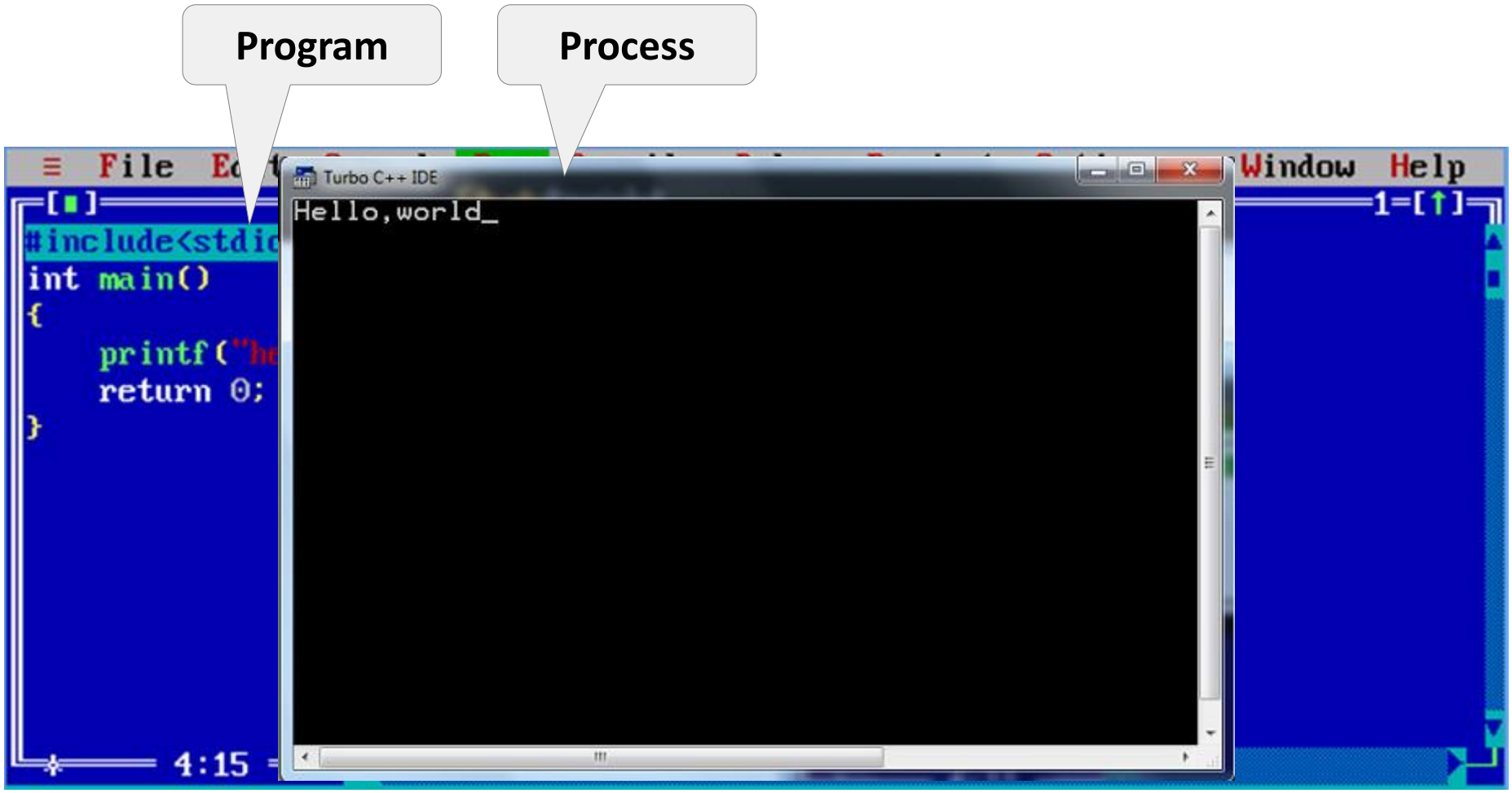
Topics to be covered

- Definition of process
- Process relationship
- Process states
- Process state transitions
- Process control
- Process control block
- Context switching

What is Process?

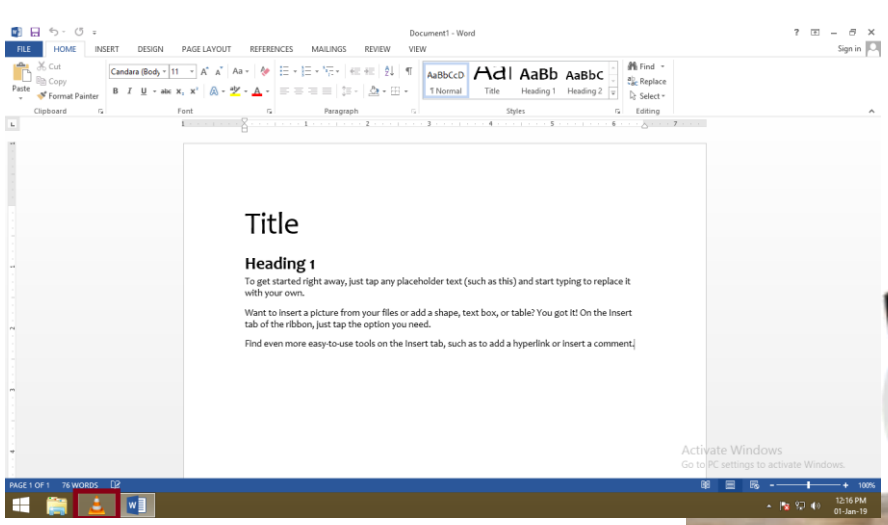


What is Process?

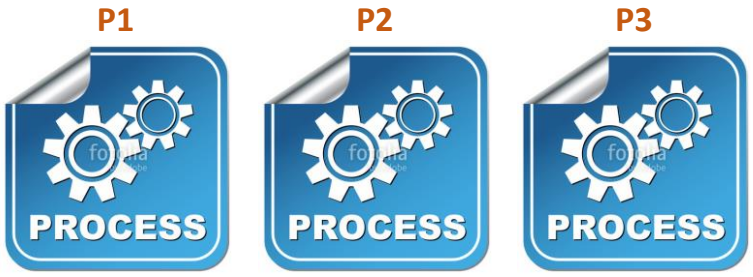
- Process is a **program under execution**.
- Process is an **abstraction of a running program**.
- Process is an **instance of an executing program**, including the current values of the program counter, registers & variables.
- Each process has its own virtual CPU.

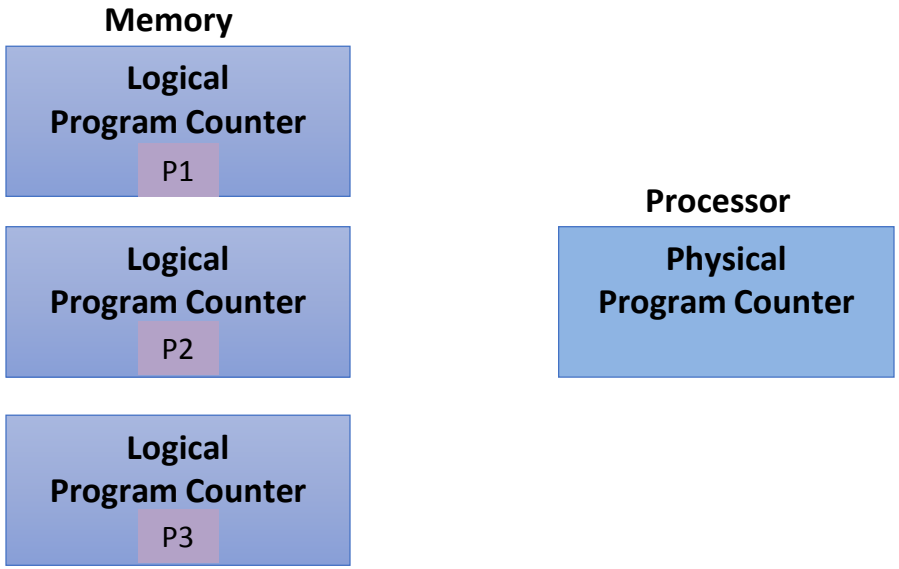
Multiprogramming

- 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**.



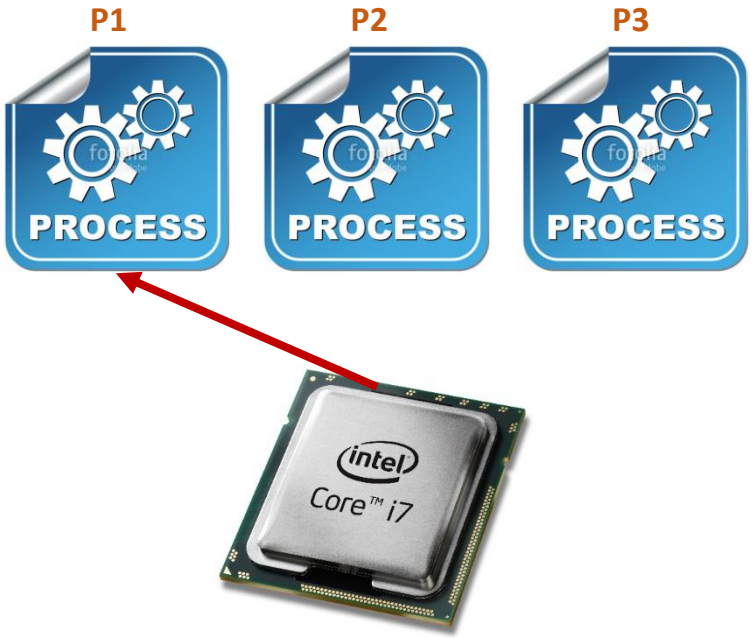
Multiprogramming execution

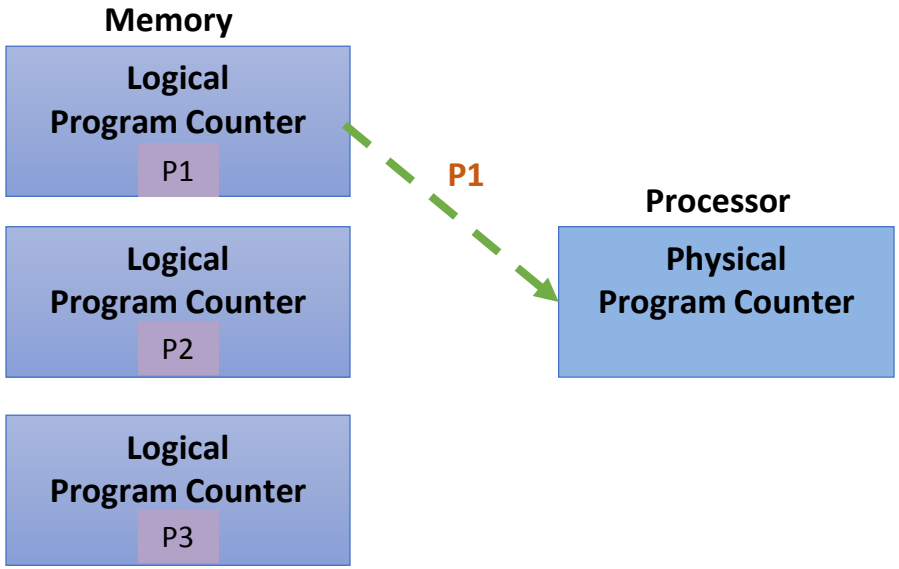




- There are **three processes**, **one processor** (CPU), **three logical program counter** (one for each processes) in memory and one physical program counter in processor.
- Here **CPU is free** (no process is running).
- No data in physical program counter.

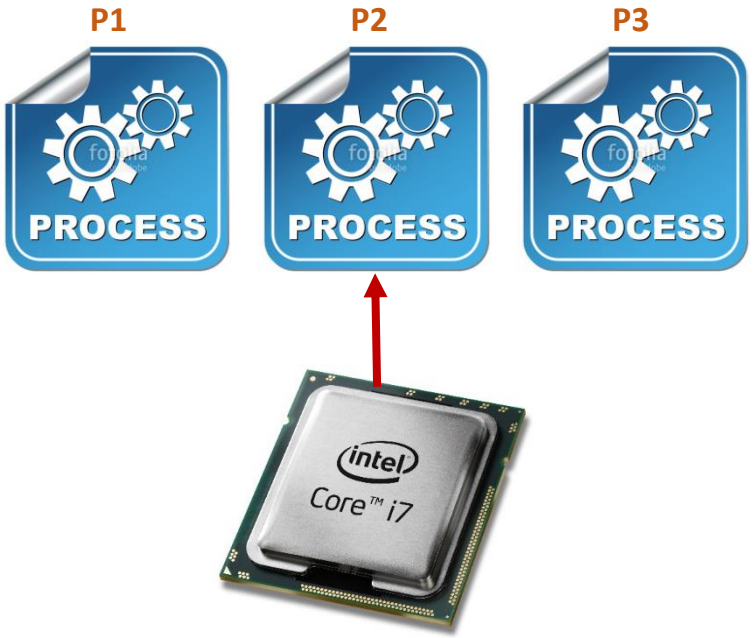
Multiprogramming execution

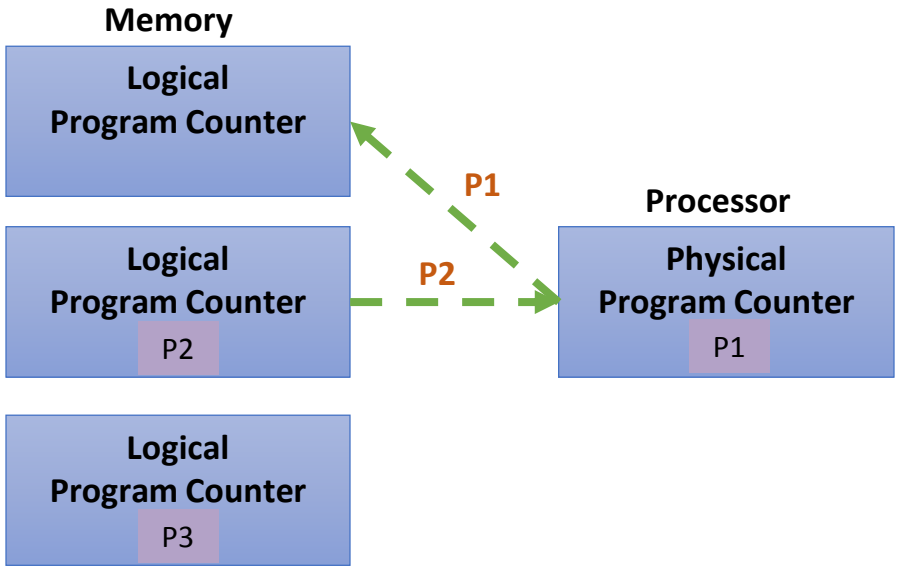




- CPU is **allocated to process P1** (process P1 is running).
- **Data of process P1 is copied** from its logical program counter to the physical program counter.

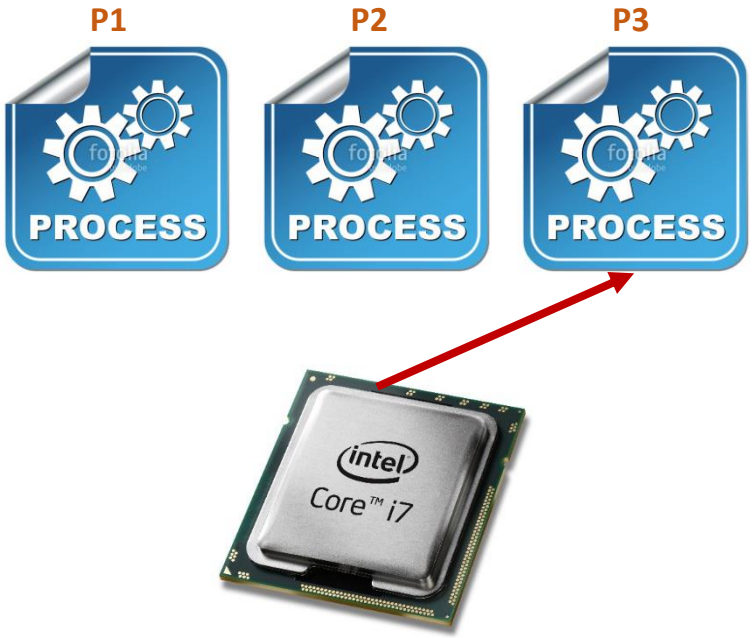
Multiprogramming execution

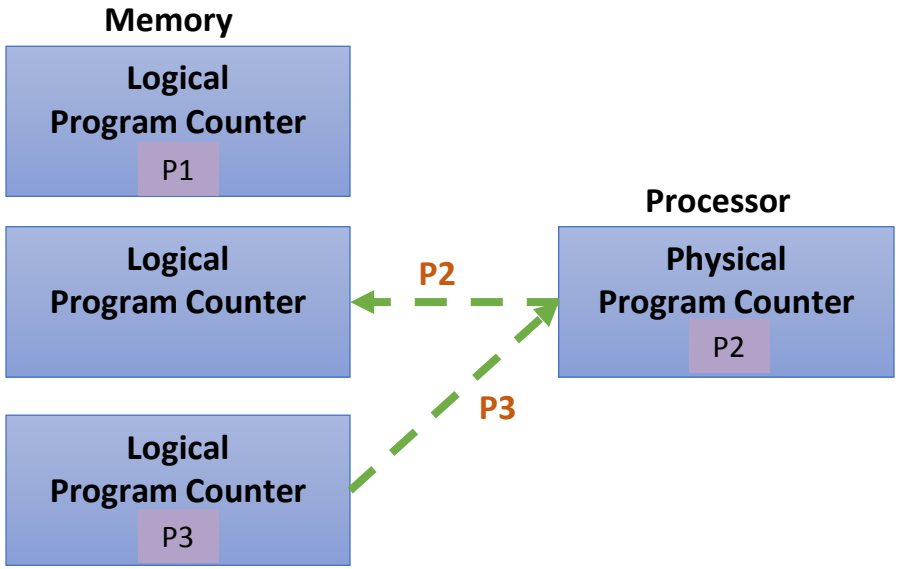




- CPU **switches** from process **P1 to** process **P2**.
- CPU is **allocated to process P2** (process P2 is running).
- **Data of process P1 is copied back** to its logical program counter.
- **Data of process P2 is copied** from its logical program counter to the physical program counter.

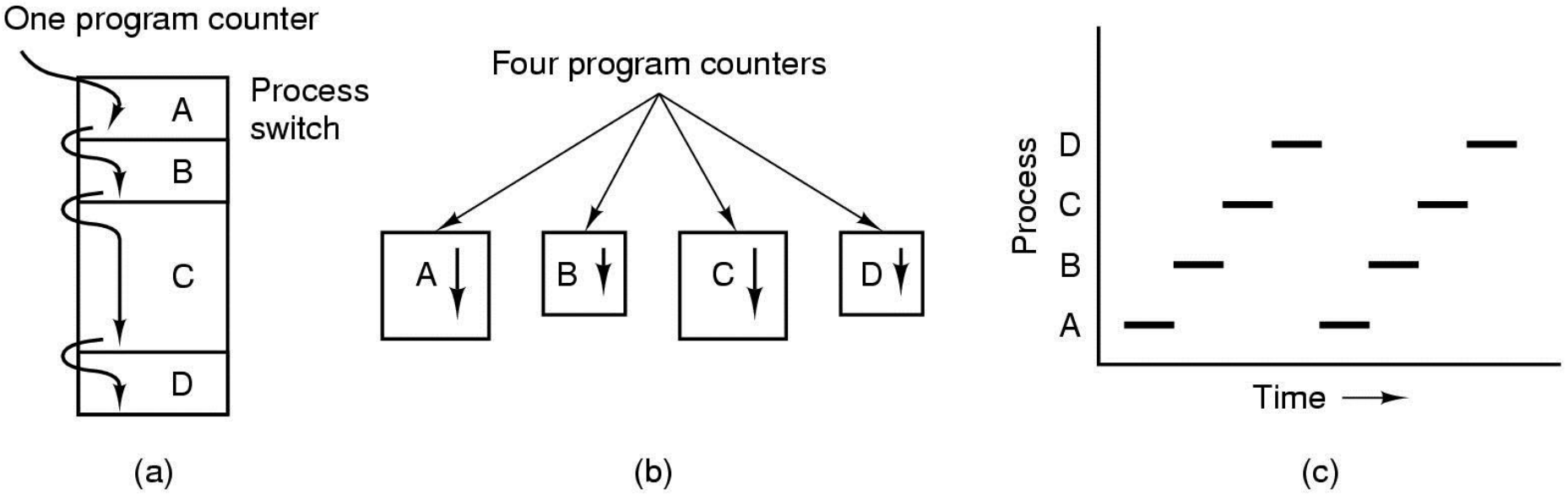
Multiprogramming execution





- 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



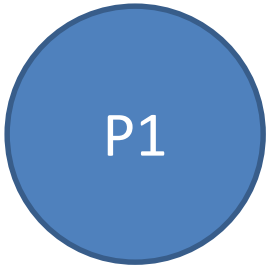
- Fig. (a) Multiprogramming of **four programs in memory**
- Fig. (b) **Conceptual model of 4 independent, sequential processes**, each with its own flow of control (i.e., its own logical program counter) and each one running independently of the other ones.
- Fig. (c) over a long period of time interval, 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**

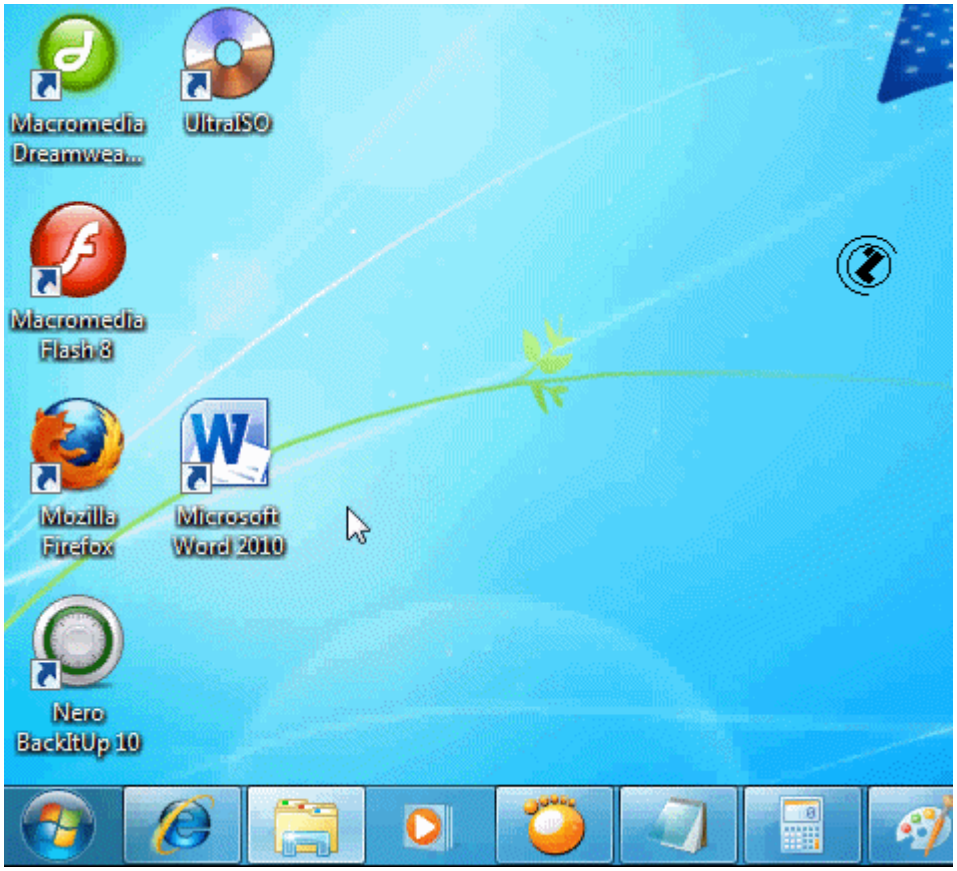
Process Creation

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 (Cont…)

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**.

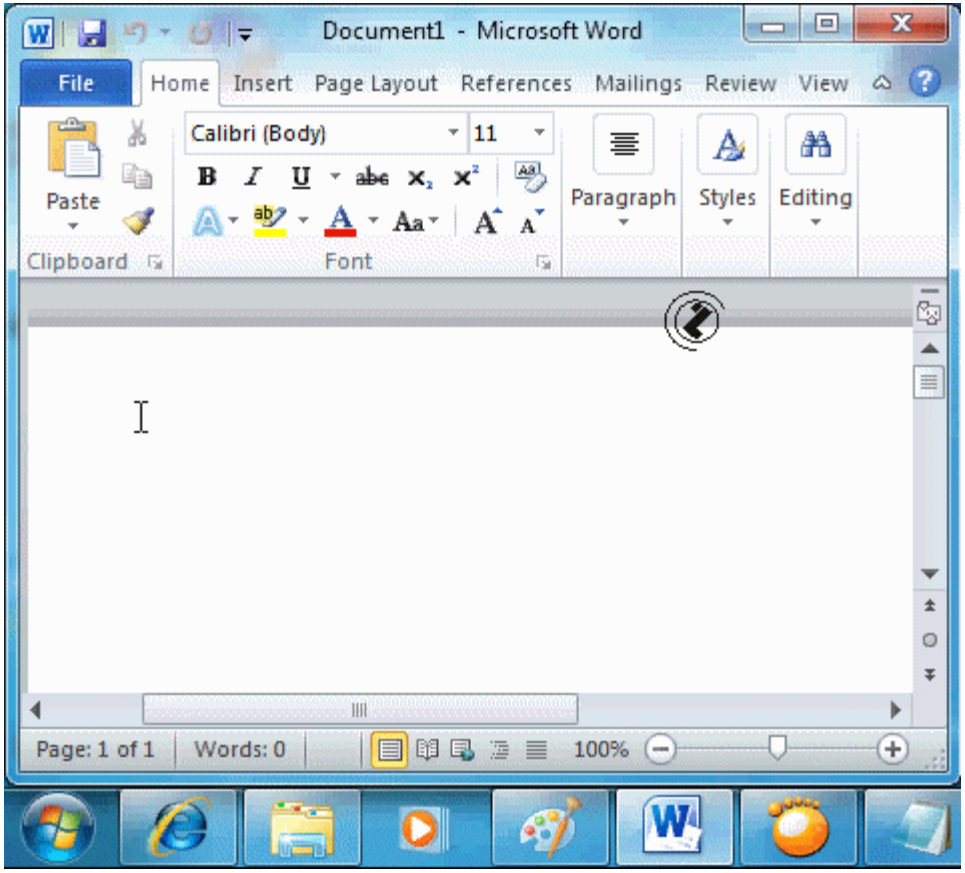


Process Creation (Cont…)

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

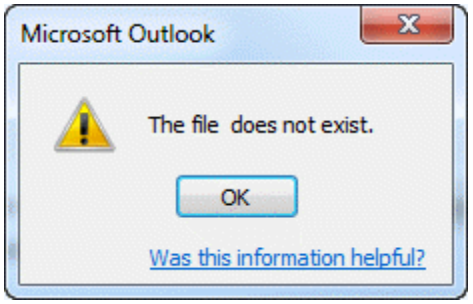
Process Termination

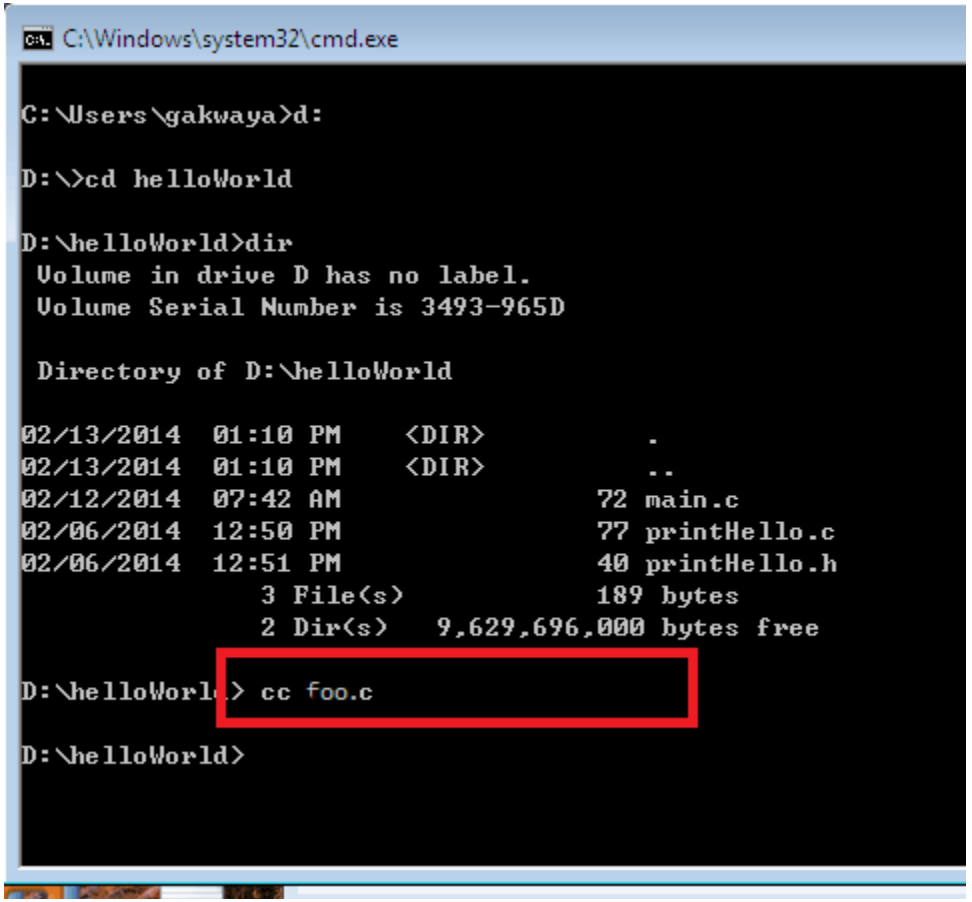
1. Normal exit (voluntary)
   - Terminated because **process** has **done its work**.



Process Termination

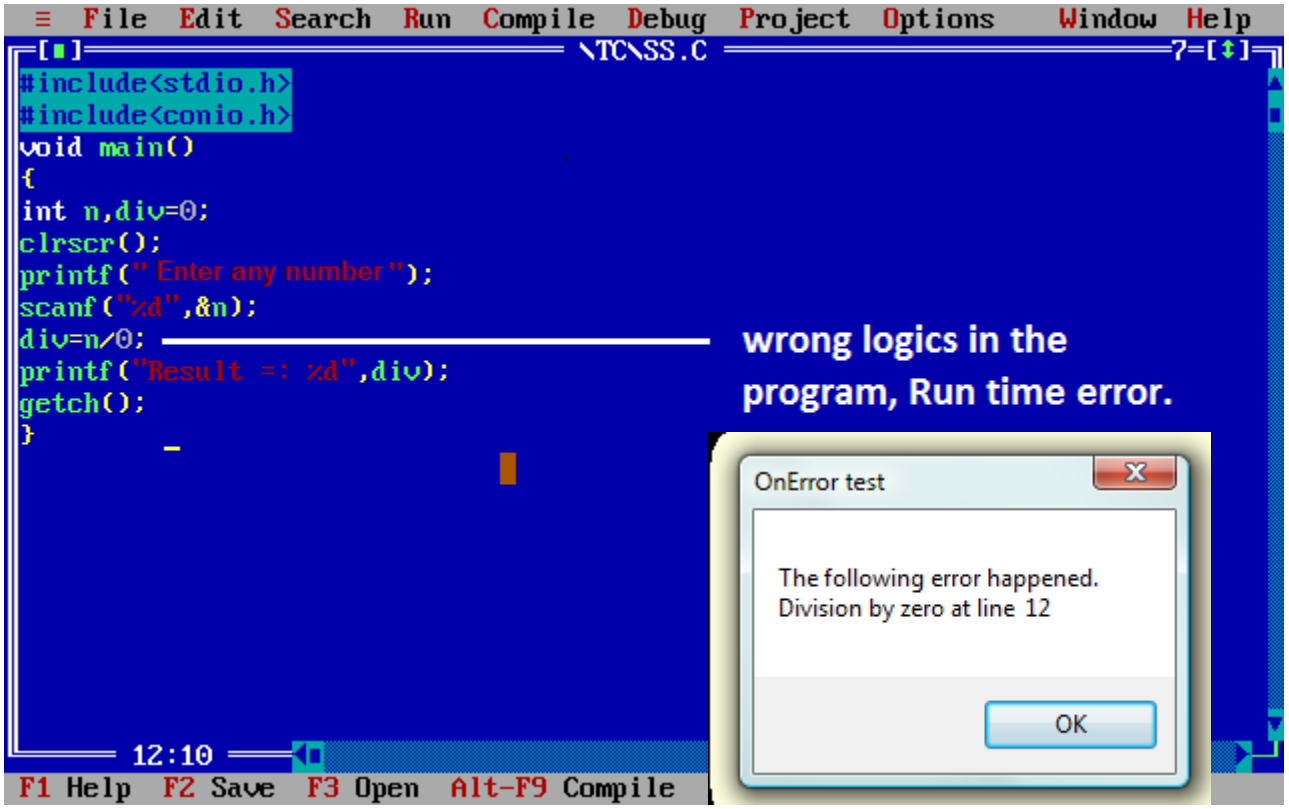
2. Error exit (voluntary)
   - The process **discovers a fatal error** e.g. user types the command **cc foo.c** to compile the program foo.c and no such file exists, the compiler simply exit.





Process Termination

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.

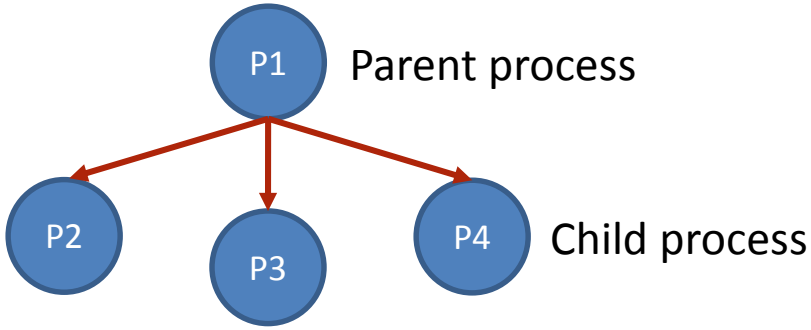


Process Termination

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

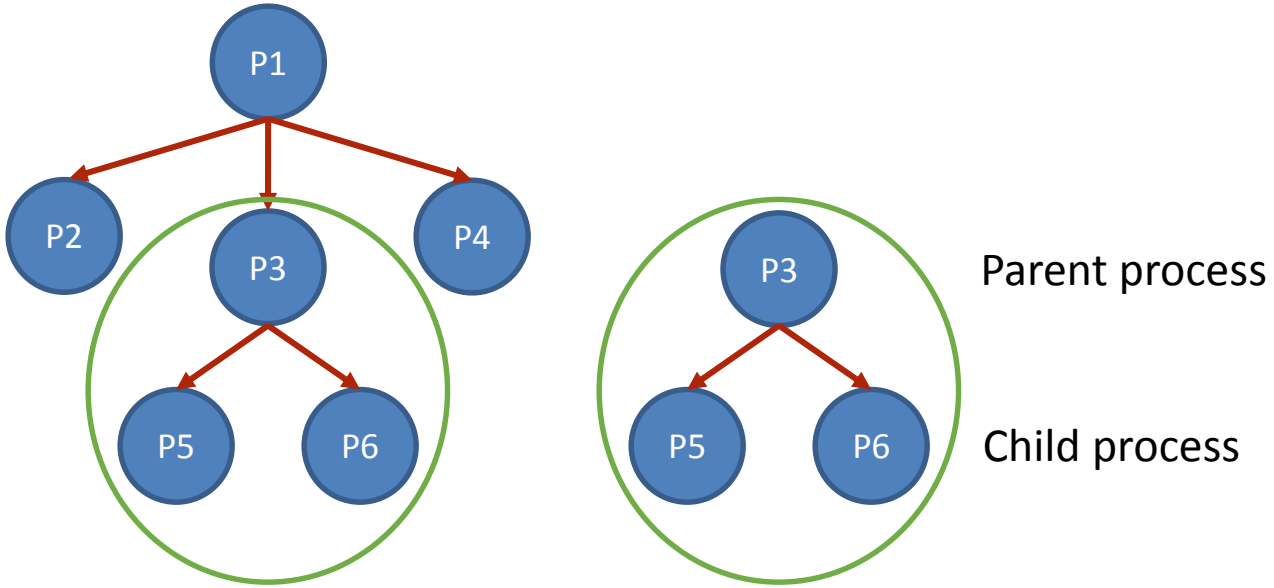
Process Hierarchies

- Parent process can create child process, child process can create its own child process.



Process Hierarchies

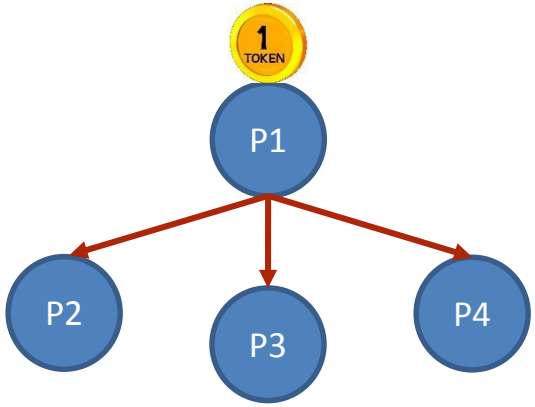
- Parent process can create child process, child process can create its own child process.



- **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)

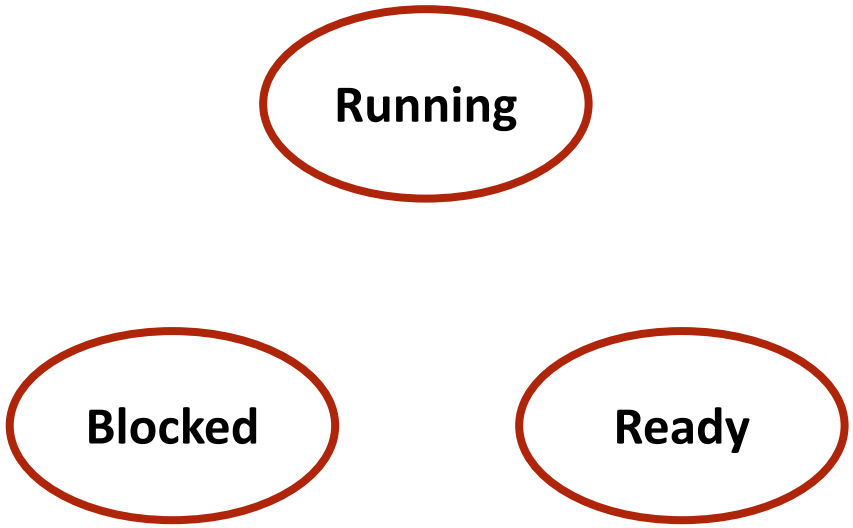
Handle

- 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.



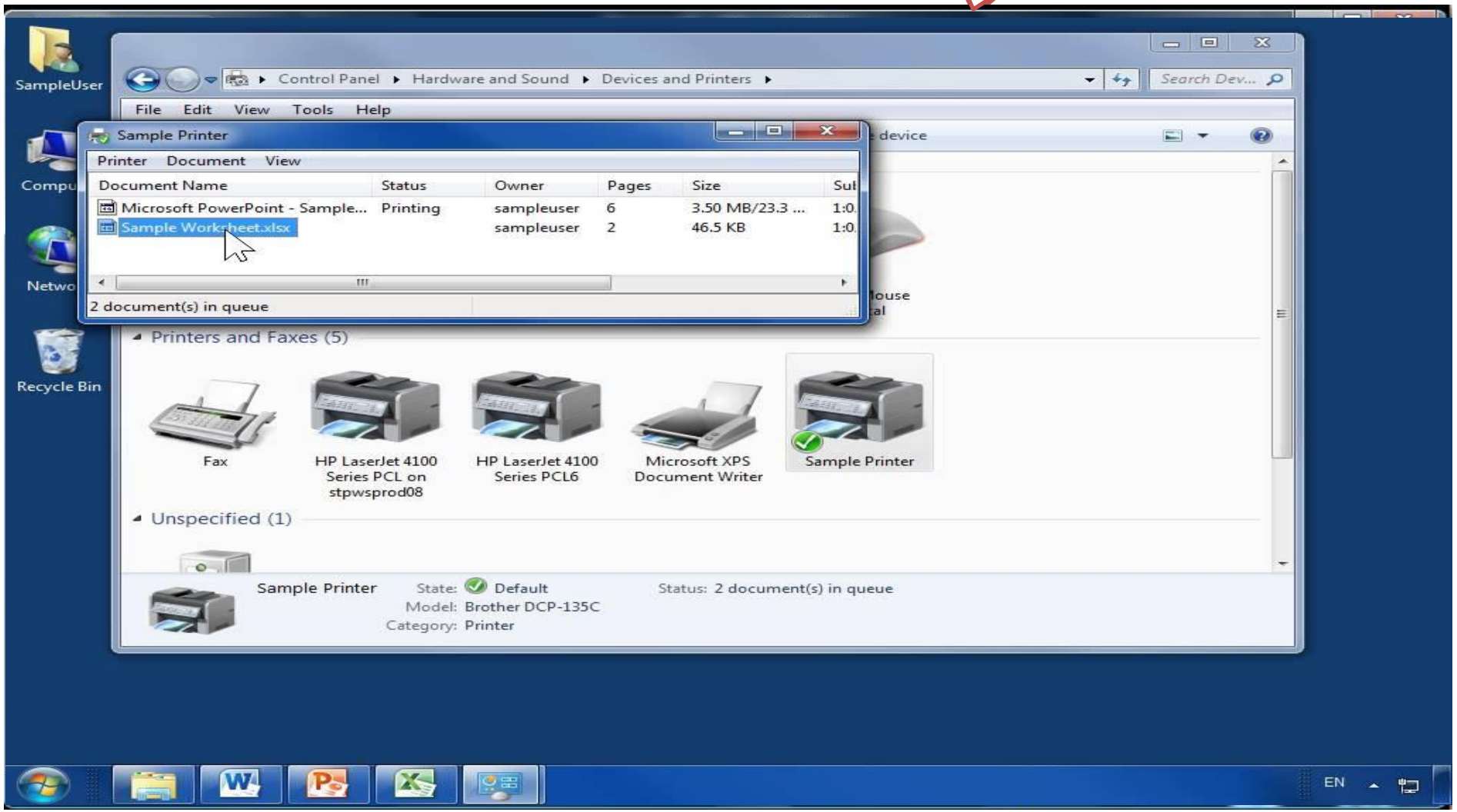
Process State

1. **Running** – Process is actually **using the CPU**
2. **Ready** – Process is runnable, **temporarily stopped to let another process to run**
3. **Blocked** – process is **unable to run until some external event happens**



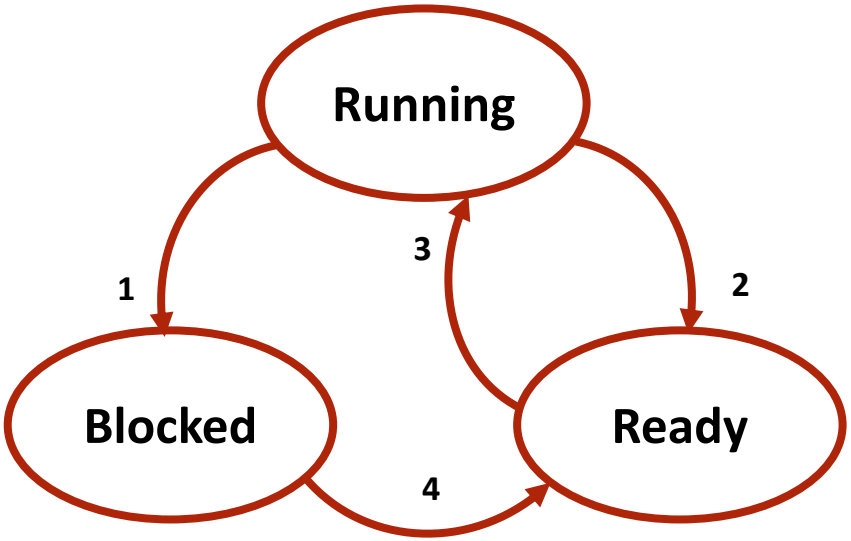
Processes are always **either executing (running) or waiting to execute (ready) or waiting for an event (blocked)** to occur.

Process State

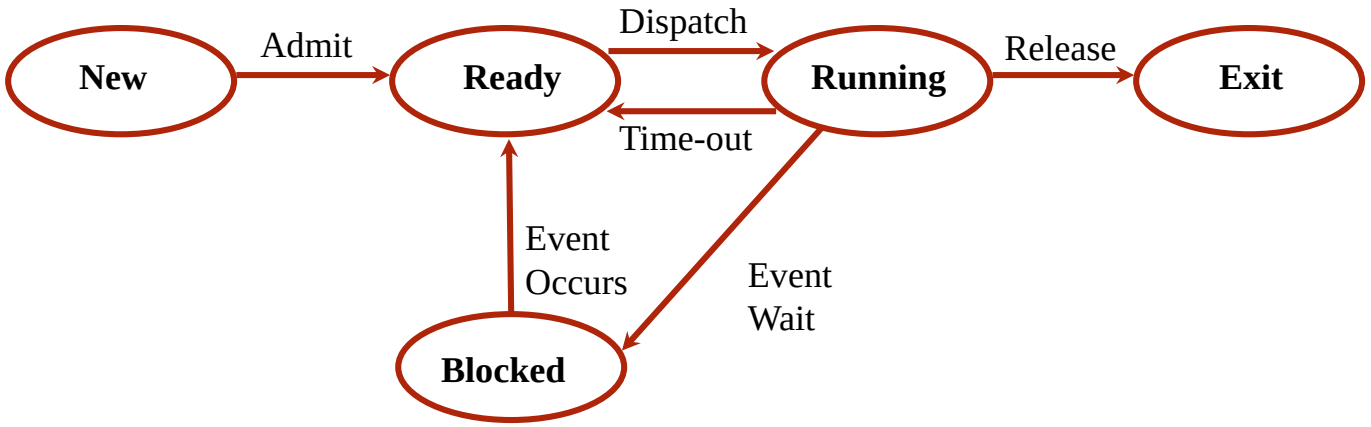


Process State Transitions

- 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)

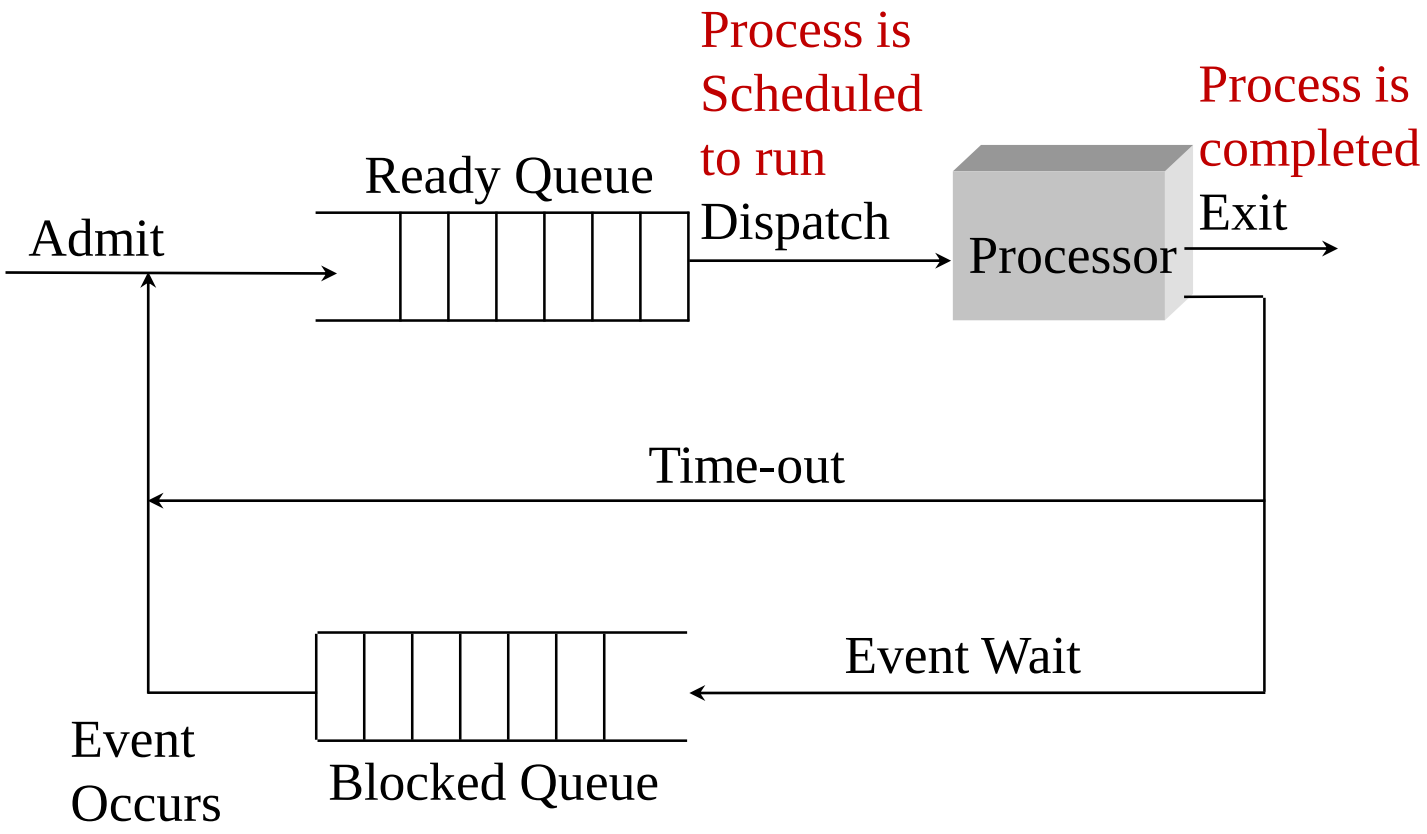


Five State Process Model and Transitions



- 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

Queue Diagram



Process Control Block (PCB)

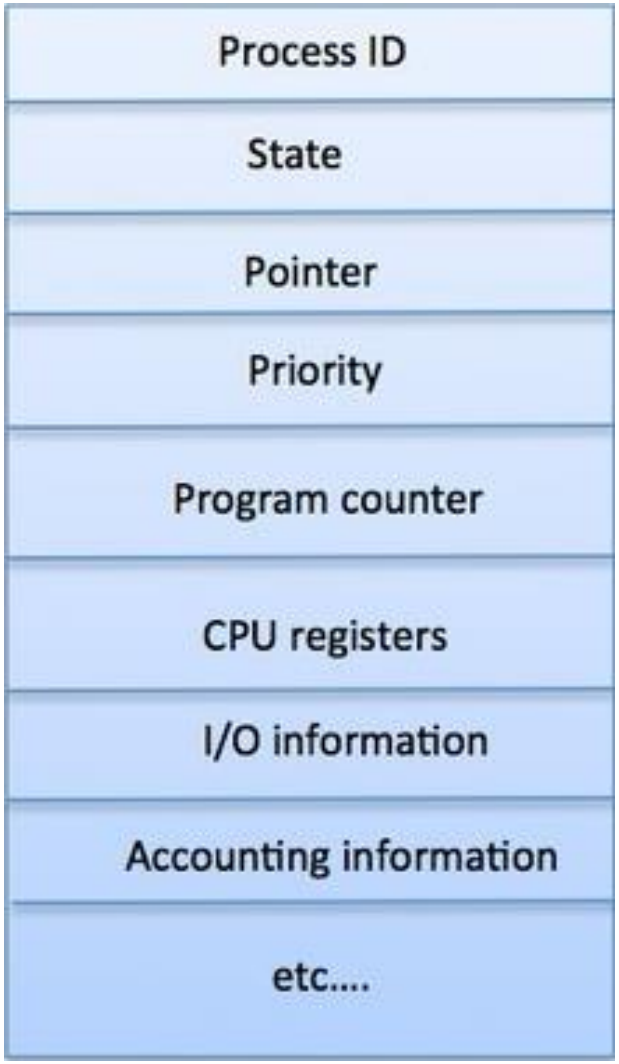
- A Process Control Block (PCB) is a **data structure maintained by the operating system for every process**.
- PCB is **used for storing the collection of information** about the processes.
- The PCB is **identified by** an **integer process ID** (**PID**).
- A PCB keeps all the information needed to keep track of a process.

Process Control Block (PCB)

- 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.

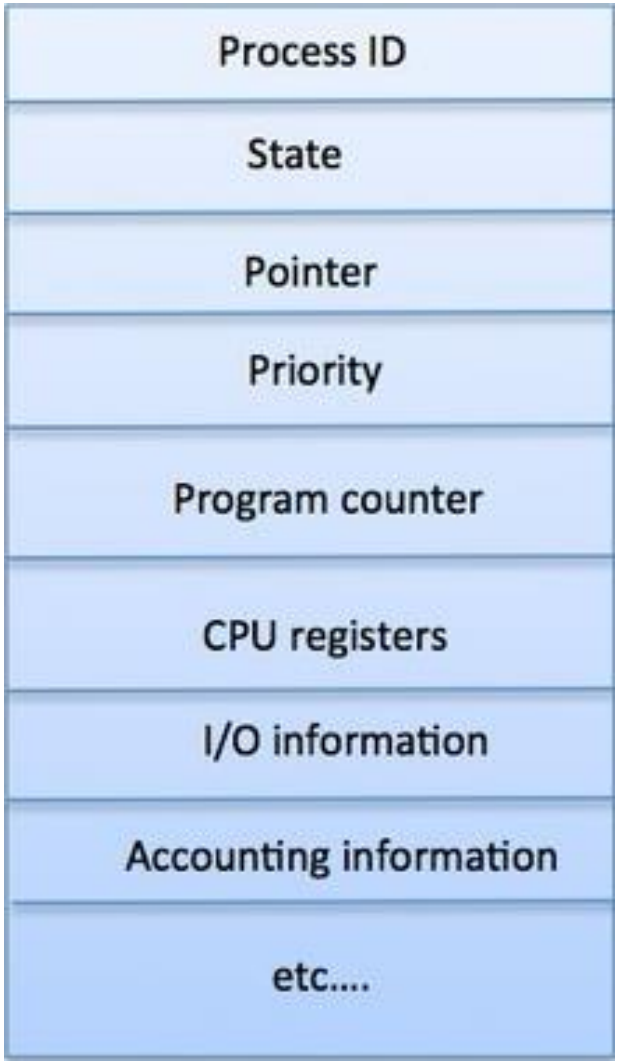
Process Control Block (PCB) contains

- **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.



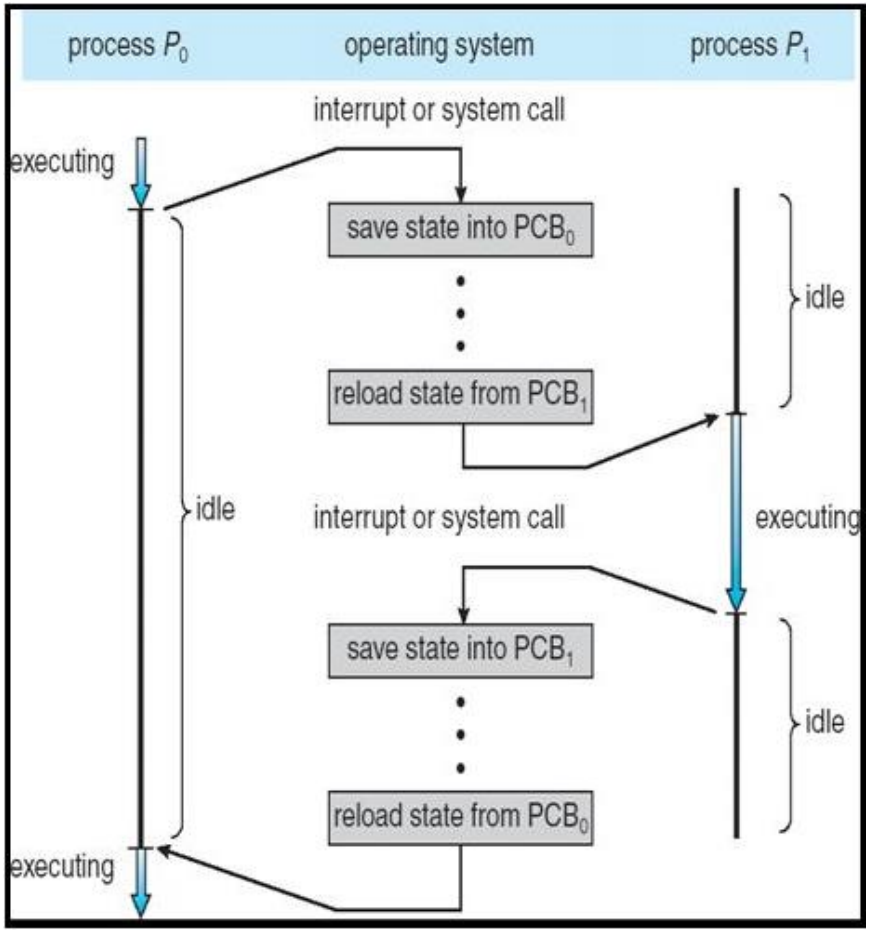
Process Control Block (PCB) contains

- **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

- Context switch means **stopping one process** and **restarting another process.**
- When an event occur, the OS **saves the state of an active process** and **restore the state of new process**.
- Context switching is **purely overhead** because system does not perform any useful work while context switch.



Steps performed by OS during Context switching

- 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**