Operating System (OS)
GTU # 3140702

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Unit-2
Process and Threads Management

Prof. Firoz A Sherasiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
firoz.sherasiya@darshan.ac.in
9879879861

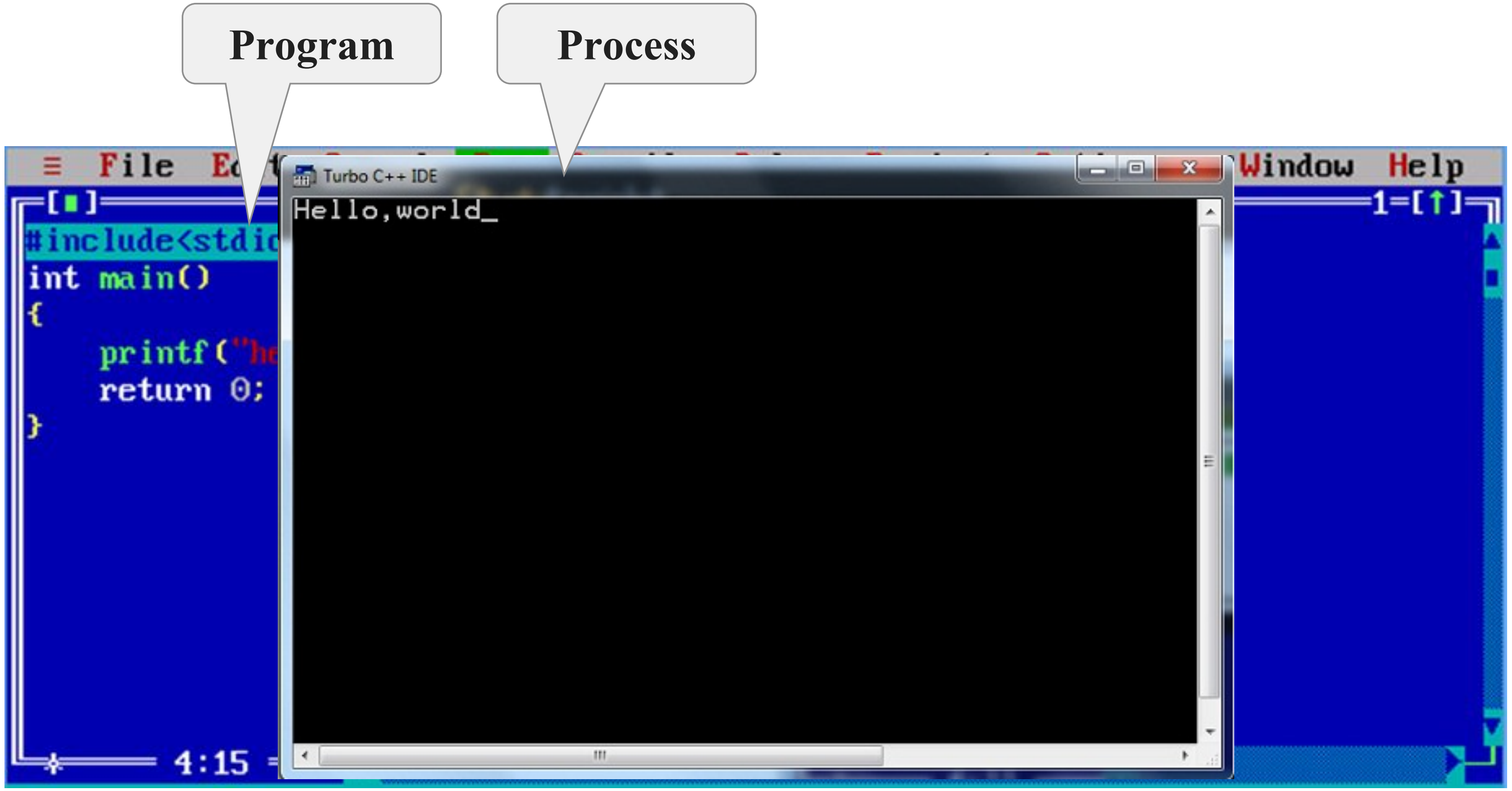
Outline

- Process concept
- Process states
- Process state transitions
- Process Control Block (PCB)
- Context switching
- Threads
- Comparison between process and thread
- Benefits/Advantages of threads
- Types of threads
- Multi Threading Models
- Pthread function calls
- System calls

Process concept

Section - 1

What is Process?



```
#include<stdio
int main()
{
    printf("he
    return 0;
}
```

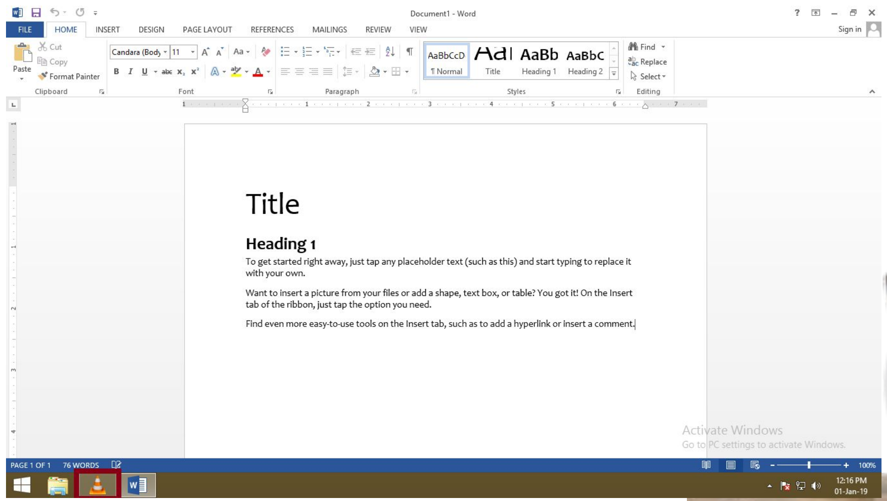
```
Hello,world_
```

What is Process?

- Process is a **program under execution**.
- Process is an **abstraction of a running program**.
- Process is an **instance of an executing program**, including the current values of the program counter, registers & variables.
- **Each process has its own virtual CPU**.

Multiprogramming

- The real **CPU switches** back and forth from process to process.
- This **rapid switching back and forth is called multiprogramming**.
- The **number of processes loaded simultaneously in memory is called degree of multiprogramming**.



Title

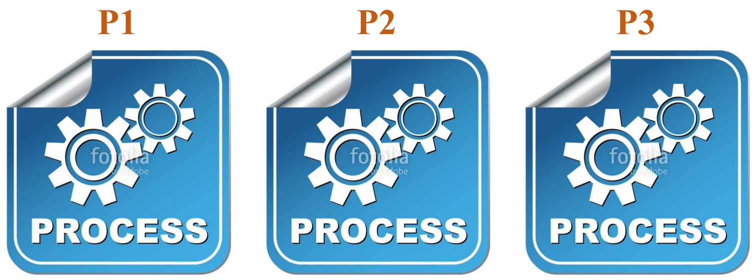
Heading 1

To get started right away, just tap any placeholder text (such as this) and start typing to replace it with your own.

Want to insert a picture from your files or add a shape, text box, or table? You got it! On the Insert tab of the ribbon, just tap the option you need.

Find even more easy-to-use tools on the Insert tab, such as to add a hyperlink or insert a comment.

Multiprogramming execution



Memory

Logical Program Counter
P1

Logical Program Counter
P2

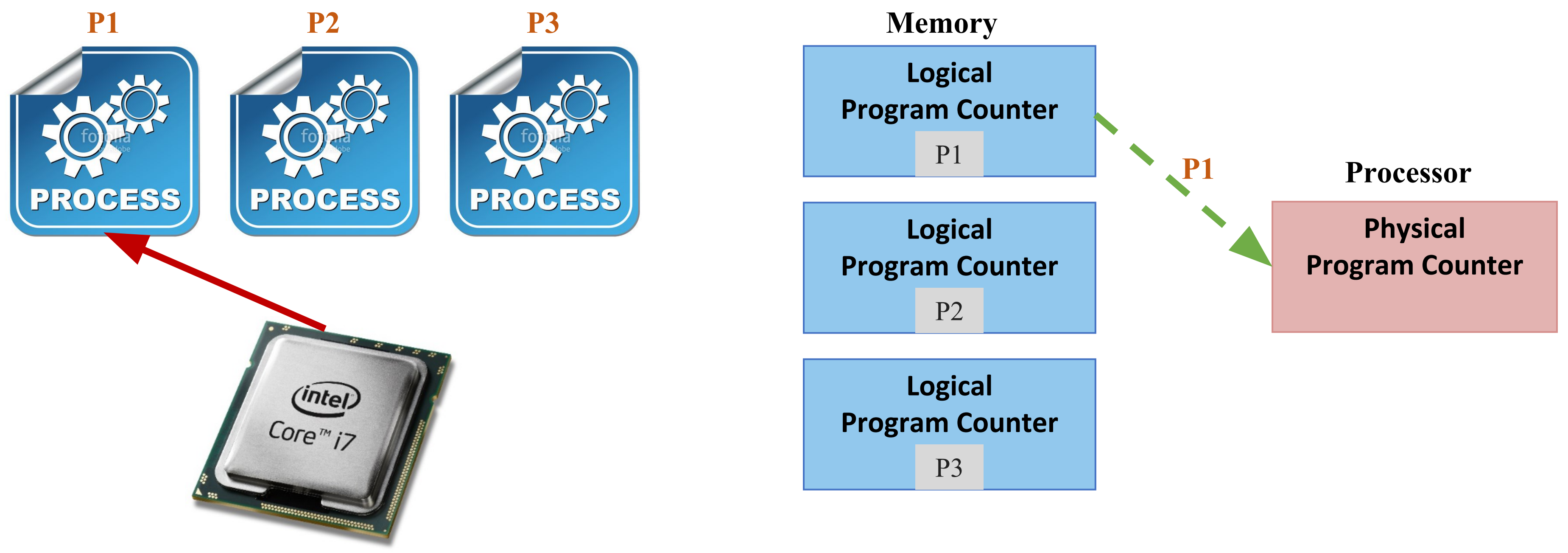
Logical Program Counter
P3

Processor

Physical Program Counter

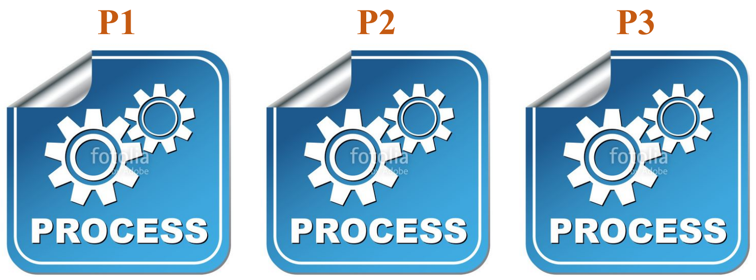
- There are **three processes**, **one processor** (CPU), **three logical program counter** (one for each processes) in memory and one physical program counter in processor.
- Here **CPU is free** (no process is running).
- No data in physical program counter.

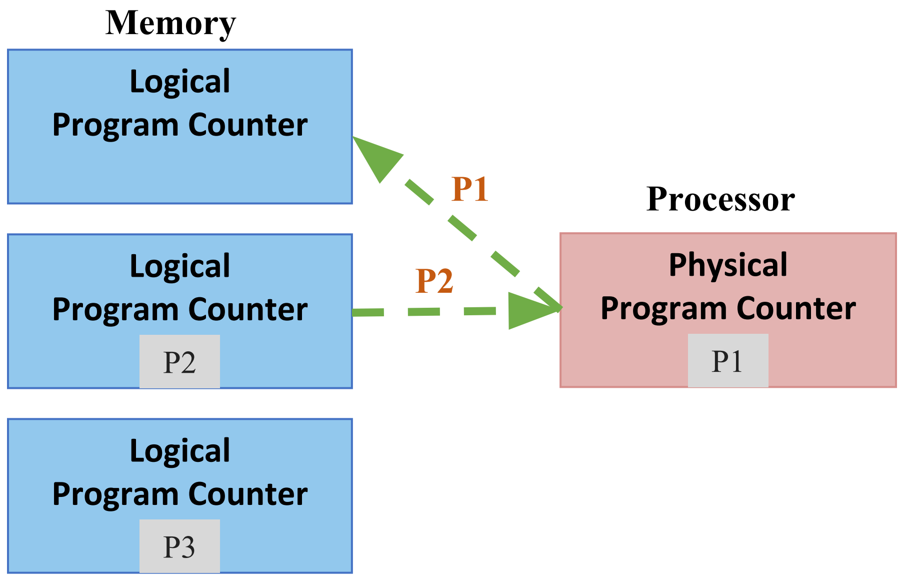
Multiprogramming execution



- CPU is **allocated to process P1** (process P1 is running).
- **Data of process P1 is copied** from its logical program counter to the physical program counter.

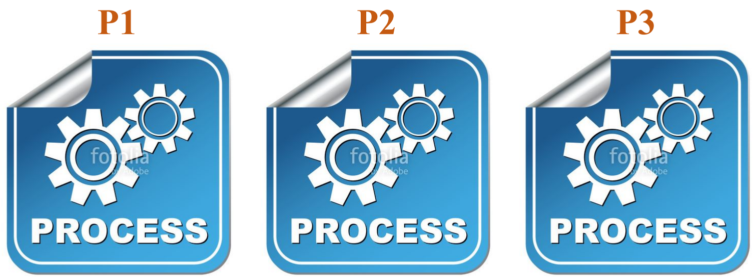
Multiprogramming execution

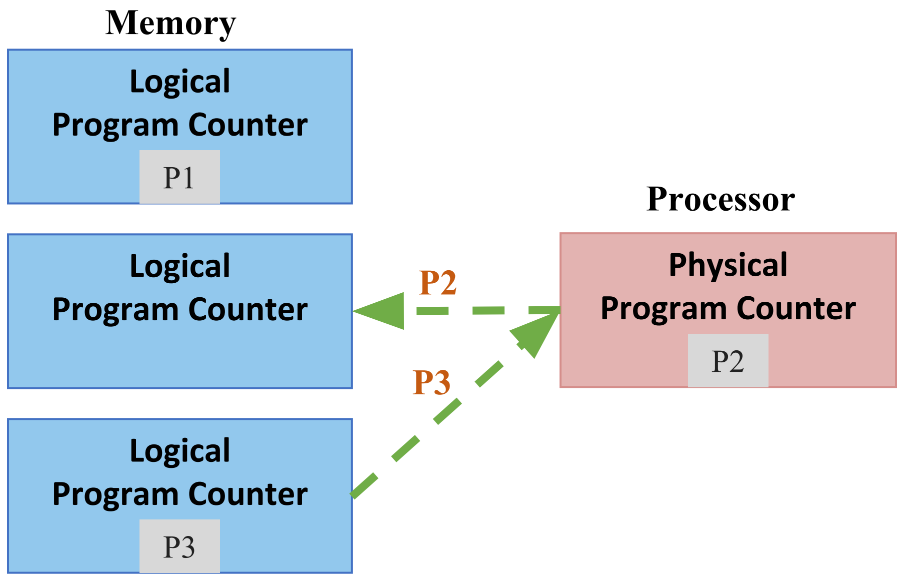




- CPU **switches** from process **P1 to** process **P2**.
- CPU is **allocated to process P2** (process P2 is running).
- **Data of process P1 is copied back** to its logical program counter.
- **Data of process P2 is copied** from its logical program counter to the physical program counter.

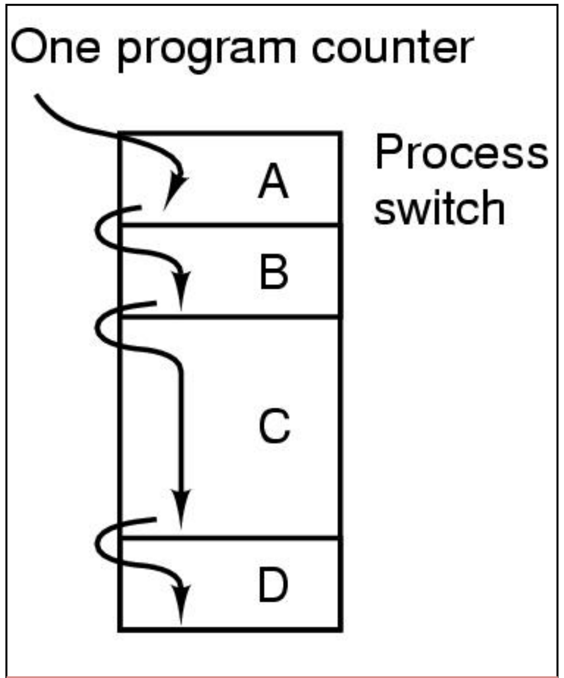
Multiprogramming execution



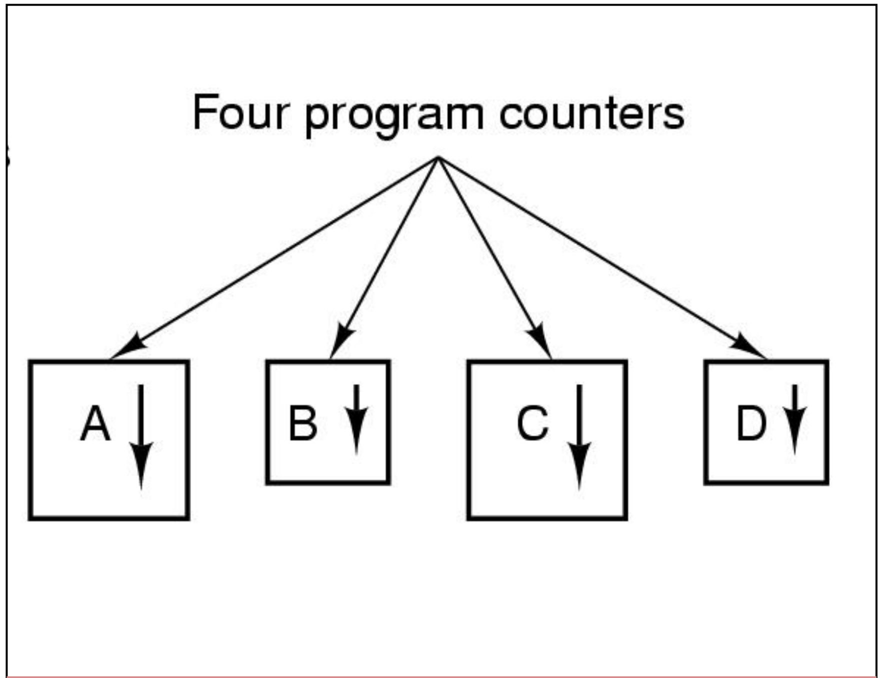


- CPU **switches** from process **P2 to** process **P3**.
- CPU is **allocated to process P3** (process P3 is running).
- **Data of process P2 is copied back** its logical program counter.
- **Data of process P3 is copied** from its logical program counter to the physical program counter.

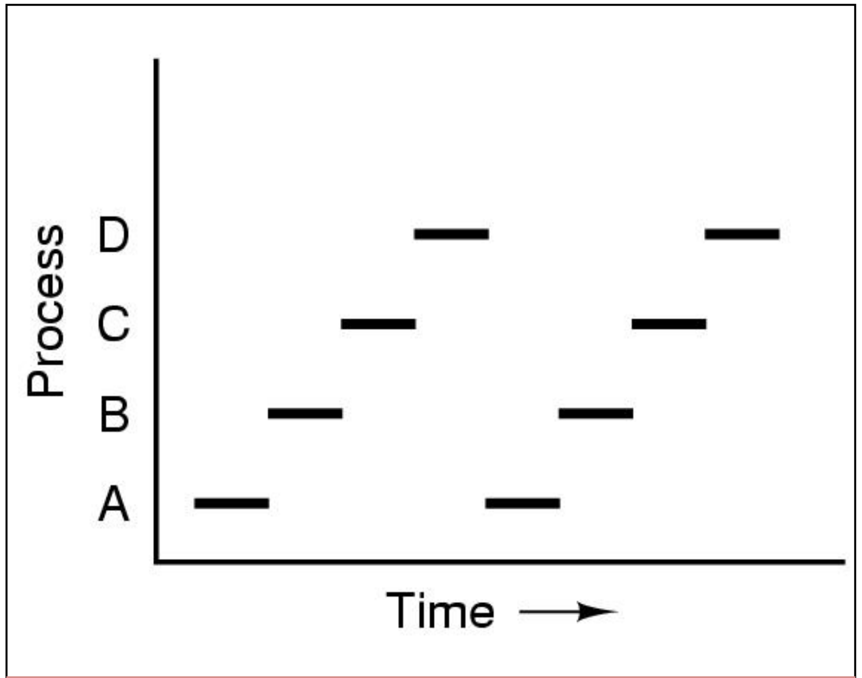
Process Model



Multiprogramming of four programs in memory



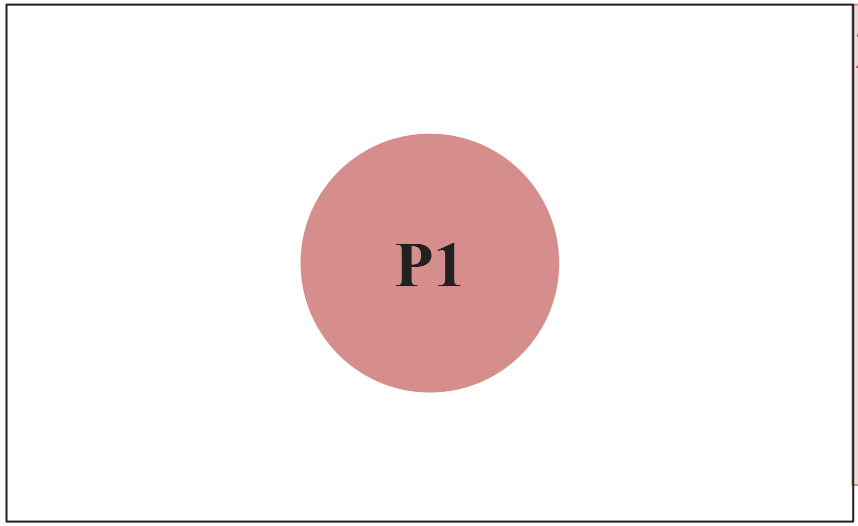
Conceptual model of 4 independent, sequential processes, each with its own flow of control (i.e., its own logical program counter) and each one running independently of the other ones.



Over a long period of time interval, all the processes have made progress, but at any given instant only one process is actually running.

Process Creation

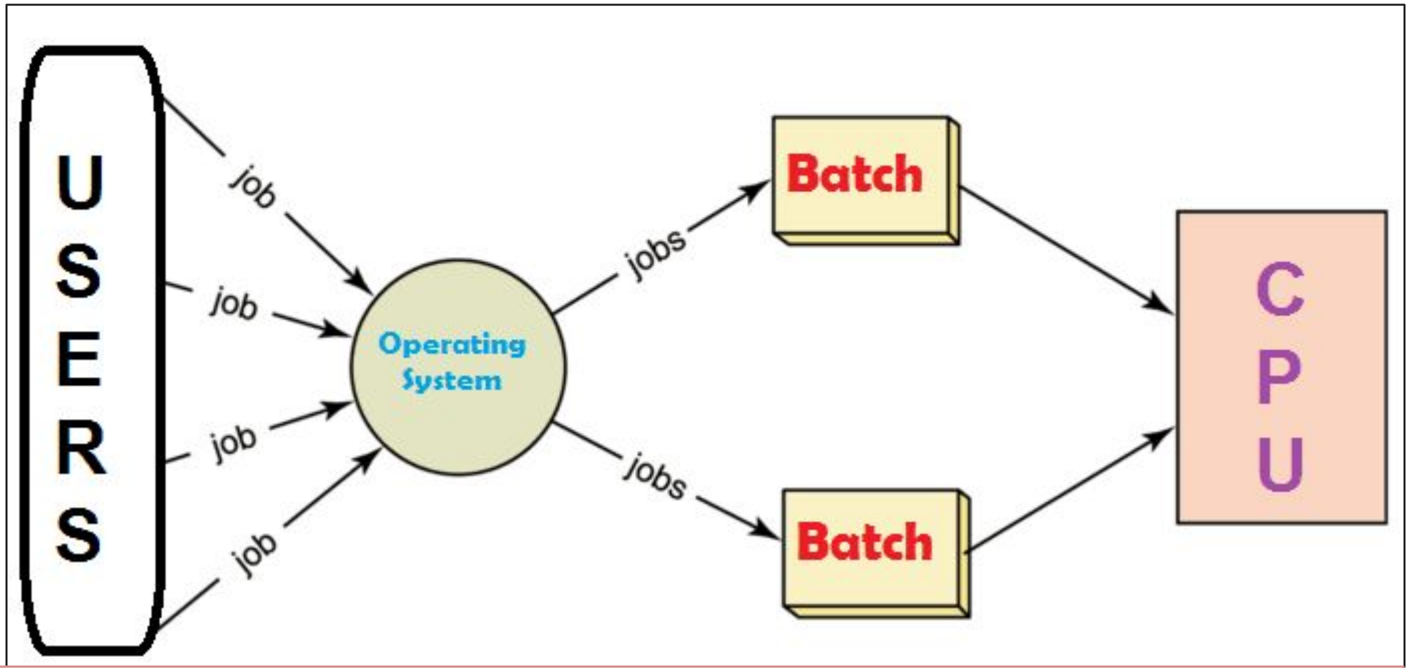
1. **System initialization**
   - At the time of **system (OS) booting** various processes are created
   - Foreground and background processes are created
   - **Background process** – that **do not interact with user** e.g. process to accept mail
   - **Foreground Process** – that **interact with user**



2. **Execution of a process creation system call (fork) by running process**
   - Running process will issue system call (fork) to **create one or more new process to help it**.
   - A process fetching large amount of data and execute it will create two different processes one for fetching data and another to execute it.

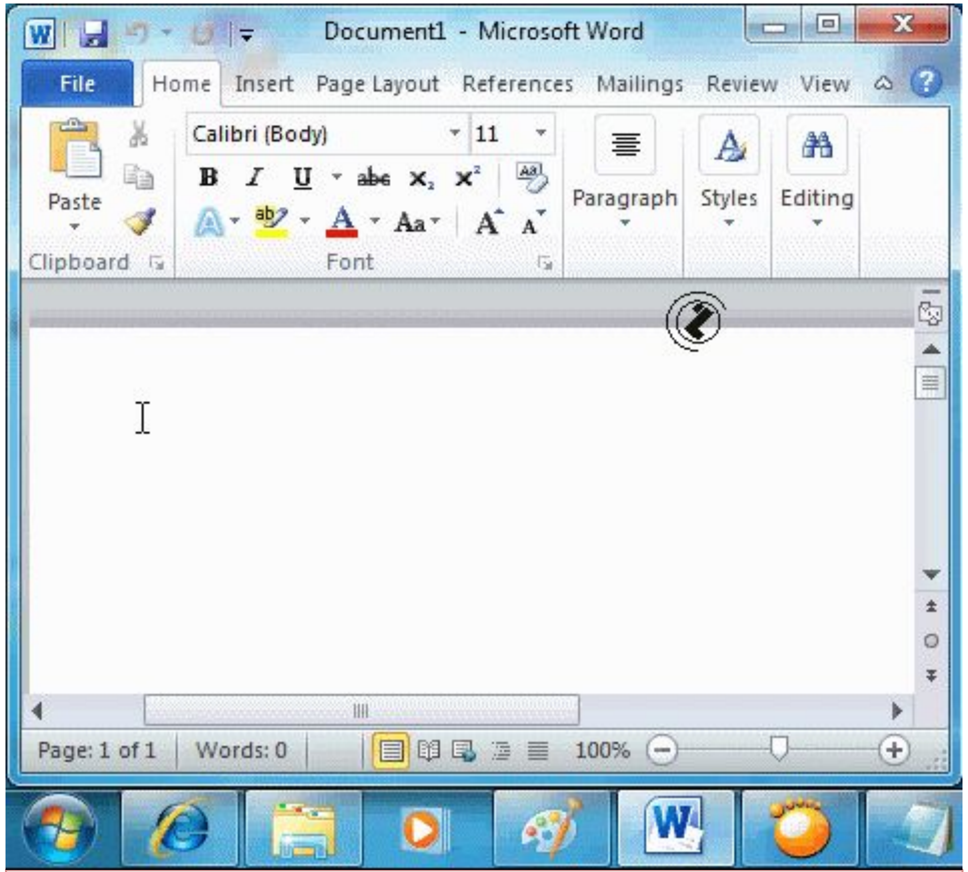
Process Creation

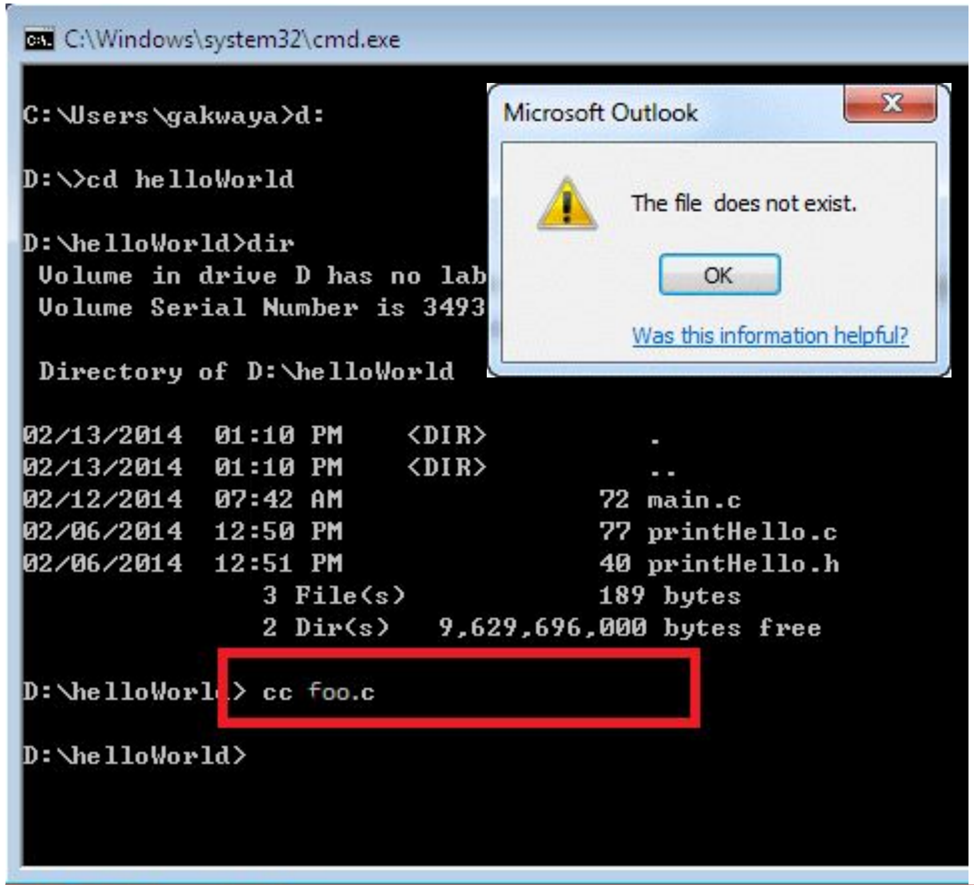
3. A **user request to create a new process**
   - Start process by clicking an icon (opening word file by double click) or by typing command.



4. **Initialization of batch process**
   - Applicable to only batch system found on large mainframe

Process Termination





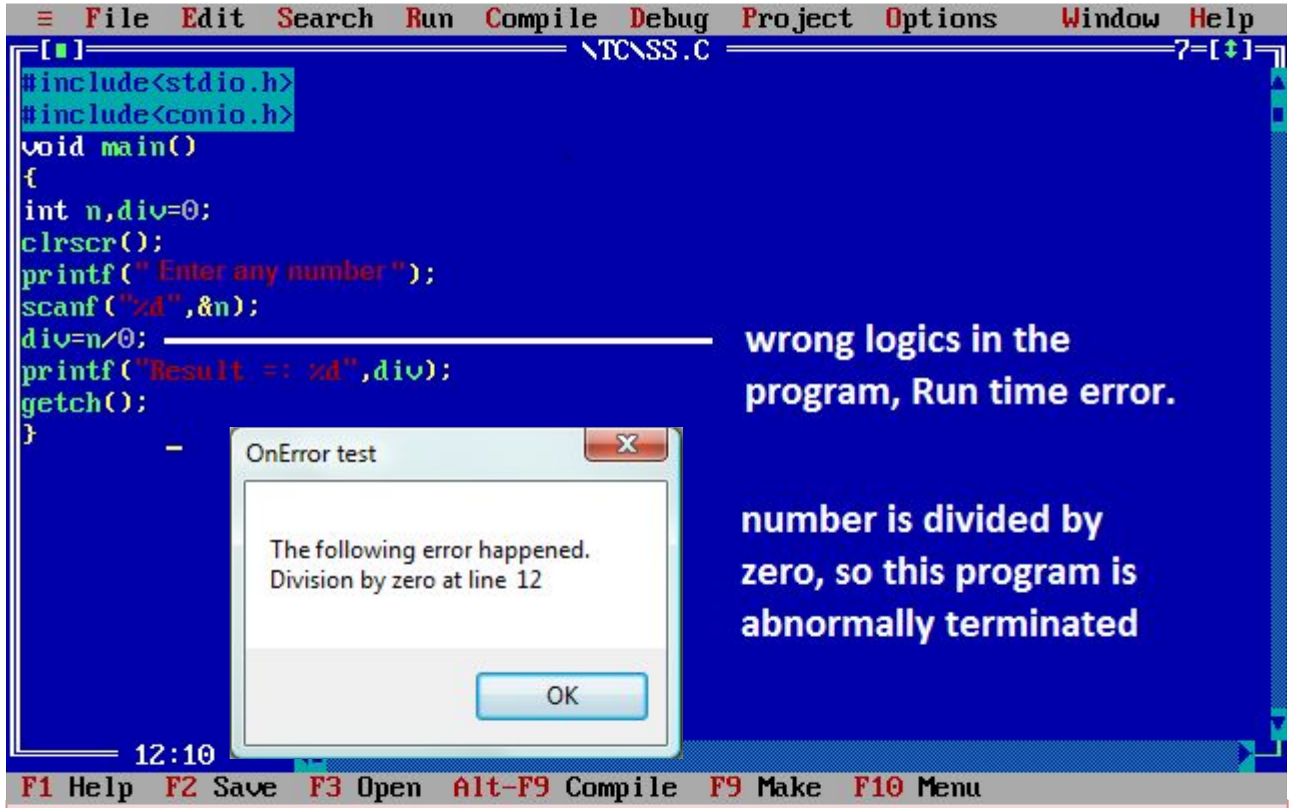
The file does not exist.

```
C:\Users\gakwaya>d:

D:\>cd helloWorld

D:\helloWorld>dir
 Volume in drive D has no lab
 Volume Serial Number is 3493

 Directory of D:\helloWorld

02/13/2014  01:10 PM    <DIR>          .
02/13/2014  01:10 PM    <DIR>          ..
02/12/2014  07:42 AM                72 main.c
02/06/2014  12:50 PM                77 printHello.c
02/06/2014  12:51 PM                40 printHello.h
               3 File(s)            189 bytes
               2 Dir(s)   9,629,696,000 bytes free

D:\helloWorld> cc foo.c

D:\helloWorld>
```

1. **Normal exit (voluntary)**
   - Terminated because process has done its work.

2. **Error exit (voluntary)**
   - The process discovers a fatal error e.g. user types the command cc foo.c to compile the program foo.c and no such file exists, the compiler simply exit.

Process Termination

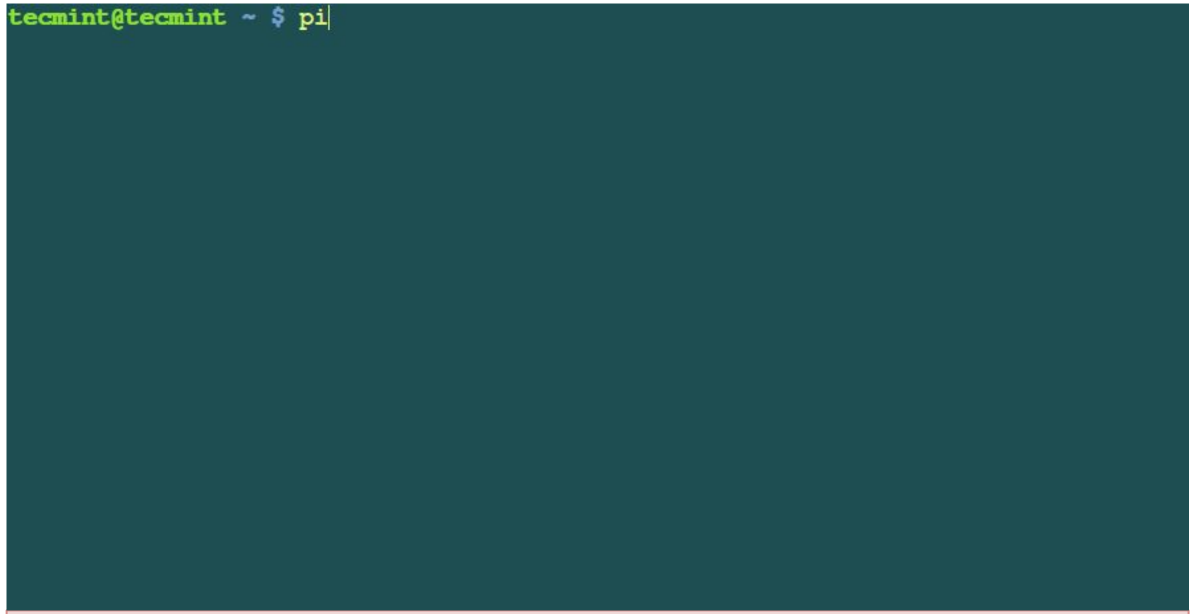

≡ File Edit Search Run Compile Debug Project Options Window Help
\TC\SS.C
12:10
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu
OnError test
OK

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,div=0;
clrscr();
printf(" Enter any number ");
scanf("%d",&n);
div=n/0;
printf("Result =: %d",div);
getch();
}
```

The following error happened.
Division by zero at line 12

wrong logics in the program, Run time error.

number is divided by zero, so this program is abnormally terminated



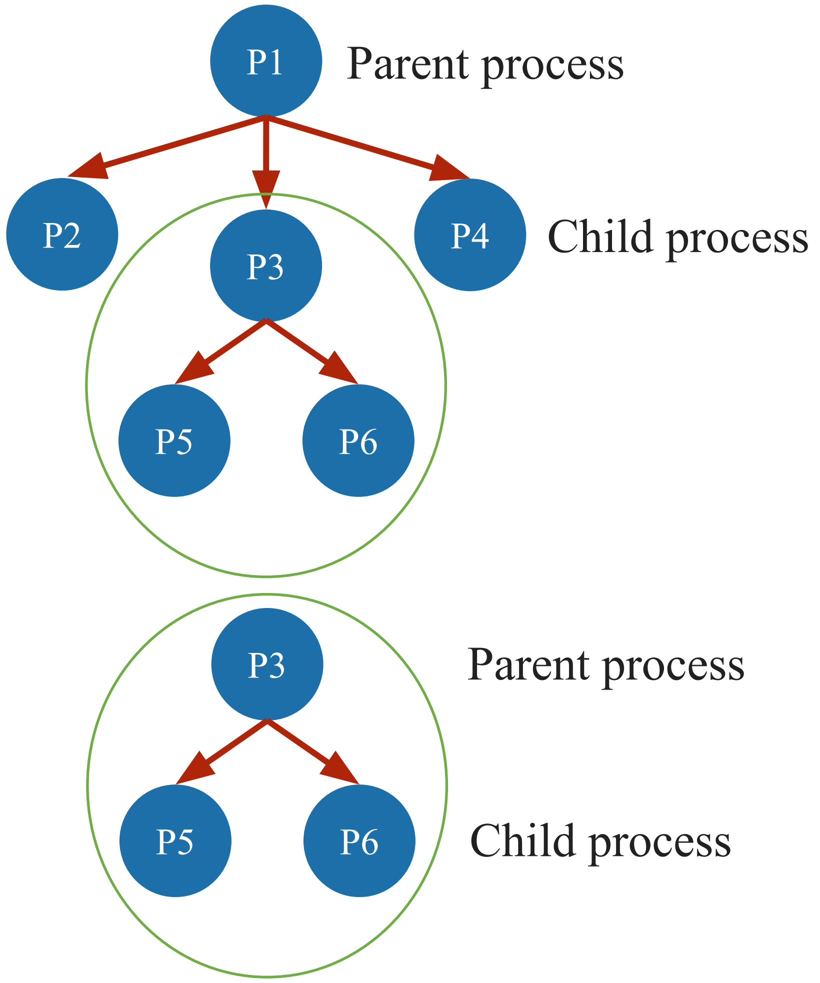
```
tecmint@tecmint ~ $ pi
```

3. **Fatal error (involuntary)**
    - An error caused by a process often due to a program bug e.g. executing an illegal instruction, referencing nonexistent memory or divided by zero.

4. **Killed by another process (involuntary)**
    - A process executes a system call telling the OS to kill some other process using kill system call.

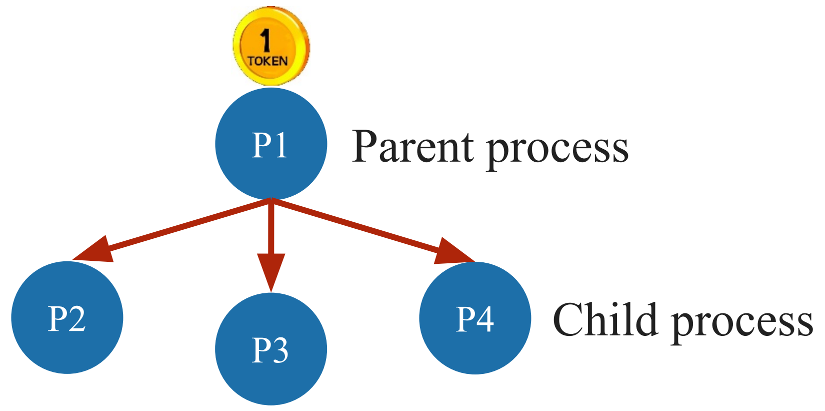
Process Hierarchies

- Parent process can create child process, child process can create its own child process.
- **UNIX has hierarchy concept** which is known as process group
- Windows has no concept of hierarchy
  - All the process as treated equal (**use handle** concept)



Handle

- When a process is created, the **parent process is given a special token called handle**.
- This **handle is used to control the child process**.
- A **process is free to pass this token** to some other process.



Process states

Section - 2

Process states

Runnin
g

Running:

Process is actually using the CPU

Ready

Ready:

Process is runnable, temporarily stopped to let another process to run

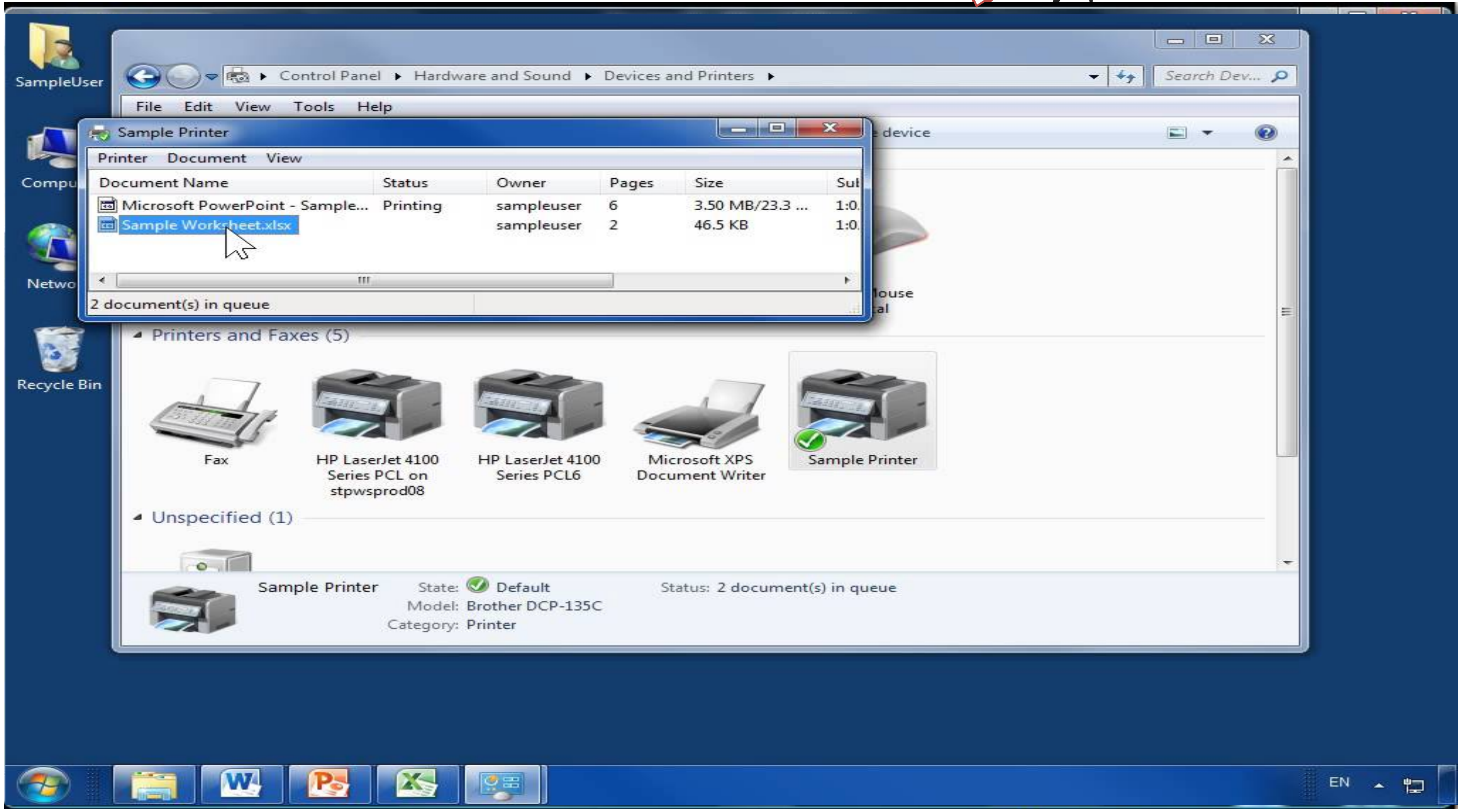
Blocked

Blocked:

Process is unable to run until some external event happens

Process states

Blocked


SampleUser
Control Panel ▸ Hardware and Sound ▸ Devices and Printers ▸
Search Dev...
File Edit View Tools Help
Sample Printer
Printer Document View
Printers and Faxes (5)
Fax
HP LaserJet 4100 Series PCL on stpwsprod08
HP LaserJet 4100 Series PCL6
Microsoft XPS Document Writer
Sample Printer
Unspecified (1)
Recycle Bin
EN

Document Name	Status	Owner	Pages	Size	Sul
Microsoft PowerPoint - Sample...	Printing	sampleuser	6	3.50 MB/23.3 ...	1:0.
Sample Worksheet.xlsx		sampleuser	2	46.5 KB	1:0.

2 document(s) in queue

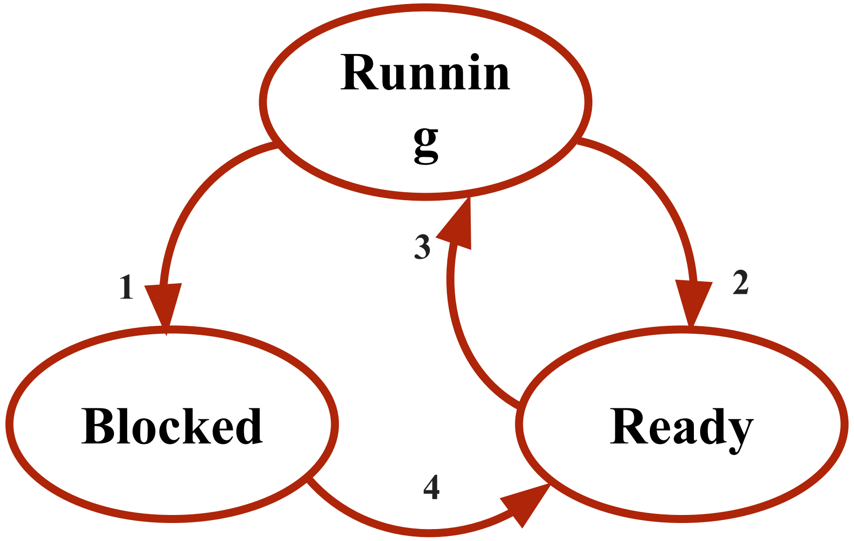
Sample Printer State: Default Status: 2 document(s) in queue
Model: Brother DCP-135C
Category: Printer

Process states transitions

Section - 3

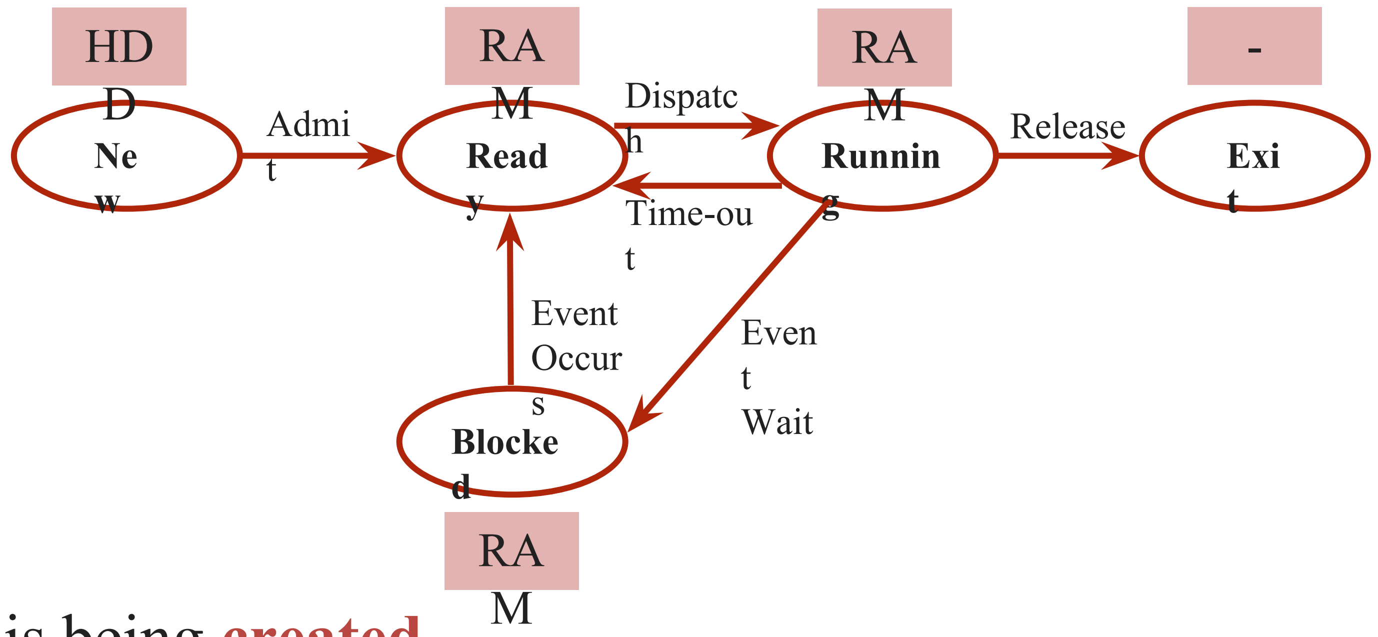
Process State Transitions

- When and how these transitions occur (process moves from one state to another)?
  1. Process blocks for input or waits for an event (i.e. printer is not available)
  2. Scheduler picks another process
     - End of time-slice or pre-emption.
  3. Scheduler picks this process
  4. Input becomes available, event arrives (i.e. printer become available)



Processes are always either executing (running) or waiting to execute (ready) or waiting for an event (blocked) to occur.

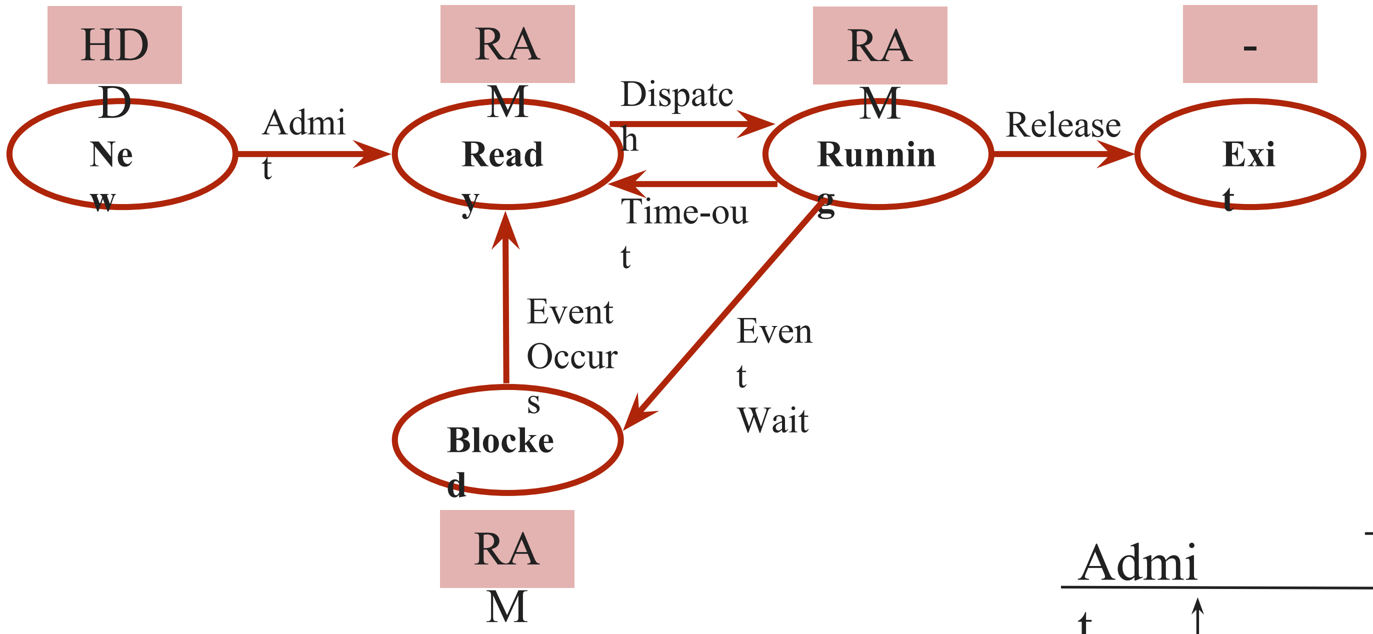
Five State Process Model and Transitions



- **New** – process is being **created**
- **Ready** – process is **waiting to run (runnable)**, temporarily stopped to let another process run
- **Running** – process is actually **using the CPU**
- **Blocked** – **unable to run** until some external event happens
- **Exit (Terminated)** – process has **finished the execution**

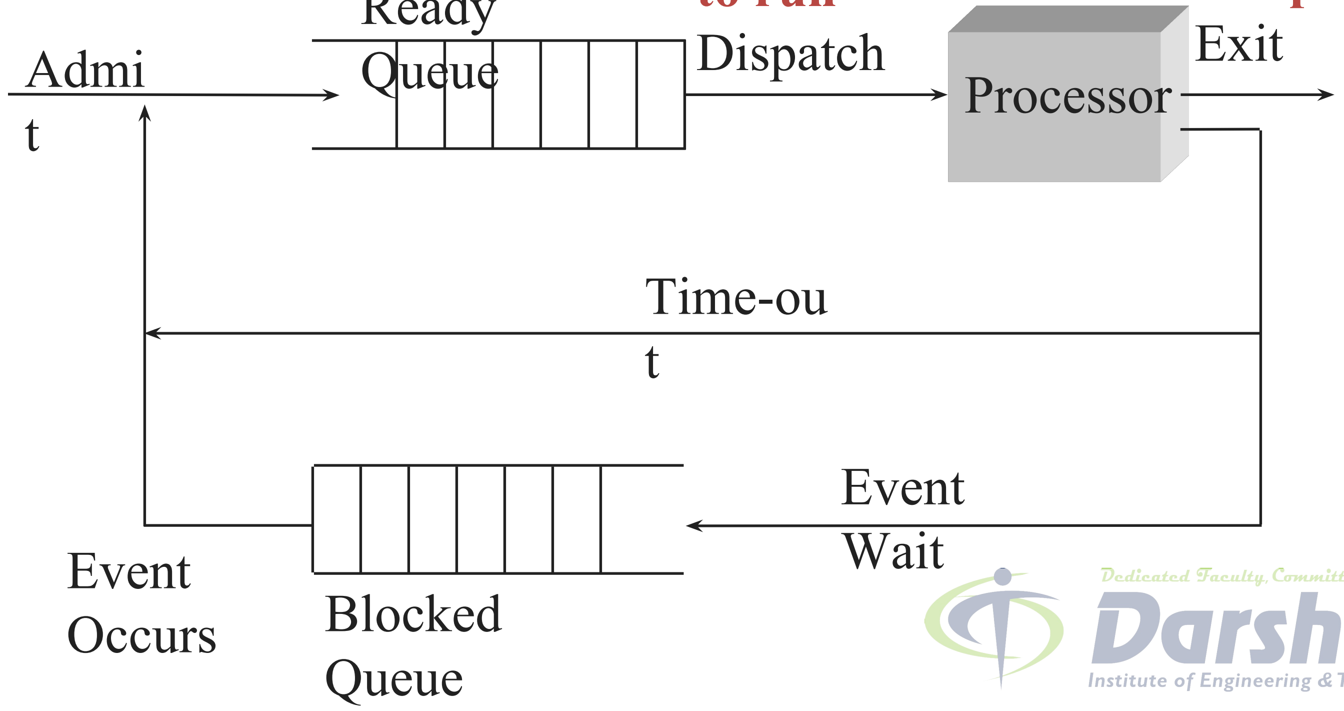
Exercise A process resides in which memory during different state?

Queue Diagram



Process is Scheduled to run

Process is completed



Process Control Block (PCB)

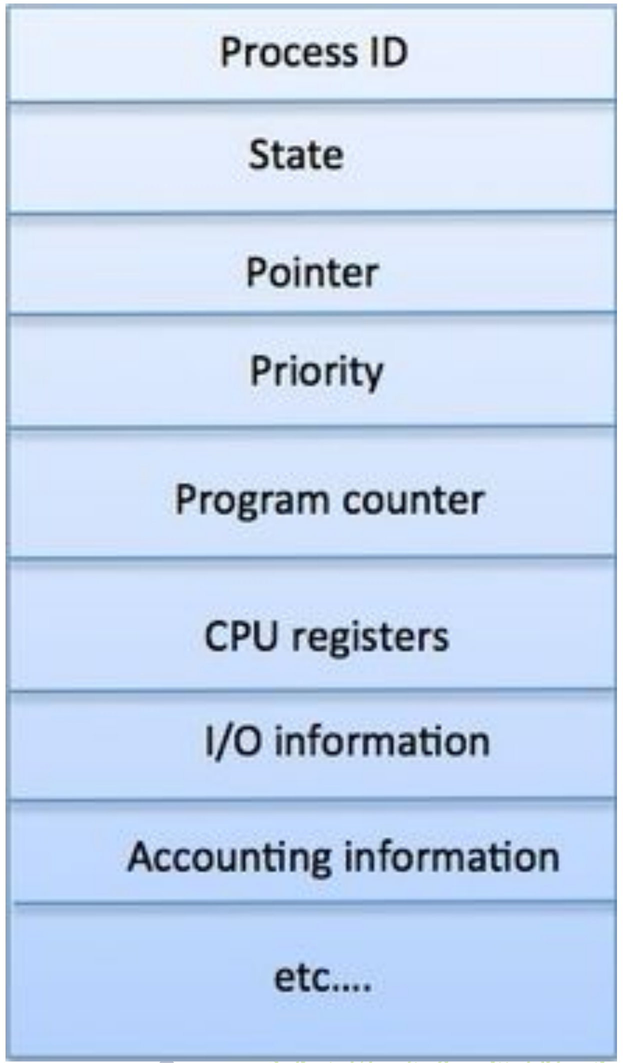
Section - 4

What is Process Control Block (PCB)?

- A Process Control Block (PCB) is a **data structure maintained by the operating system for every process**.
- PCB is used for **storing the collection of information about the processes**.
- The PCB is **identified by an integer process ID (PID)**.
- A PCB **keeps all the information needed to keep track of a process**.
- The PCB is maintained for a process **throughout its lifetime** and is **deleted once the process terminates**.
- The **architecture** of a PCB is completely **dependent on operating system** and may contain different information in different operating systems.
- PCB **lies in kernel memory space**.

Fields of Process Control Block (PCB)

- **Process ID** - Unique identification for each of the process in the operating system.
- **Process State** - The current state of the process i.e., whether it is ready, running, waiting.
- **Pointer** - A pointer to parent process.
- **Priority** - Priority of a process.
- **Program Counter** - Program Counter is a pointer to the address of the next instruction to be executed for this process.
- **CPU registers** - Various CPU registers where process need to be stored for execution for running state.
- **IO status information** - This includes a list of I/O devices allocated to the process.
- **Accounting information** - This includes the amount of CPU used for process execution, time limits etc.

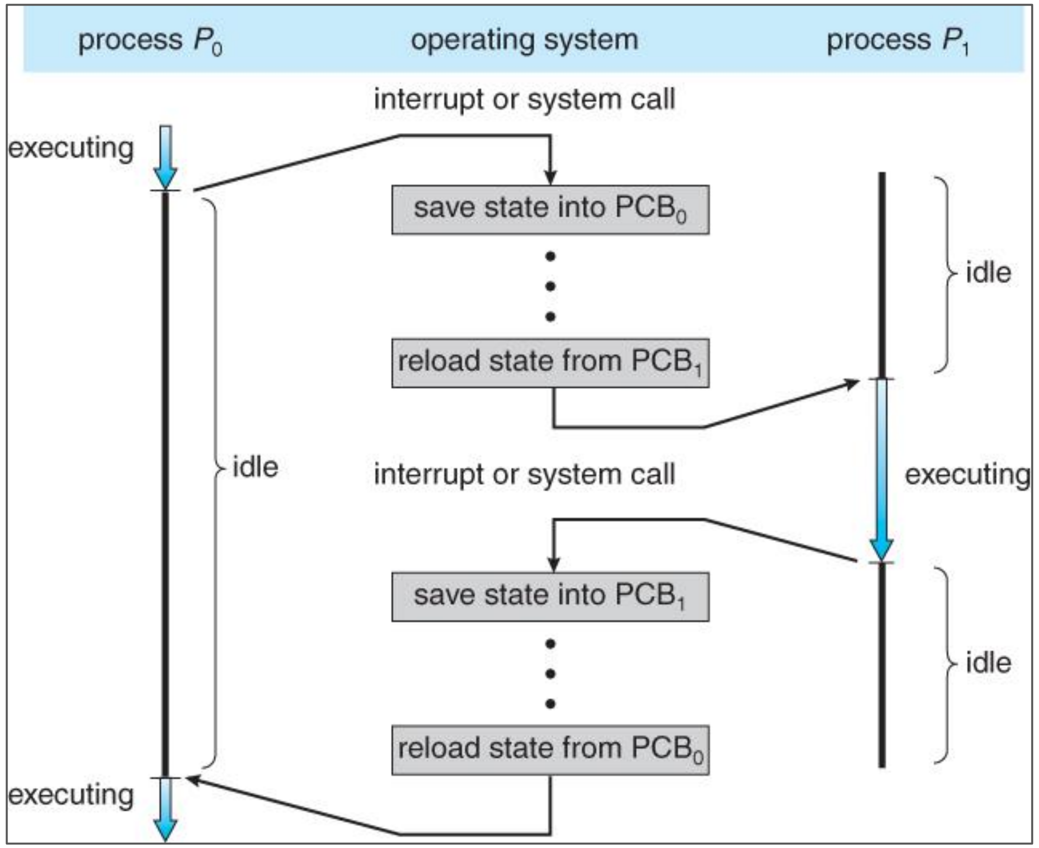


Context switching

Section - 5

Context switching

- Context switch means **stopping one process** and **restarting another process**.
- When an event occur, the **OS saves the state of an active process (into its PCB)** and **restore the state of new process (from its PCB)**.
- Context switching is **purely overhead** because system does not perform any useful work while context switch.
- Sequence of action:
  1. **OS takes control** (through interrupt)
  2. **Saves context of running process** in the process PCB
  3. **Reload context of new process** from the new process PCB
  4. **Return control** to new process



Exercise What causes (Reasons for) context switching?

1. Time slice has elapsed
2. Process with a higher priority has become ready to run.

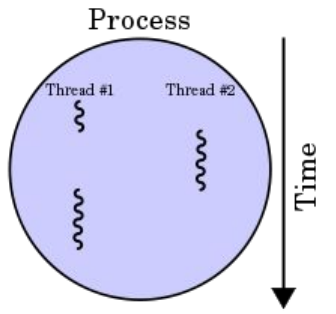
Threads

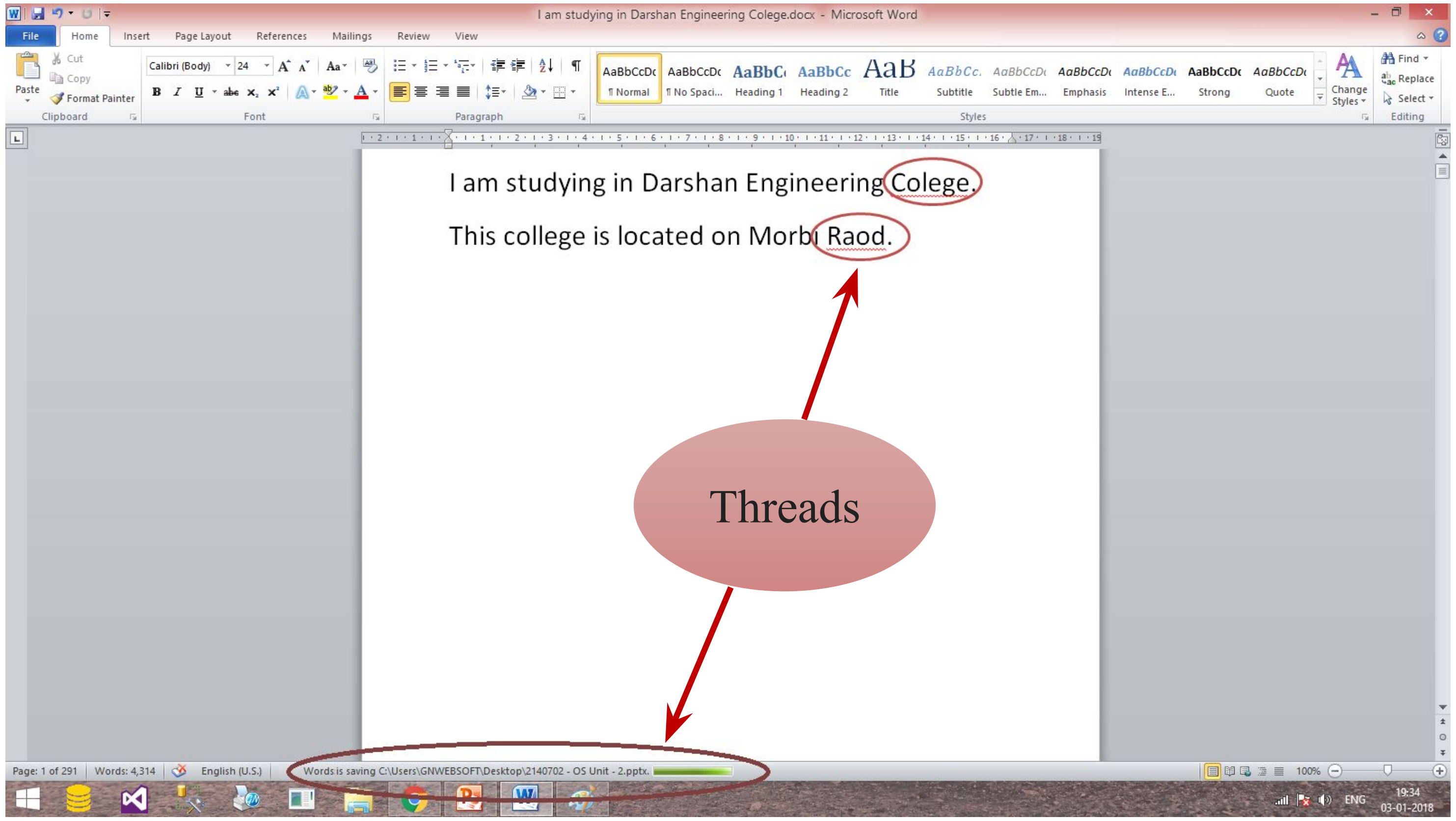
Section - 6

What is Threads?

- Thread is **light weight process** created by a process.

Processes are used to execute large, 'heavyweight' jobs such as working in word, while threads are used to carry out smaller or 'lightweight' jobs such as auto saving a word document.




I am studying in Darshan Engineering Colege.docx - Microsoft Word
File
Home
Insert
Page Layout
References
Mailings
Review
View
Threads

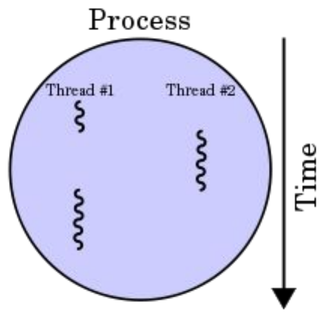
I am studying in Darshan Engineering Colege.

This college is located on Morbi Raod.

Page: 1 of 291 | Words: 4,314 | English (U.S.) | Words is saving C:\Users\GNWEBSOFT\Desktop\2140702 - OS Unit - 2.pptx.

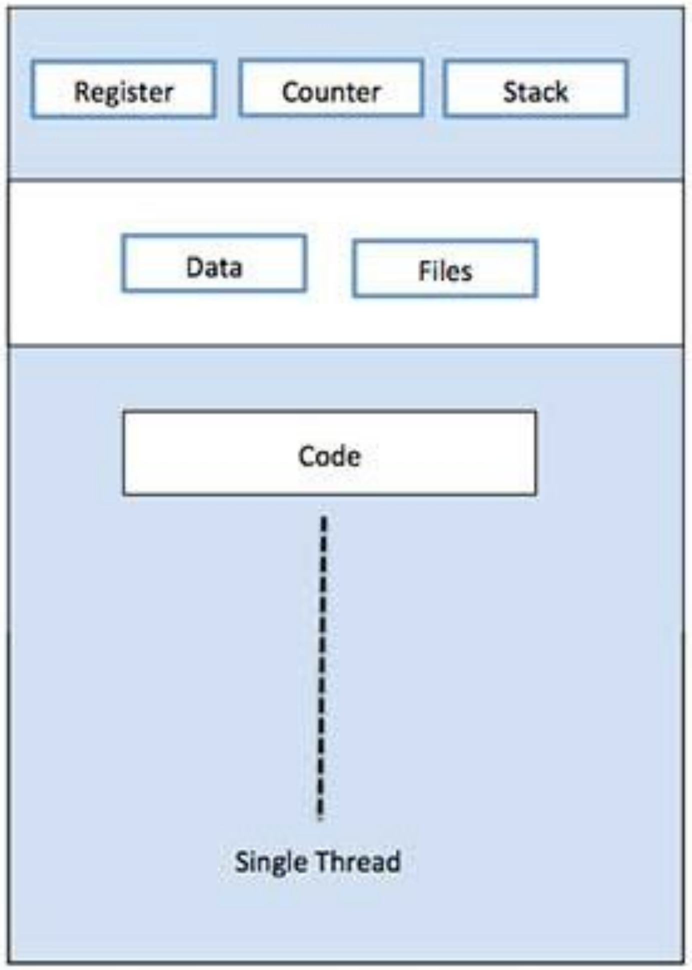
What is Threads?

- Thread is **light weight process** created by a process.
- Thread is a **single sequence stream** within a process.
- Thread has it own

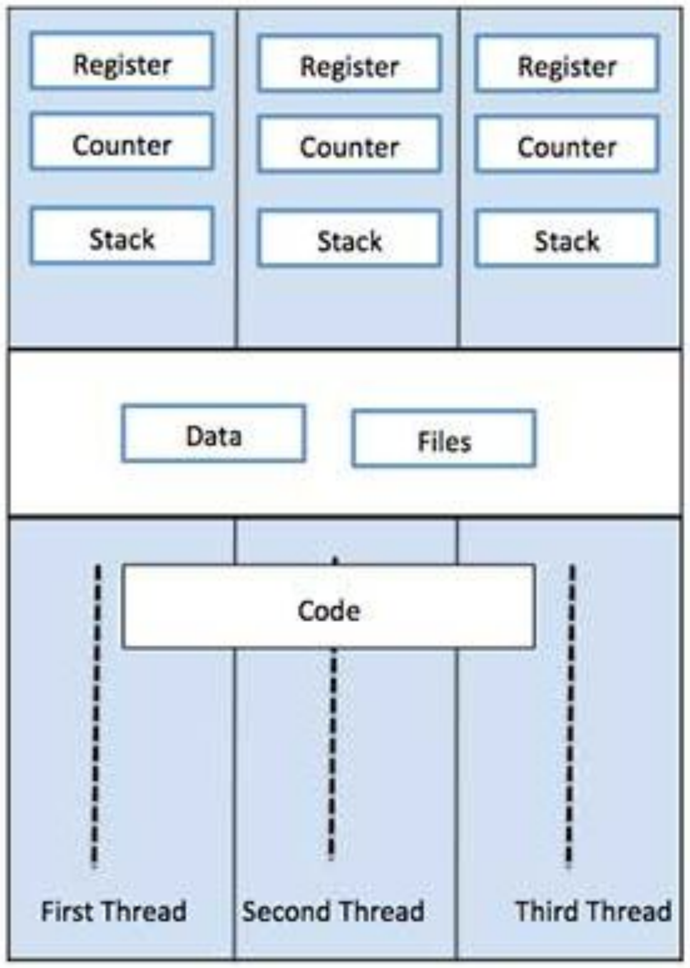


  - **program counter** that **keeps track of which instruction to execute next**.
  - **system registers** which **hold its current working variables**.
  - **stack** which **contains the execution history**.

Single Threaded Process VS Multiple Threaded Process



Single Process P with single thread



Single Process P with three threads

- A single-threaded process is a process with a single thread.
- A multi-threaded process is a process with multiple threads.
- The multiple threads have its own registers, stack and counter but they share the code and data segment.

Comparison between process and thread

Section - 7

Similarities between Process & Thread

- Like processes threads **share CPU and only one thread is running at a time**.
- Like processes threads **within a process execute sequentially**.
- Like processes thread **can create children's**.
- Like a traditional process, a thread **can be in any one of several states: running, blocked, ready or terminated**.
- Like process threads **have Program Counter, Stack, Registers and State**.

Dissimilarities between Process & Thread

- Unlike processes threads are **not independent of one another**.
- Threads within the same process **share an address space**.
- Unlike processes all threads **can access every address in the task**.
- Unlike processes threads are **design to assist one other**. Note that processes might or might not assist one another because processes may be originated from different users.

Benefits/Advantages of threads

Section - 8

Benefits/Advantages of Threads

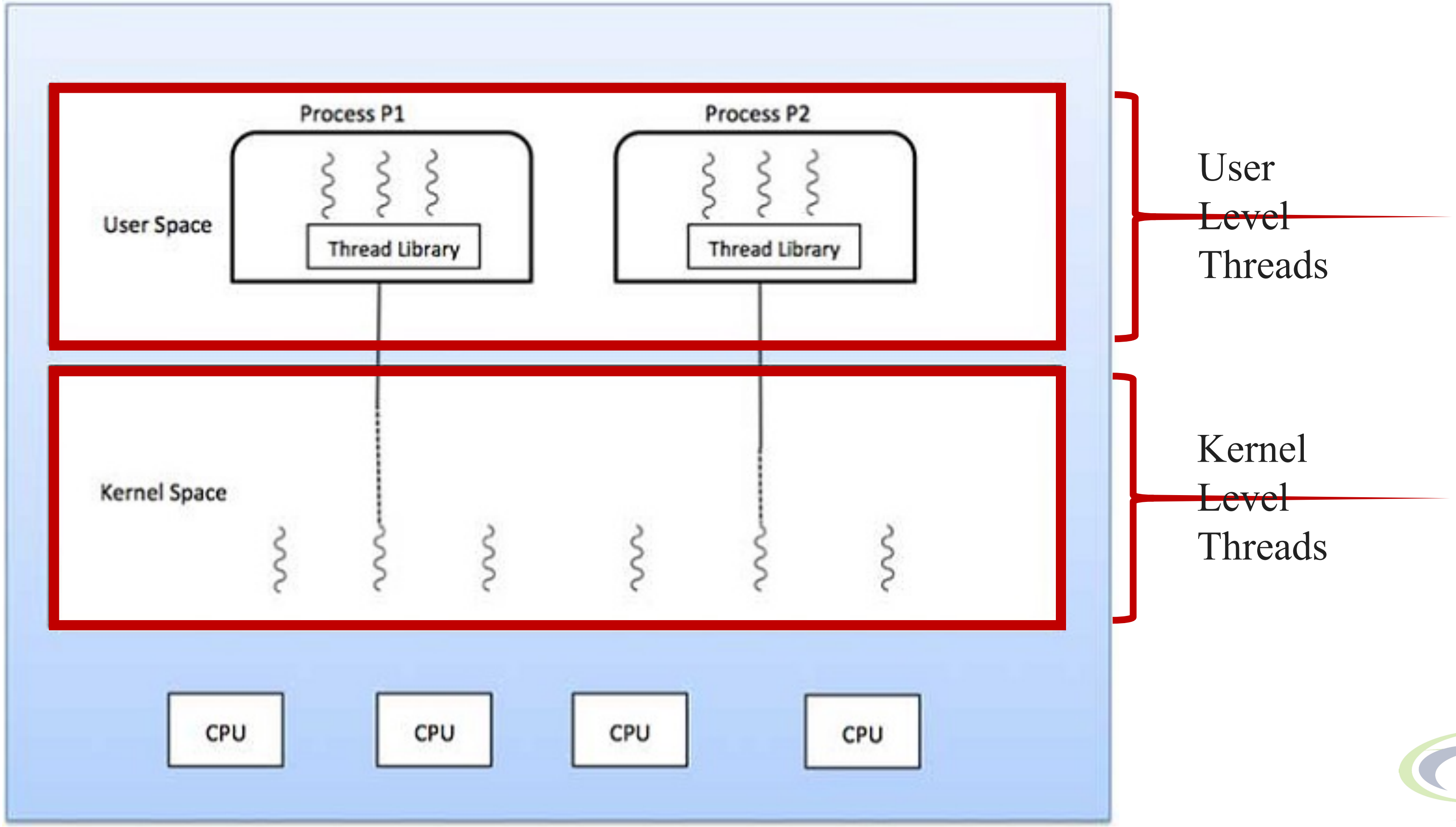
- Threads **minimize** the **context switching time**.
- Use of threads **provides concurrency** within a process.
- **Efficient communication**.
- It is more **easy to create** and **context switch** threads.
- Threads can **execute in parallel** on multiprocessors.
- With threads, an application can **avoid per-process overheads**
  - Thread creation, deletion, switching easier than processes.
- Threads have **full access to address space** (easy sharing).

Types of threads

Section - 9

Types of Threads

1. Kernel Level Thread
2. User Level Thread

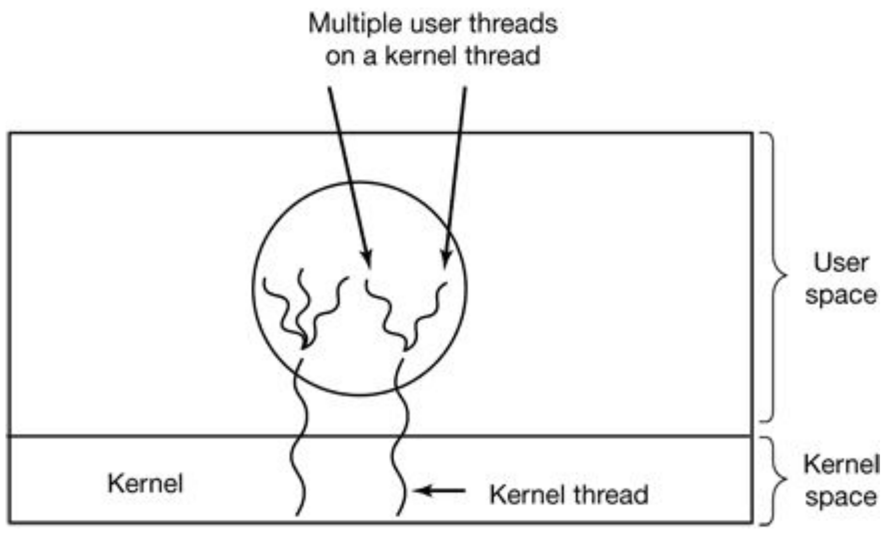


Types of Threads

User Level Thread	Kernel Level Thread
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn’t recognize user level threads.	Kernel threads are recognized by OS.
Implementation of user threads is easy.	Implementation of kernel thread is complex.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread with in same process can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris

Hybrid Threads

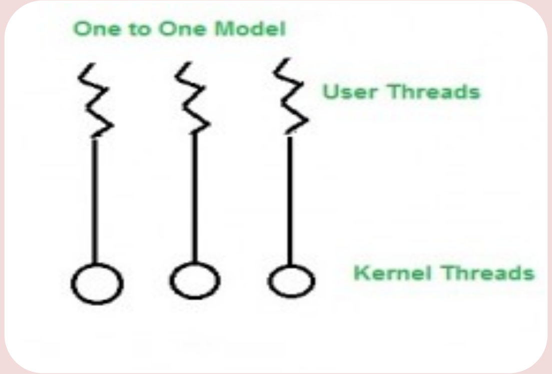
- Combines the advantages of user level and kernel level thread.
- It **uses kernel level thread** and then **multiplex user level thread on to some or all of kernel threads**.
- **Gives flexibility** to programmer that how many kernel level threads to use and how many user level thread to multiplex on each one.
- **Kernel is aware of only kernel level threads** and schedule it.

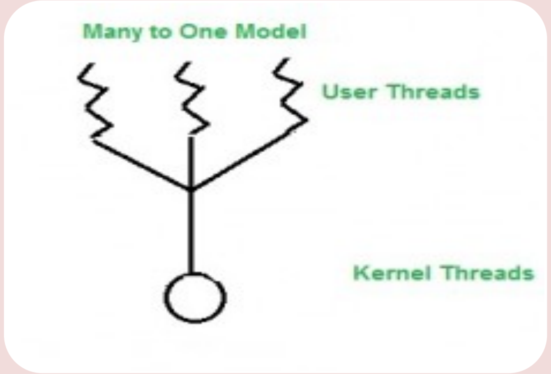


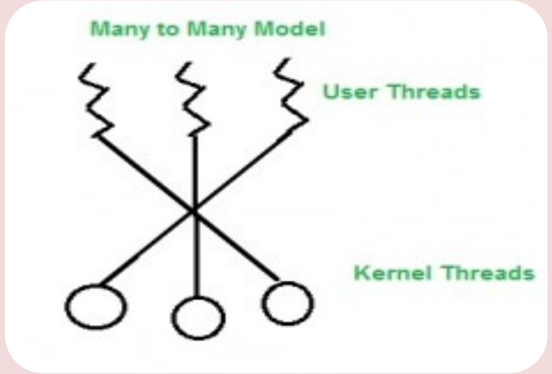
Multi Threading Models

Section - 10

Multi Threading Models







One to One Model

Each user threads mapped to one kernel thread.

Problem with this model is that creating a user thread requires the corresponding kernel thread.

Many to One Model

Multiple user threads mapped to one kernel thread.

Problem with this model is that a user thread can block entire process because we have only one kernel thread.

Many to Many Model

Multiple user threads multiplex to more than one kernel threads.

Advantage with this model is that a user thread can not block entire process because we have multiple kernel thread.

Pthread function calls

Section - 11

Pthread function calls

1. Pthread_create:- Create a new thread
2. Pthread_exit:- Terminate the calling thread
3. Pthread_join:- Wait for a specific thread to exit
4. Pthread_yield:- Release the CPU to let another thread run
5. Pthread_attr_init:- Create and initialize a thread’s attribute structure
6. Pthread_destroy:- Remove a thread’s attribute structure

System calls

Section - 12

System calls

- A system call is the **programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on**.
- A system call is a **way for programs to interact with the operating system**.
- A **computer program makes a system call when it makes a request to the operating system's kernel**.
- System call **provides the services of the operating system to the user programs via Application Program Interface(API)**.
- It **provides an interface** between a process and operating system to allow user-level processes to request services of the operating system.
- System calls are the **only entry points into the kernel system**.
- **All programs needing resources must use system calls**.

System calls

- ps (process status):- The ps (process status) command is used to **provide information about the currently running processes**, including their process identification numbers (PIDs).
- fork:- Fork system call is used for **creating a new process, which is called child process**, which runs concurrently with the process that makes the fork() call (parent process).
- wait:- Wait system call **blocks the calling process until one of its child processes exits** or a signal is received. After child process terminates, parent continues its execution after wait system call instruction.
- exit:- Exit system call **terminates the running process normally**.
- exec family:- The exec family of functions **replaces the current running process with a new process**.

Questions asked in GTU

1. Explain Process/Thread Life Cycle with diagram.
2. Explain process control block (PCB) with diagram.
3. Difference between process and thread.
4. Write various multi threading models.
5. Write benefits of threads.

Dedicated Faculty, Committed Education
Darshan
Institute of Engineering & Technology

Than k You

Prof. Firoz A Sherasiya
Computer Engineering Department
Darshan Institute of Engineering & Technology, Rajkot
✉ firoz.sherasiya@darshan.ac.in
📞 9879879861