CSE3241: Operating System and System Programming

Class-Process

Sangeeta Biswas, Ph.D.

Assistant Professor
Dept. of Computer Science and Engineering (CSE)
Faculty of Engineering
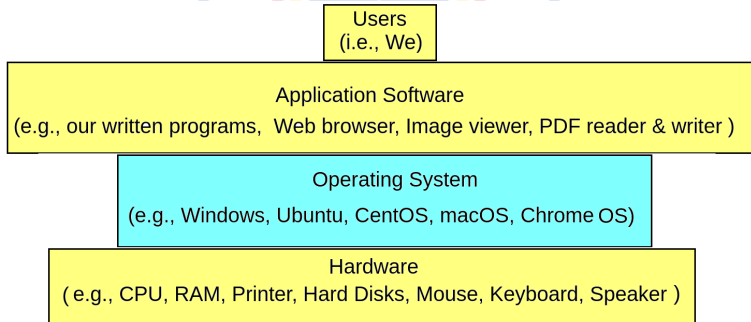University of Rajshahi (RU)
Rajshahi-6205, Bangladesh
E-mail: sangeeta.cse@ru.ac.bd

What is Operating System?

Operating System (OS) is a system software which-
- ▶ manages computer resources (harware, software) and
- ▶ provides an environment where application software can run in order to full-fill users' demands.

As shown in Figure, an OS acts as a bridge between hardware and software that we run to access hardware.

Users (i.e., We)

Application Software
(e.g., our written programs, Web browser, Image viewer, PDF reader & writer)

Operating System
(e.g., Windows, Ubuntu, CentOS, macOS, Chrome OS)

Hardware
(e.g., CPU, RAM, Printer, Hard Disks, Mouse, Keyboard, Speaker)

Figure: A computer System

Multitasking OS

Multi-tasking OS allows a user to perform multiple computer tasks using a single set of resources in such a way that user get pseudo-parallel feeling, i.e., the user feels all of her/his tasks are running in parallel.

▶ Almost all modern OS, such as Microsoft Windows 2000, IBM's OS/390, and Linux, have multi-tasking capability.

▶ The main aim of multitasking is to ensure maximum utilization of advanced CPU, which is much more faster than the old time CPU, by keeping it busy at the maximum time.

▶ In multitasking, OS switches execution power from one process to another frequently so that each process has a progress instead of waiting until a specific process completely finished.

▶ OS needs to keep track of all processes to do multitasking smoothly, therefore, it uses **Process Control Block**.

Concepts of Process

1. Process is a program in execution.

2. Program is a passive entity stored on disk (executable file); process is an active entity.

3. Program becomes process when an executable file is loaded into memory.

4. One program can have several processes
   - multiple users or one user can execute multiple copies of the same program at a time.

5. Process execution must progress in sequential fashion.
   - No parallel execution of instructions of a single process.

6. Modern OS manages thousand of processes of a variety of application software, system software as well as its own processes at a time.

Check Process in Linux based OS
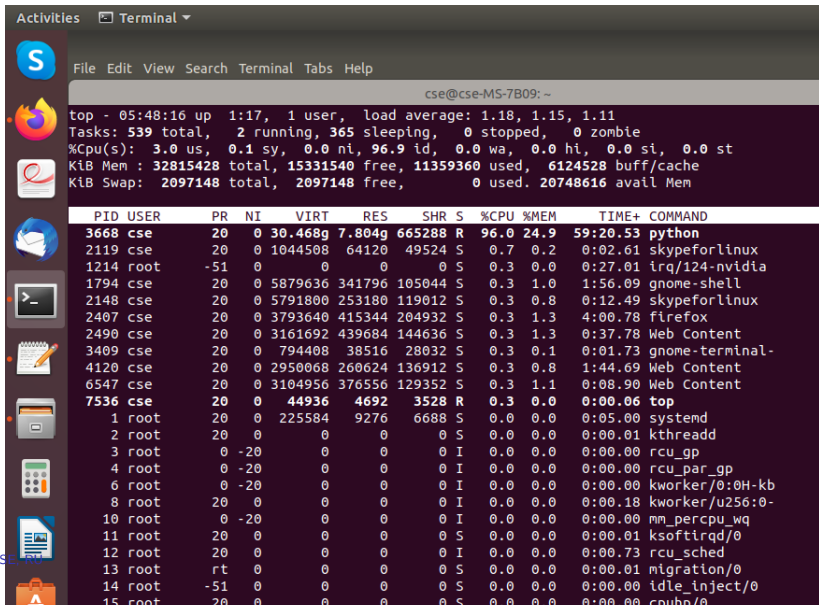
To see all processes, currently managed by a Linux based OS (e.g., Ubuntu), at real time:

- ▶ Open a terminal, the black screen, by typing Ctrl + Alt + T .
- ▶ Type 'top' and press Enter. Press 'Q' to exit.
- ▶ $ top

To get a snapshot of all processes, currently managed by a Linux based OS (e.g., Ubuntu): $ ps aux

Check Process in Linux based OS

$ top

Check Process in Linux based OS

$ ps aux

PID

PID or Process number is a unique number of a process assigned by the OS so that it can easily keep track all current processes.

- ▶ It is only valid until the process is properly terminated in the current session.
    - ▶ If ,for some reasons, a process is killed/ finished execution, but do not get a chance to properly inform its parent process, then it holds the process ID.
- ▶ For a new start of a program, it gets a new PID.
    - ▶ So, if we run our code at different time/ different sessions, our program will become a new process and get a different PID.
- ▶ As long as a process has its PID, it has an entry in the **Process Table**.

Investigate PID and PCB in Ubuntu

Commands to see PID and
PCBs of all processes,
currently managed by the OS
including its own processes:

▶ For PID:
  ▶ $ top
▶ For PCB:
  ▶ $ ls /proc
  ▶ $ ls /proc/<PID>/
  e.g., $ ls /proc/3668

Know PID of Our Program

Run the following program in one terminal and check its PID and
parent's PID in another terminal:

- ▶ $ sudo apt install htop [if htop is not installed]
- ▶ $ htop
- ▶ $ pstree -ps <PID>
- ▶ $ kill -9 <PID> [terminate infinitely running process.]

```c
1 #include<stdio.h>
2
3 int main(){
4         for(int i = 0; i <1; i--)
5                 printf("Hi\n");
6
7         return 0;
8 }
```

Figure: InfiniteLoop.c

Know PID Inside Our Program

Run the following program again and again and see PIDs.

- ▶ getpid() is used to know PID of the process when the executable file of this C code runs.
- ▶ getppid() is used to know parent's PID of the process.

```c
1 #include<unistd.h>
2 #include<stdio.h>
3
4 int main(){
5         pid_t myPID, parentPID;
6
7         myPID = getpid();
8         parentPID = getppid();
9
0         printf("PID of this process: %u\n", myPID);
1         printf("PID of parent process: %u\n", parentPID);
2
3         return 0;
4 }
```

Figure: PID.c

Process Tree

Processes are arranged in a tree structure, therefore, except the root process, each process has a parent process and $0 - n$ number of child processes.

In Ubuntu:

- **sched** has PID: 0.
- **init / systemd** has PID: 1.
    - it is directly or indirectly the parent process of all processes.
    - it starts as soon as the computer starts and continue running till, it is shutdown.

To see the process tree, type:

- $ pstree
- $ pstree -p
- $ pstree -ps <PID> [e.g., $ pstree -ps 1656]

Schematic diagram of Process Tree in Ubuntu
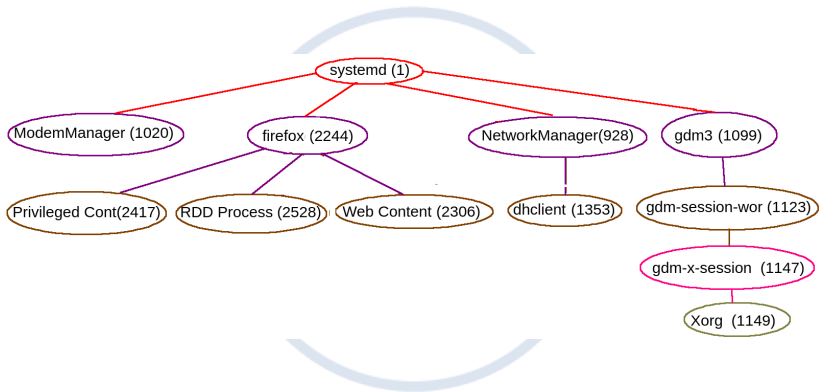
Process Creation: fork()

- When fork() is executed successfully:
  - ▶ Linux makes two identical copies of address spaces, one for the parent process and the other for the child process.

  - ▶ Both processes starts their execution at the next statement following the fork() call.

  - ▶ Since both processes have identical but separate address spaces, those variables initialized before the fork() call have the same values in both address spaces.

  - ▶ Since every process has its own address space, any modifications will be independent of the others.
- Return value of fork(): **Child_PID/ 0 / -1**.

Parent-Child

■ Both parent and child process will start their execution at the next statement following the fork() call.

Parent

```
#include<unistd.h>
#include<stdio.h>

int main(){
    int x;
    pid_t myPID, childPID;

    x = 10;
    myPID = getpid();
    childPID = fork();

→   printf("How are you?");

    return 0;
}
```

myPID	childPID
2784	2785

x
10

Child

```
#include<unistd.h>
#include<stdio.h>

int main(){
    int x;
    pid_t myPID, childPID;

    x = 10;
    myPID = getpid();
    childPID = fork();

→   printf("How are you?");

    return 0;
}
```

myPID	childPID
2784	0

x
10

Process Control Block

Process Control Block (PCB) is a block of information that represent a process in an OS. It is also known as a **task control block**.

- **Process state**: running, waiting, etc.
- **Program counter**: location of instruction to next execute.
- **CPU registers**: contents of all process-centric registers
- **CPU scheduling information**: priorities, scheduling queue pointers.
- **Memory limits**: memory allocated to the process.
- **Accounting information**: CPU used, clock time elapsed since start, time limits.
- **I/O status information**: I/O devices allocated to process.

Pointers	PID
Process State	
Program Counter	
Registers	
Memory Limits	
CPU Scheduling Information	
Memory Management Information	
List of Open Files	
Accounting Information	
I/O status Information	
- - - - - - - - - - - - -	
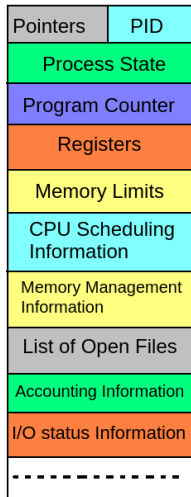
Process Table

Process table is simply a data structure in the RAM of a computer. It holds information about the processes that are currently handled by the OS.

▶ OS maintains pointers to each PCB in the process table so that it can access the PCB quickly.

Figure: PCB table

Process States

As a process executes, it changes state

- ▶ **New**: The process is being created
- ▶ **Running**: Instructions are being executed
- ▶ **Waiting**: The process is waiting for some event to occur
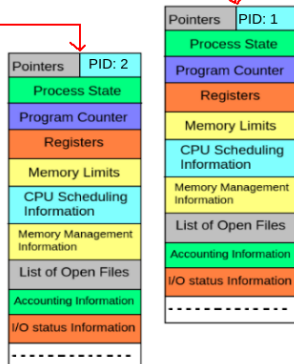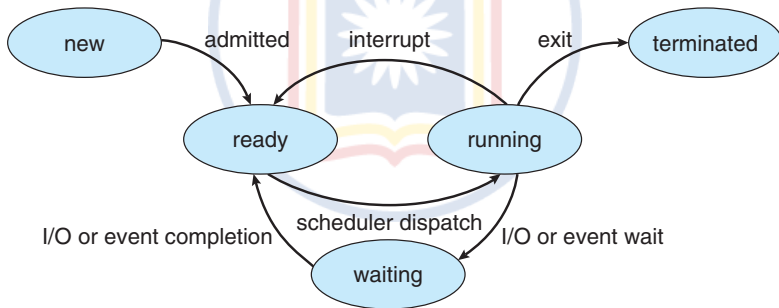- ▶ **Ready**: The process is waiting to be assigned to a processor
- ▶ **Terminated**: The process has finished execution

Process State Code in Ubuntu

As a process executes, it changes state

- ▶ D : uninterruptible sleep (usually IO)
- ▶ R : running or runnable (on run queue)
- ▶ S : interruptible sleep (waiting for an event to complete)
- ▶ T : stopped by job control signal
- ▶ t : stopped by debugger during the tracing
- ▶ W : paging (not valid since the 2.6.xx kernel)
- ▶ X : dead (should never be seen)
- ▶ Z : defunct ("zombie") process, terminated but not reaped by its parent

Meaning of additional characters:

- ▶ $<$: high-priority (not nice to other users)
- ▶ N : low-priority (nice to other users)
- ▶ L : has pages locked into memory (for real-time and custom IO)
- ▶ s : is a session leader
- ▶ l : is multi-threaded
- ▶ $+$: is in the foreground process group

Change of Process States
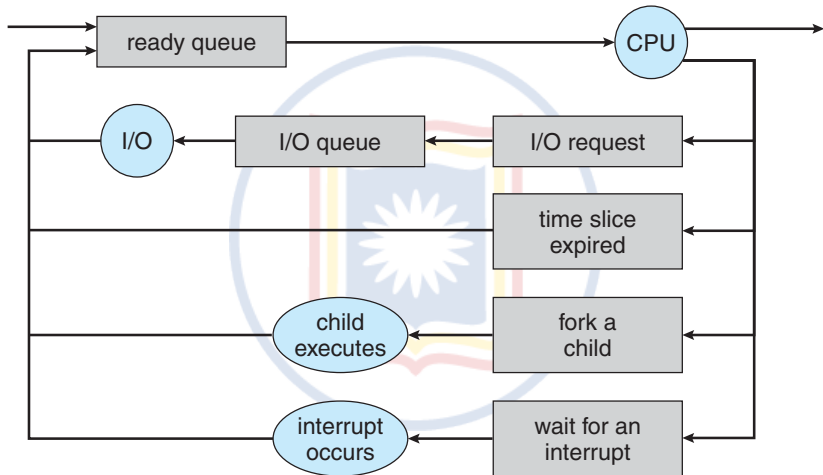
Swap Out and Swap In

Swapping is a memory management scheme in which any process is temporarily moved from the RAM to hard disk so that the RAM can be made available for other processes.

▶ Swap-out is a method of removing a process from RAM and adding it to the hard disk.

▶ Swap-in is a method of removing a program from a hard disk and putting it back into the main memory or RAM.
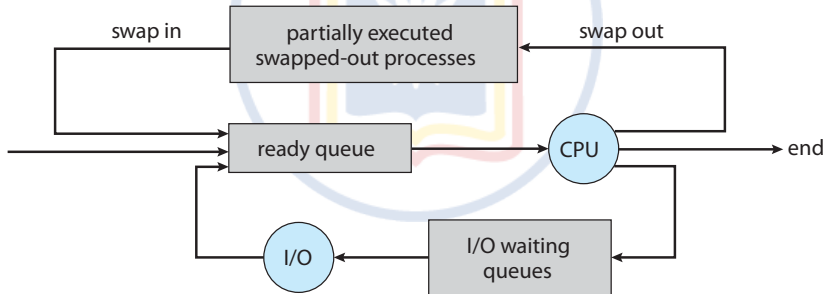
Context Switch

A context switch occurs when the CPU switches from one process to another.

▶ OS saves the state of the old process and load the saved state for the new process via a context switch.
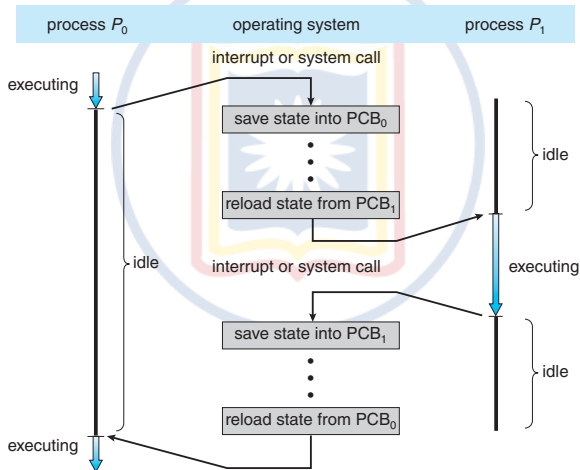
Process in Memory

A process has multiple parts, when it is in memory:

- ▶ **Text**: contains program code

- ▶ **Stack**: contains temporary data
  - ▶ Function parameters, return addresses, local variables

- ▶ **Data**: contains global variables

- ▶ **Heap**: contains dynamically allocated memory during run time

max

stack
↓
↑
heap
data
text

0

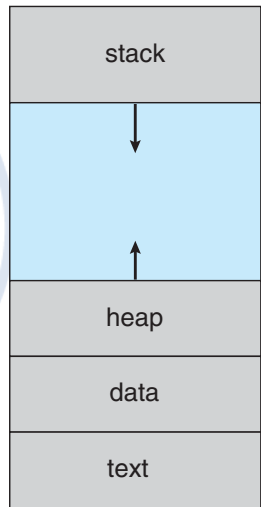# Memory Layout of a C Program

high memory	argc, agrv
	stack
	heap
	uninitialized data
	initialized data
low memory	text

```c
#include <stdio.h>
#include <stdlib.h>

int x;
int y = 15;

int main(int argc, char *argv[])
{
    int *values;
    int i;

    values = (int *)malloc(sizeof(int)*5);

    for(i = 0; i < 5; i++)
        values[i] = i;

    return 0;
}
```
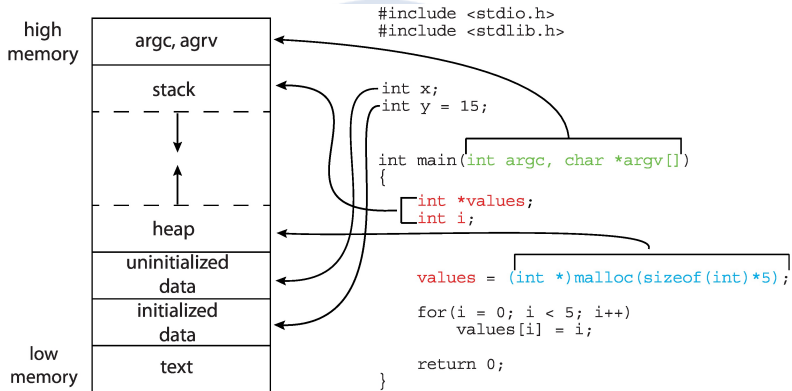
Inter-Process Communication (IPC)

Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.

- ▶ Regular File
- ▶ Anonymous Pipe
- ▶ Named Pipe or FIFO
- ▶ Message Passing
- ▶ Socket
- ▶ Signal
- ▶ Shared Memory
- ▶ Semaphore

Pipe-1

■ Pipe is the oldest mechanism for IPC in Unix.

■ There are two types of pipe:

1. Ordinary or unnamed pipe:
   ▶ it cannot be accessed outside the process that creates it.
   ▶ parent-child relationship is necessary between the communicating processes.
   ▶ it exists only while the processes are communicating with one another.
   ▶ communication is unidirectional.

2. Named pipe:
   ▶ it can be accessed by any number of processes.
   ▶ no parent-child relationship is necessary for communication.
   ▶ it exists until it is deleted from the file system.
   ▶ communication can be bidirectional.