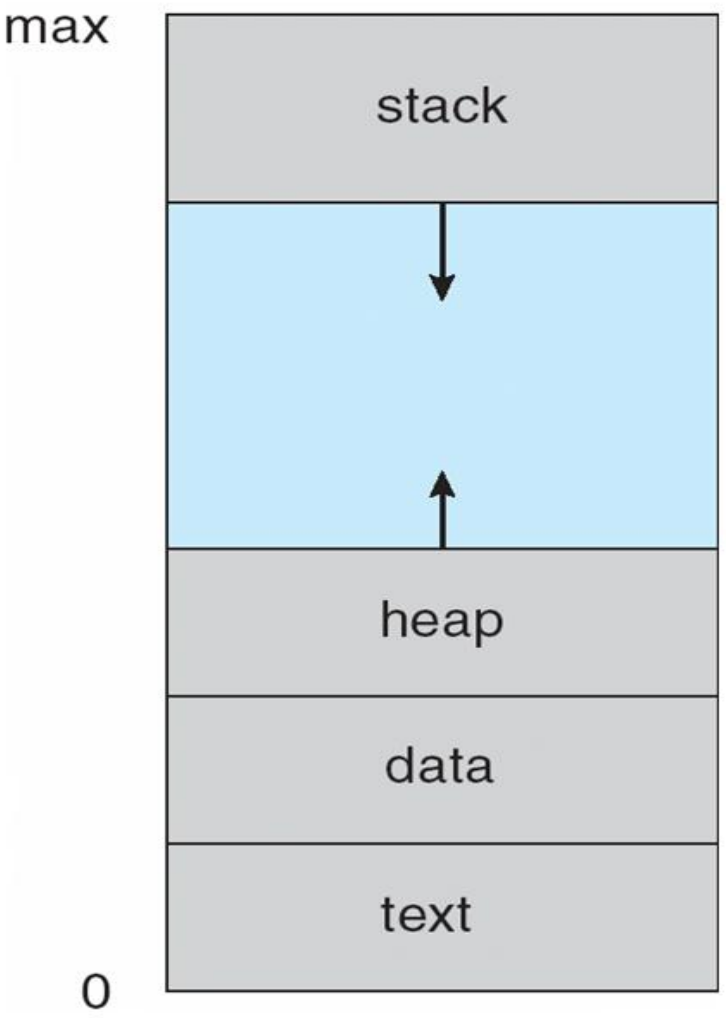
- A process is an executable entity – it's a program in execution
  - When we compile a C language program we get an a.out file which is an executable file.
  - When we seek to run this file – we see the program in execution.
- A process is created when a program or command is executed.
- Every process has its instruction sequence.
- Therefore, at any point in time there is a current instruction in execution.

Process in Memory

- A process includes three segments/sections:
  - Program: code/text.
  - Data: global variables and heap
  - Heap contains memory dynamically allocated during run time.
  - Stack: temporary data
    - Procedure/Function parameters, return addresses, local variables.
- Current activity of a program includes its Context: program counter, state, processor registers, etc.
- One program can be several processes:
  - Multiple users executing the same Sequential program.
  - Concurrent program running several process.

Process in Memory (continued)

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

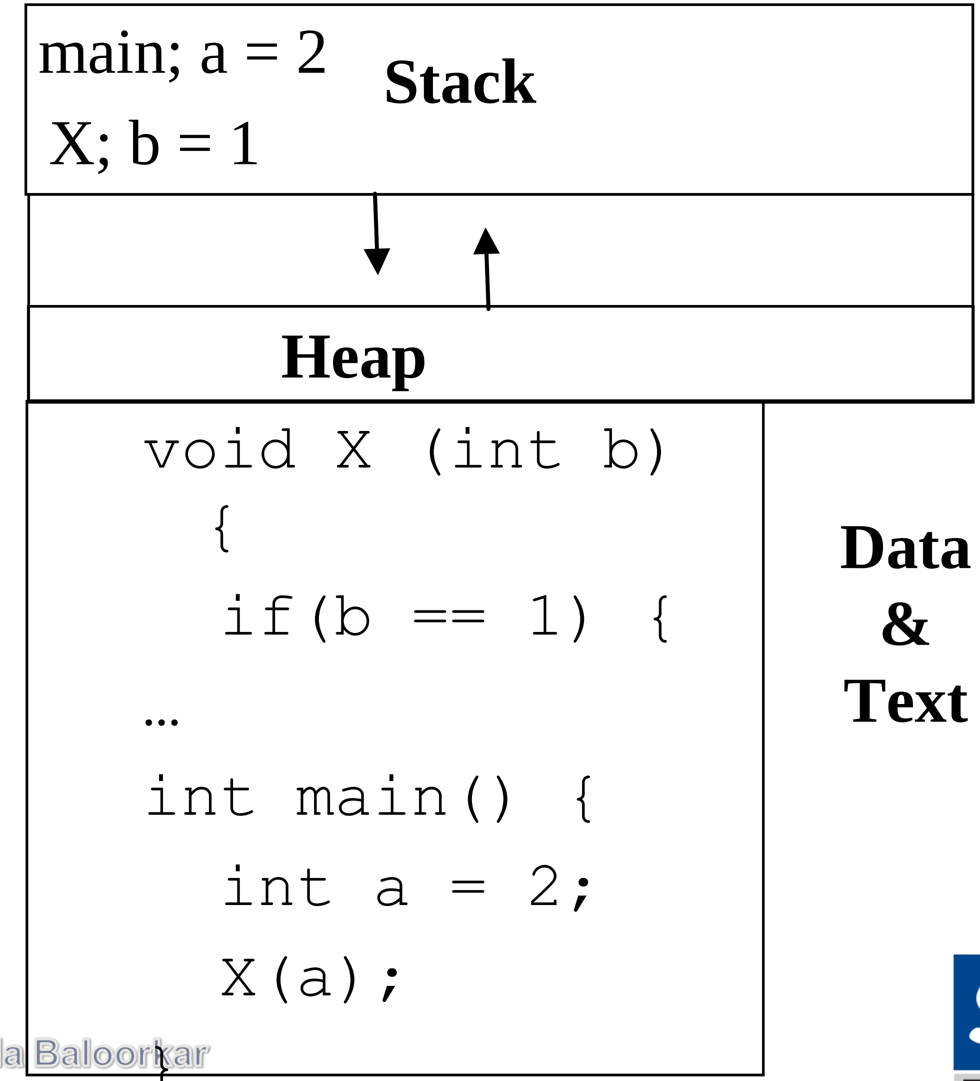


Process in Memory (continued)

- Program to process.
  - What you wrote

```
void X (int b) {
  if(b == 1) {
…
int main() {
  int a = 2;
  X(a);
}
```

- What is in memory.



Quiz

What is the typical output file generated when a C program is compiled in Windows?

A. program.exe

B. a.out

C. main.c

D. script.sh

Task Manager

Type a name, publisher, or PID to search

Services

Run new task | Start | Stop | Restart | Open Services | ...

Name	PID	Description	Status	Group
AarSvc		Agent Activation Runtime	Stopped	AarSvcGroup
AarSvc_818d46a		Agent Activation Runtime_818d46a	Stopped	AarSvcGroup
AJRouter		AllJoyn Router Service	Stopped	LocalServiceN...
ALG		Application Layer Gateway Service	Stopped	
AppIDSvc	2372	Application Identity	Running	LocalServiceN...
Appinfo	9500	Application Information	Running	netsvcs
AppMgmt	14008	Application Management	Running	netsvcs
AppReadiness		App Readiness	Stopped	AppReadiness
AppVClient		Microsoft App-V Client	Stopped	
AppXSvc	20464	AppX Deployment Service (AppXSVC)	Running	wsappx
AssignedAccessManager...		AssignedAccessManager Service	Stopped	AssignedAcce...
AudioEndpointBuilder	3768	Windows Audio Endpoint Builder	Running	LocalSystemN...
Audiosrv	4484	Windows Audio	Running	LocalServiceN...
autotimesvc		Cellular Time	Stopped	autoTimeSvc
AVP.KES.21.14	6248	Kaspersky Endpoint Security Service (KES.21.14)	Running	
avpsus.KES.21.14	8680	Kaspersky Seamless Update Service (KES.21.14)	Running	
AxInstSV		ActiveX Installer (AxInstSV)	Stopped	AxInstSVGroup
BcastDVRUserService		GameDVR and Broadcast User Service	Stopped	BcastDVRUser...
BcastDVRUserService_81...		GameDVR and Broadcast User Service_818d46a	Stopped	BcastDVRUser...
BDESVC		BitLocker Drive Encryption Service	Stopped	netsvcs
BFE	2956	Base Filtering Engine	Running	LocalServiceN...
BITS		Background Intelligent Transfer Service	Stopped	netsvcs
BluetoothUserService		Bluetooth User Support Service	Stopped	BthAppGroup
BluetoothUserService_81...		Bluetooth User Support Service_818d46a	Stopped	BthAppGroup
BrokerInfrastructure	1340	Background Tasks Infrastructure Service	Running	Dc
BTAGService		Bluetooth Audio Gateway Service	Stopped	Loc

Show desktop

SUNPHARMA +1.33%

Search

ENG IN

11:27 22-01-2025

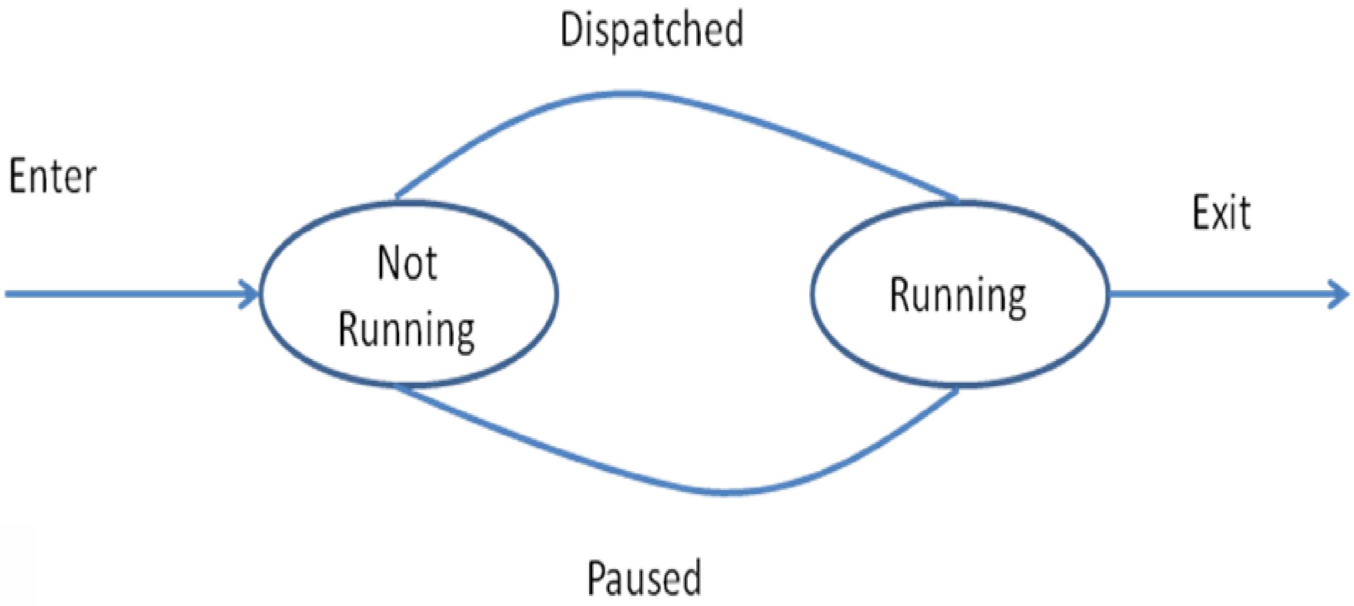
Process State

As a process executes, it changes *state*

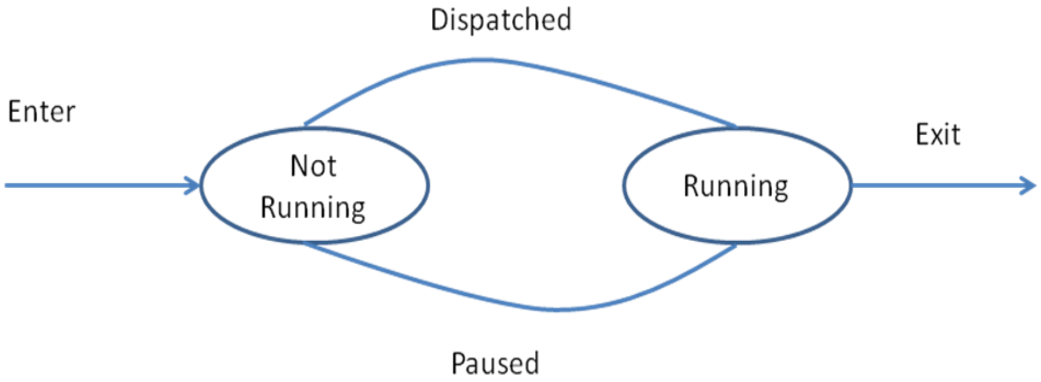
- New
  - When process creation is taking place, the process is in a new state.
- Ready
  - During this state, the process is loaded into the main memory and will be placed in the queue of processes **which are waiting for the CPU allocation**.
- Running
  - Instruction is being executed.
- waiting
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

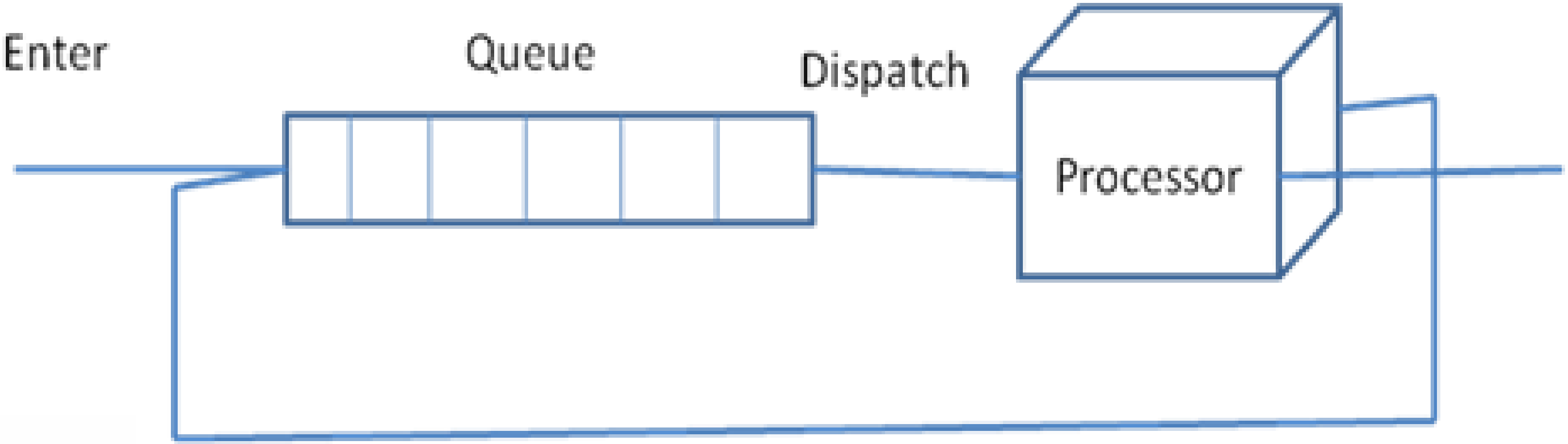
Process state transition

- 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

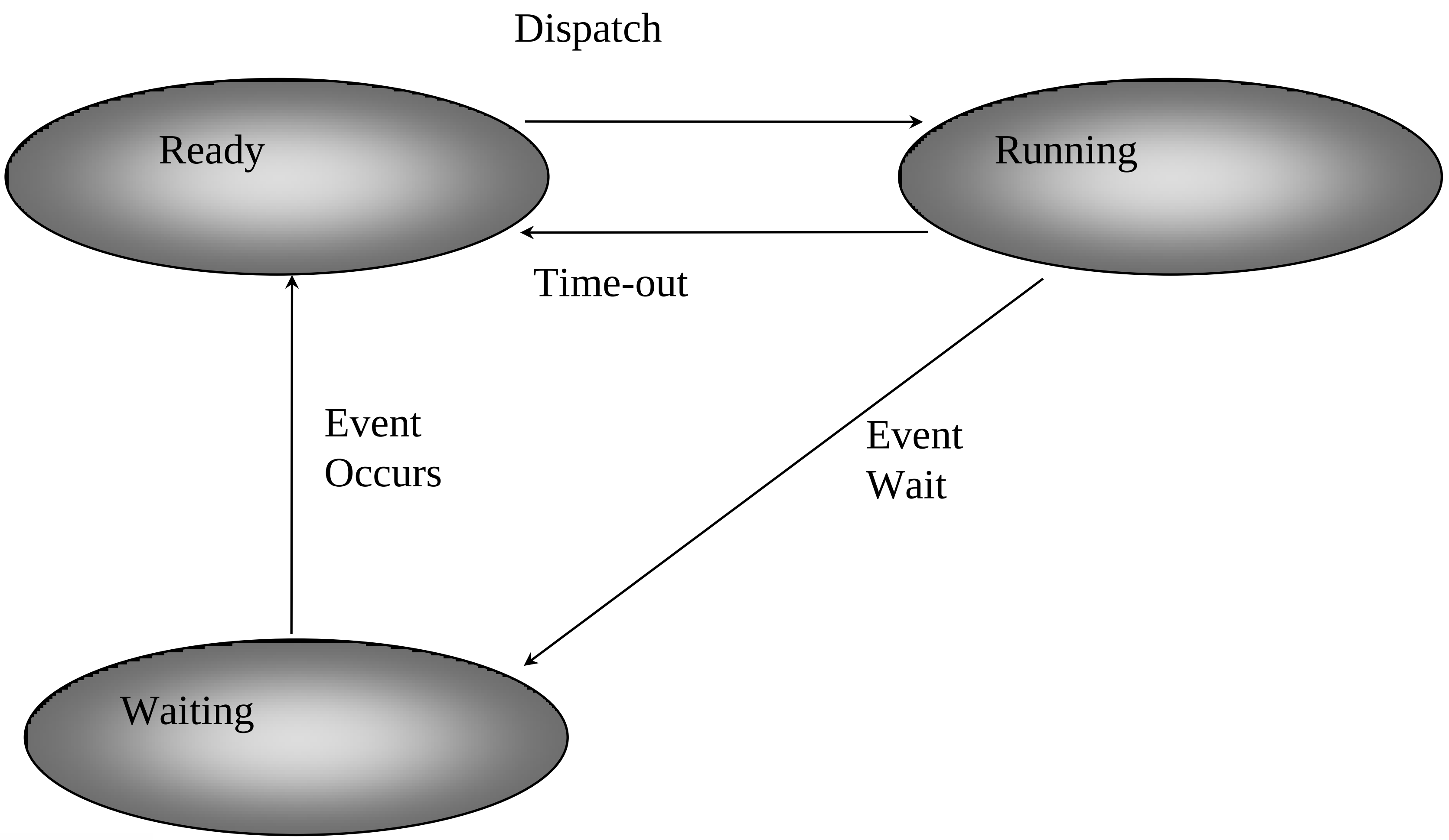


Two State Model – Queuing Diagram

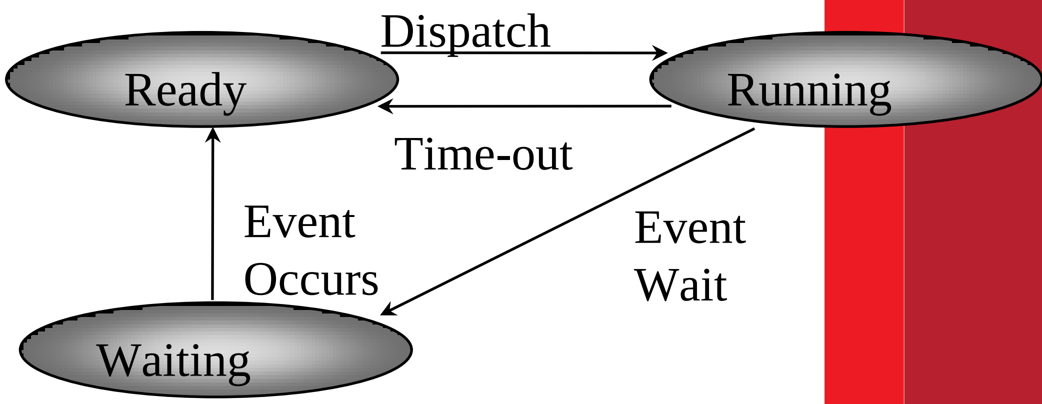




Three-state Process Model



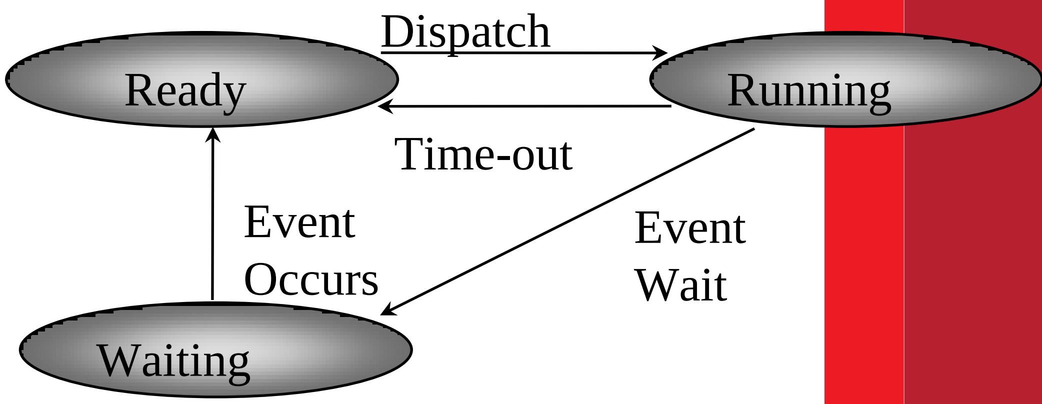
Three-state Process Model (continued)



- Ready –> Running
  - When it is time, the dispatcher selects a new process to run.

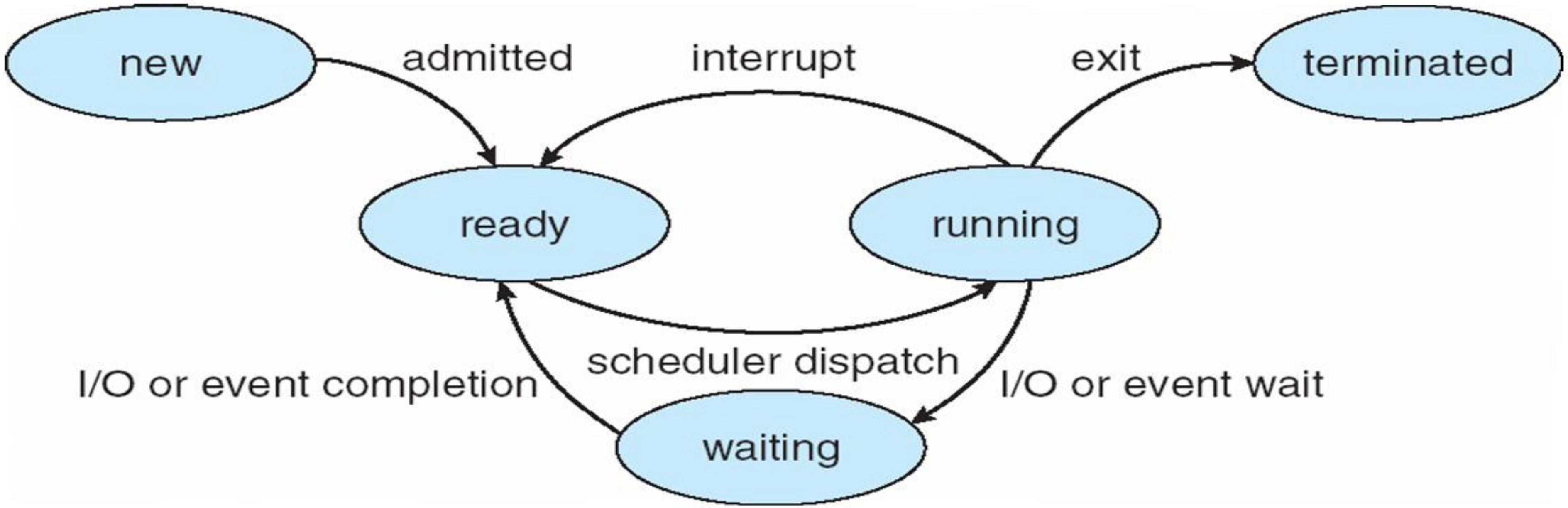
- Running –> Ready
  - the running process has expired his time slot.
  - the running process gets interrupted because a higher priority process is in the ready state

Three-state Process Model (continued)

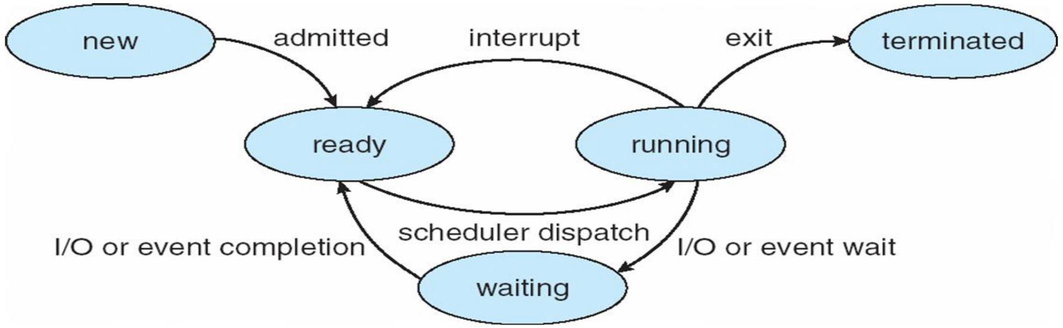


- Running –> Waiting
  - When a process requests something for which it must wait:
    - a service that the OS is not ready to perform.
    - an access to a resource not yet available.
    - initiates I/O and must wait for the result.
    - waiting for a process to provide input.
- Waiting –> Ready
  - When the event for which it was waiting occurs.

Five State Model

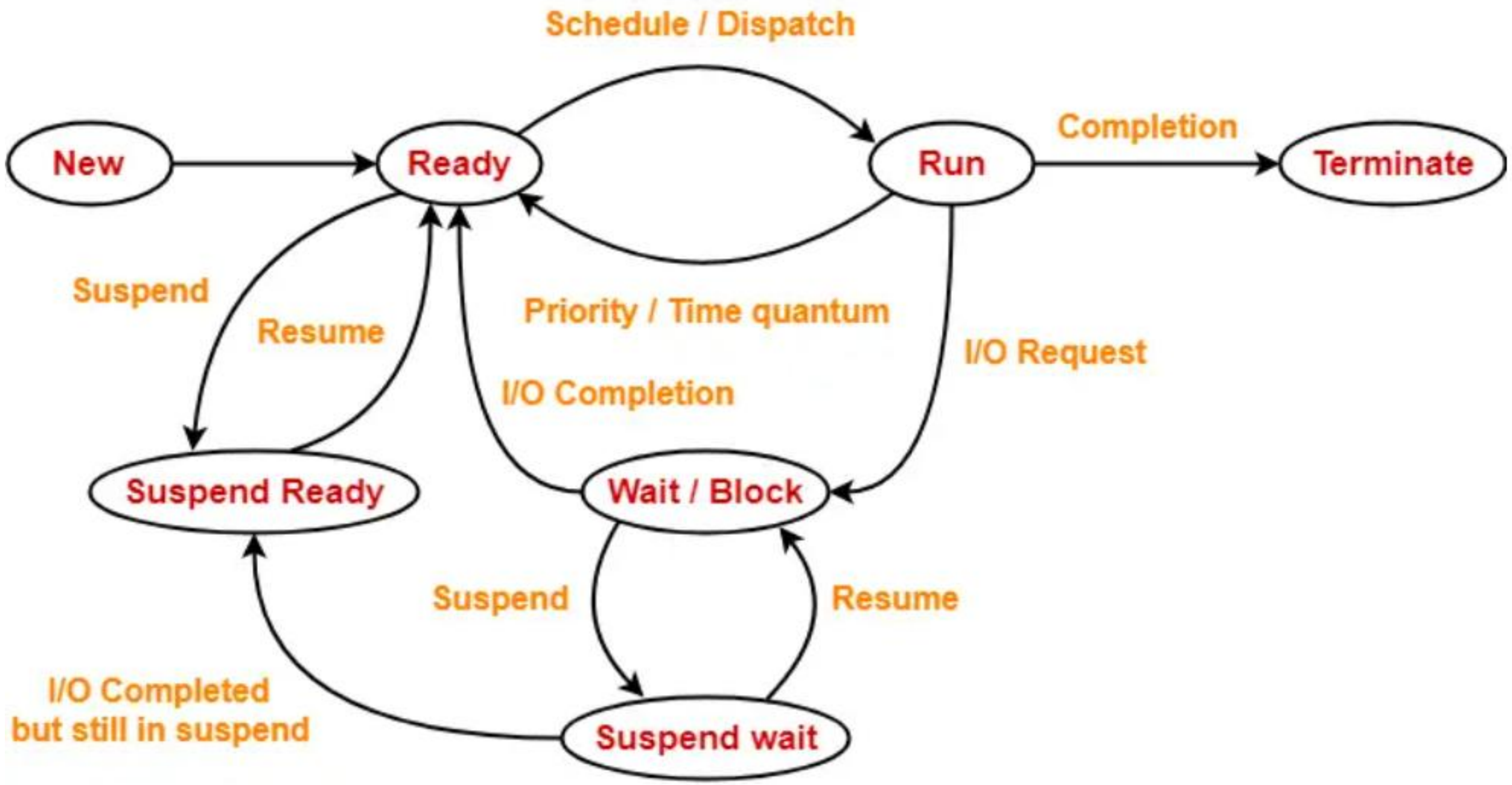


Five State Model (continued)



- 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

7-state process model (continued)

- **New**: The process is being created.
- **Ready**: The process is loaded into main memory and is waiting for CPU time.
- **Running**: The process is currently being executed by the CPU.
- **Wait/Blocked**: The process is waiting for an event to occur (e.g., I/O completion).
- **Wait/Blocked Suspend**: The process is in secondary memory and is waiting for an event to occur.
- **Ready Suspend**: The process is in secondary memory but is ready to execute as soon as it is loaded into main memory.
- **Terminate**: The process has completed its execution and is being removed from the system.

7-state process model (continued)

State Transitions

- **New to Ready**: The process is created and loaded into main memory.
- **Ready to Running**: The process is selected by the scheduler and given CPU time.
- **Running to Blocked**: The process is waiting for an event (e.g., I/O operation).
- **Blocked to Ready**: The event the process was waiting for has occurred, and it is ready to execute.
- **Running to Ready**: The process is preempted by the scheduler to allow another process to run.

7-state process model (continued)

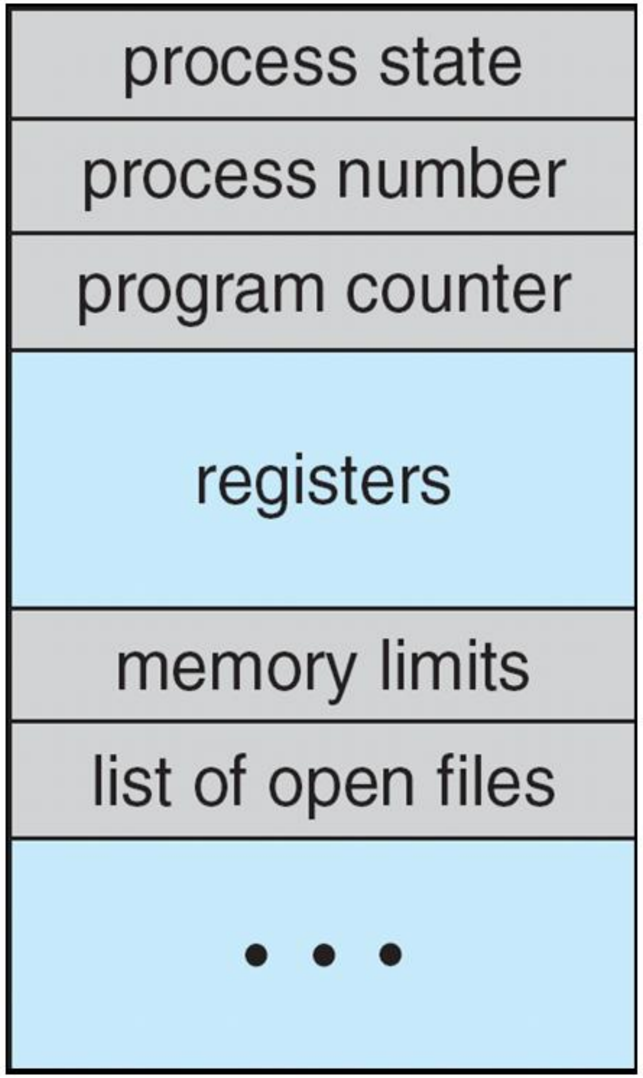
- **Blocked to Blocked Suspend**: The process is moved to secondary memory while waiting for an event.
- **Ready to Ready Suspend**: The process is moved to secondary memory to free up main memory.
- **Wait/Blocked Suspend to Ready Suspend**: The event the process was waiting for has occurred, but it remains in secondary memory.
- **Ready Suspend to Ready**: The process is moved back to main memory and is ready to execute.
- **Running to Terminate**: The process has completed its execution.

Process Control Block (PCB)

- **Information associated with each process (also called task control block)**
- Repository of any information related to Process.

Process Control Block (PCB) (continued)

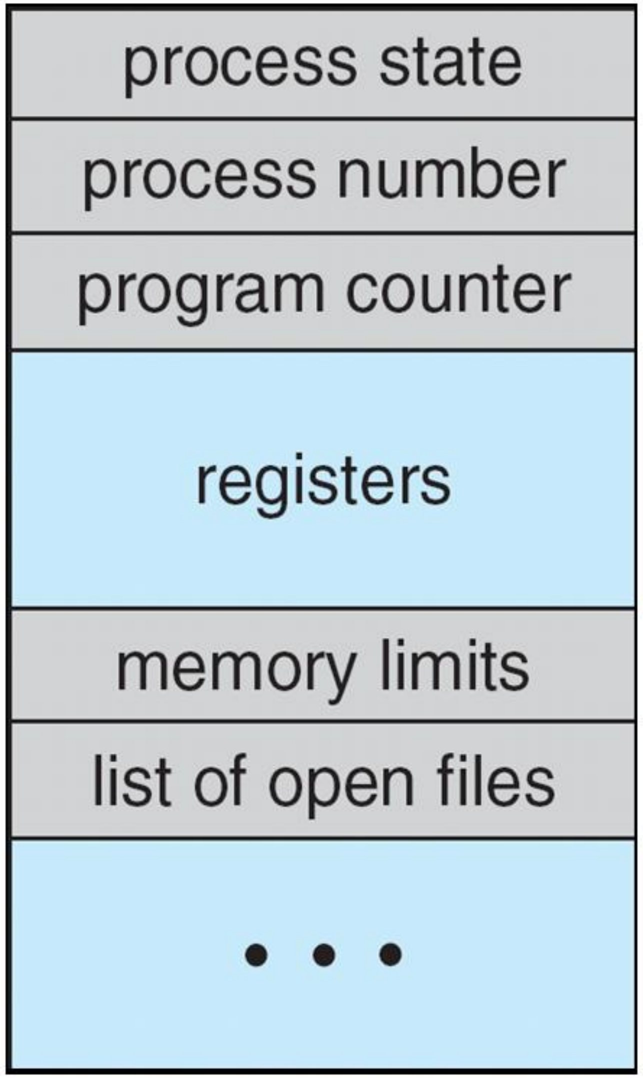
- **Process Identifier/Number -** An identifier that helps us in identifying and locating a process.
- **Process state –**It identifies the state that the process is currently in. It could be running, waiting, etc
- **Program counter** – address of the next instruction to be executed for this process.



Process Control Block (PCB) (continued)

CPU registers –

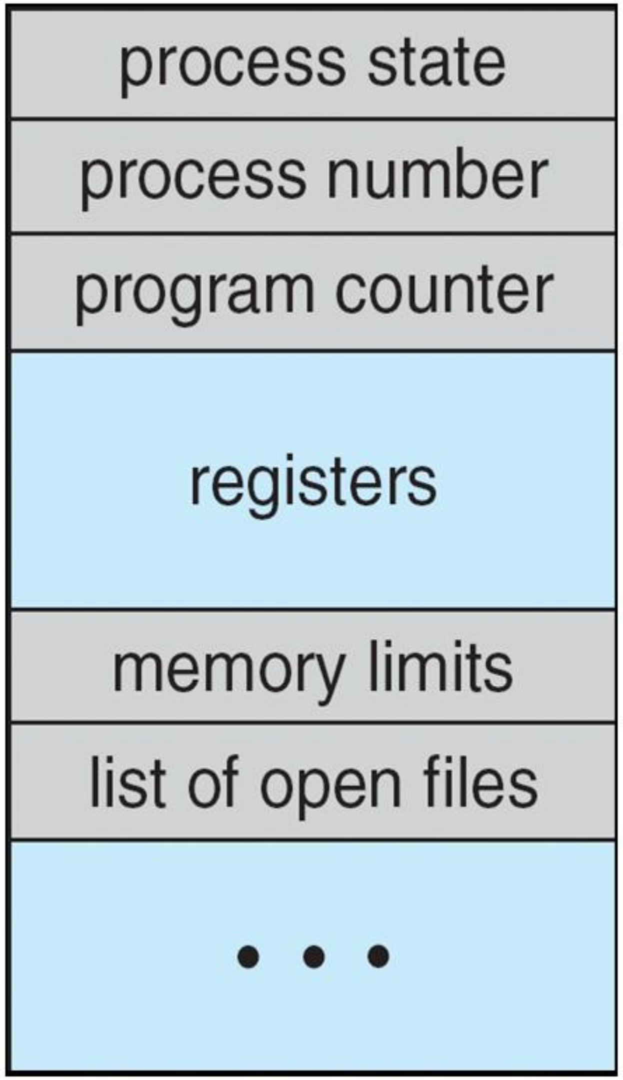
- contents of all process-centric registers,
- **Registers may vary in number and type depending on system architecture.**



Process Control Block (PCB) (continued)

CPU scheduling information-

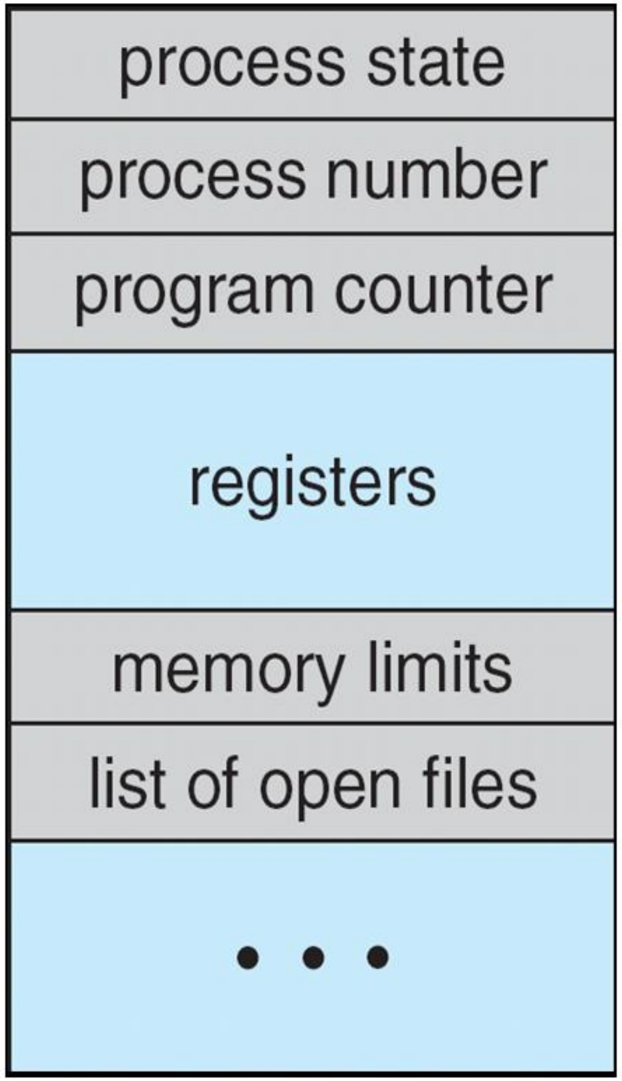
- Process priorities,
- scheduling queue pointers,
- any other scheduling parameters.



Process Control Block (PCB) (continued)

Memory-management information –

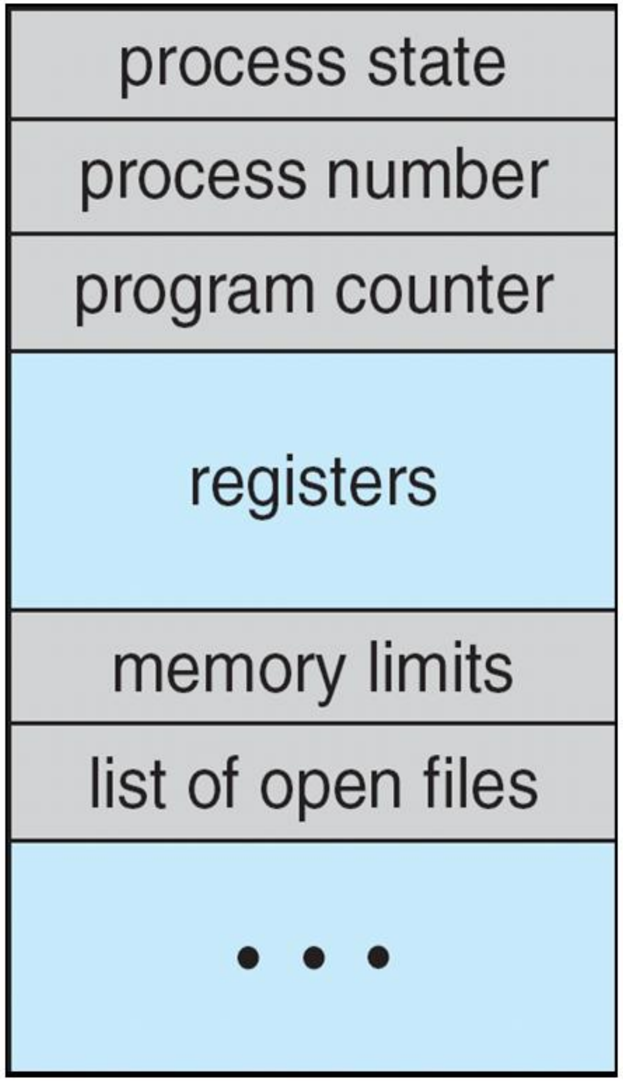
- memory allocated to the process,
- value of the Base and limit registers,
- the page tables or segment tables.



Process Control Block (PCB) (continued)

Accounting information –

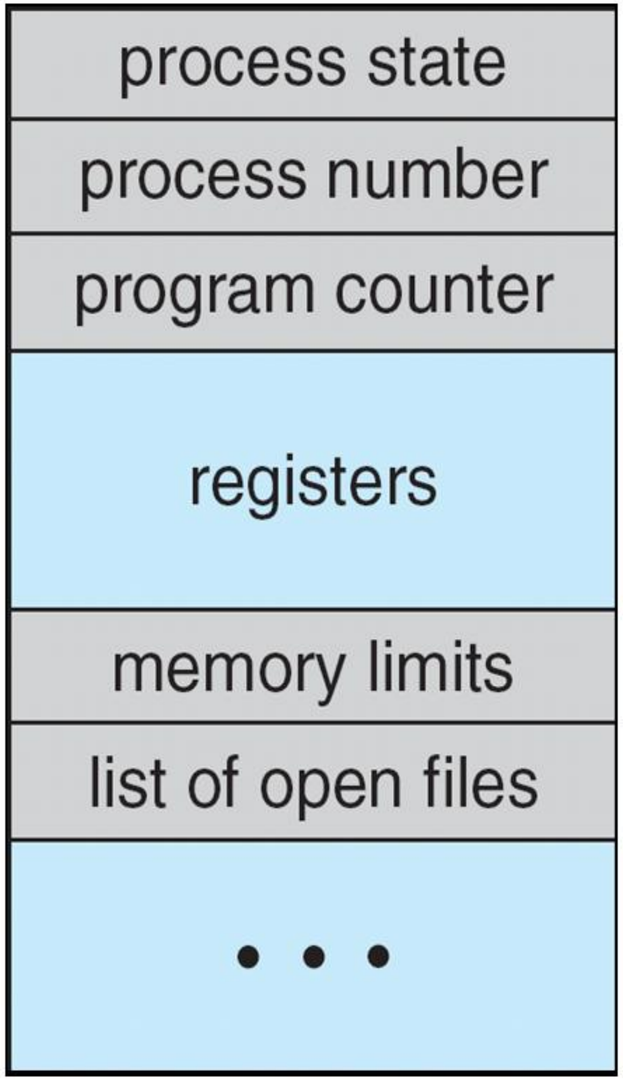
- CPU used,
- clock time elapsed since start, time limits,
- job or process numbers



Process Control Block (PCB) (continued)

I/O status information –

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



Operation on Process

- Process Creation
- Process Termination

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

Operations on Process

Process creation

- Parent – creating process
- Child – new process

pid – process identifier

How?

- fork() – unix
- CreateProcess () – Windows

How a child process gets it resources?

- OS
- Subset of parent resources

Why to create a child process?

Parallel Execution:

- For example, if you have a program that needs to perform multiple calculations and printing the document simultaneously, you can create child processes to handle calculation in parallel, with printing reducing the overall execution time.

Isolation:

- For instance, if you have a program that processes user input, you can create a child process to handle each input separately. If one child process encounters an error, it won't affect the parent process or other child processes.

How the OS creates a process?

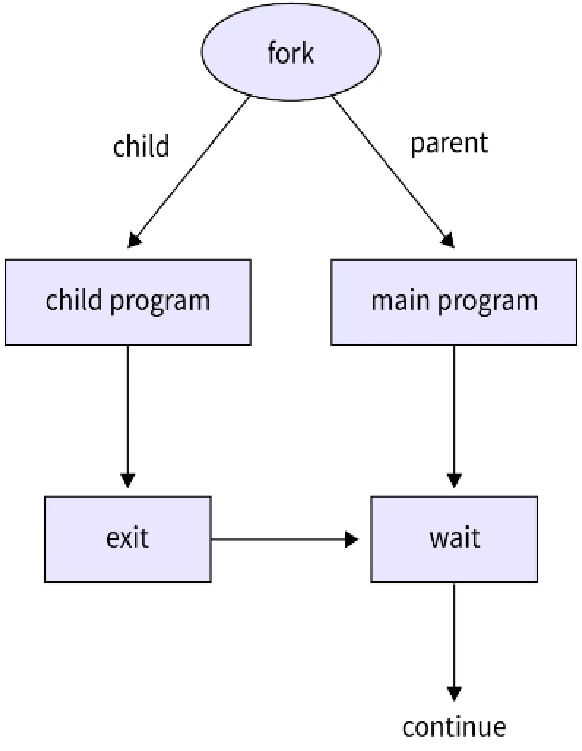
1. Create a process
2. Assign a unique process ID to newly created process
3. Allocate the memory and create its process image
4. Initialize process control block
5. Set the appropriate linkages to the different data structures such as ready queue etc.
6. Create or expand the other data structures if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

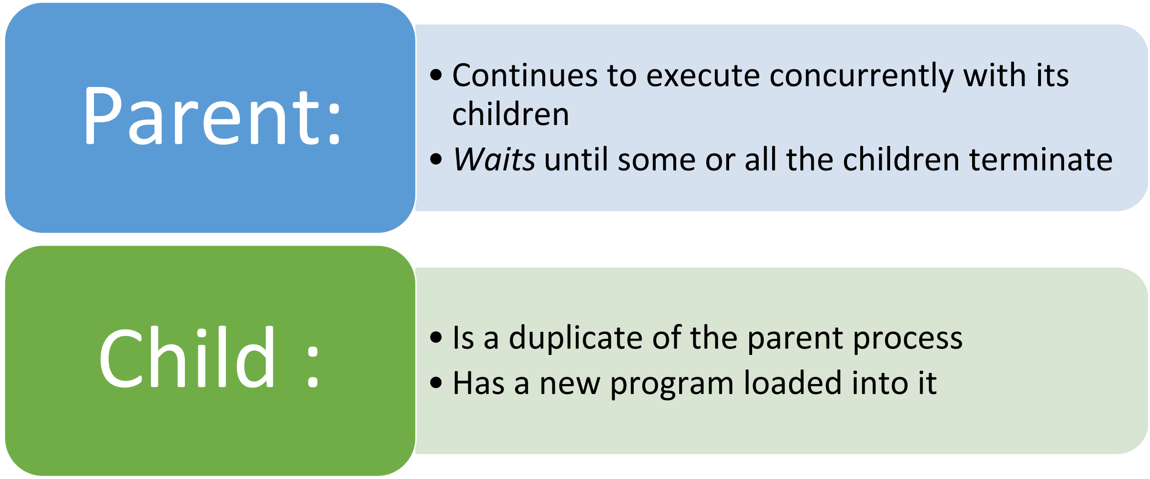
Causes of process blocking

- 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.

How to create process?

- A process uses a System call to create a child process.
- The system call will differ from OS to OS.
- For Unix/Linux the system call is **fork().**





Example – fork()

```python
#Import os Library
import os

os.fork();
print("the process is created using fork() system call\n");
```

```
the process is created using fork() system call

the process is created using fork() system call
```

Example – fork(), getpid()

Use os.getpid() to get the process id within the child, and os.getppid() to get the process id of the parent process (within the child).

```
Fork function  0
Child process => Parent ID  1234791  Child ID  1234794
Fork function  1234794
Parent process =>  1234791

Waiting for child process to finish.....

the child process is finished
```

```python
import os
import sys
# Fork a child process
processid = os.fork()
print("Fork function ",proce


if processid == 0 :
  print("Child process => Parent ID ",os.getppid() , " Child ID " , os.getpid())
  sys.exit(0)

else :
  print("Parent process => ", os.getpid())
  print("\nWaiting for child process to finish.....\n")
  os.wait()
  print("the child process is finished")
```

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 (continued)

- **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**

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

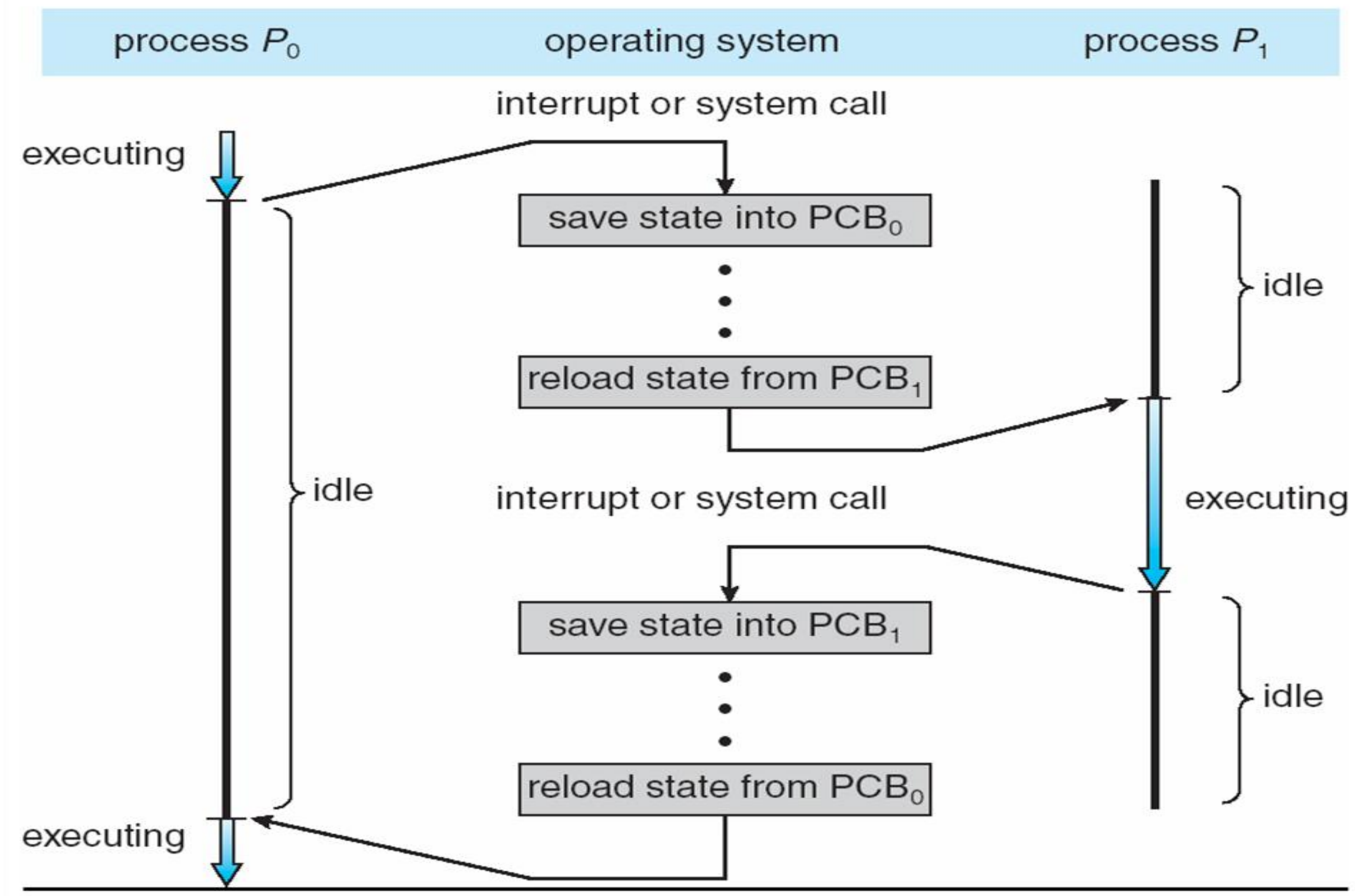
Process context switch Vs mode switch

- Context Switch:
  - A context switch is the process of saving the state of a currently running process and restoring the state of another process.
  - This allows the operating system to switch between processes, giving the illusion that multiple processes are running simultaneously.
  - Steps involved
    - Save the state of the current process (e.g., CPU registers, program counter).
    - Load the state of the next process to be executed.
    - Update the process control block (PCB) of both processes.

Process context switch Vs mode switch

- 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.
  - Steps Involved:
    - The application makes a system call.
    - The CPU switches to kernel mode to execute the system call.
    - Once the system call is completed, the CPU switches back to user mode.

CPU Switch From Process to Process



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 ?