Concepts of Operating System

OBJECTIVE

To introduce Operating System concepts with Linux environment, and to learn Shell Programming.

Course Content

https://www.cdac.in/index.aspx?id=DAC&courseid=0#

- Introduction to OS
- Introduction to Linux
- Shell Programming
- Process management
- Signals
- Threads
- Memory management
- Virtual Memory
- Deadlock
- Semaphore
- Inter process communication

Text Books

- Operating Systems Principles by Abraham Silberschatz, Peter Galvin & Greg Gagne / Wiley

  https://os-book.com/OS9/index.html

- Unix Concepts and Applications by Sumitabha Das / McGraw Hill

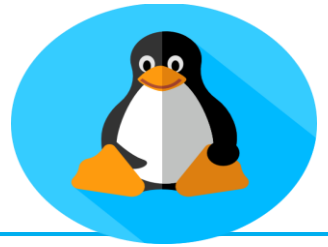
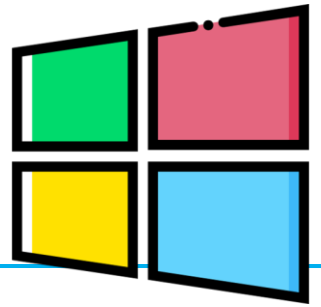
- **References:**
  - Modern operating Systems by Andrew Tanenbaum & Herbert Bos/ Pearson
  - Principles of Operating Systems by Naresh Chauhan / Oxford University Press
  - Beginning Linux Programming by Neil Matthew & Richard Stones / Wrox
  - Operating System : A Design-Oriented Approach by Charles Crowley / McGraw Hil

Introduction to OS

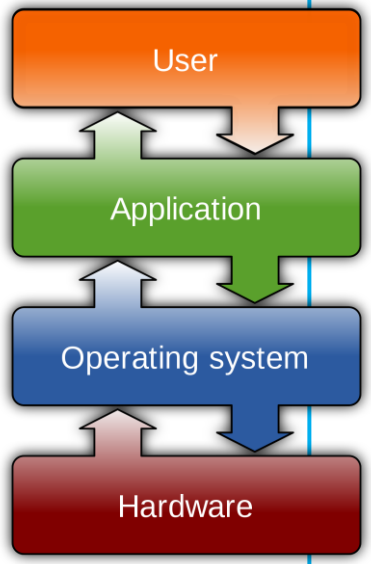
- What is OS
- How is it different from other application software;
- Why is it hardware dependent
- Different components of OS
- Basic computer organization required for OS
- Examples of well known OS including mobile OS, embedded system OS, Real Time OS, desktop OS server machine OS etc.
- How are these different from each other and why
- Functions of OS
- User and Kernel space and mode
- Interrupts and system calls

What is Operating System

- An operating system is software that manages a computer's hardware.
- It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware.

Different components



- A computer system can be divided roughly into four components: the hardware, the operating system, the application programs, and a user
- The hardware—the central processing unit (CPU), the memory, and the input/output (I/O) devices—provides the basic computing resources for the system.
- The application programs—a software or program that is designed to complete a specific task of users. Eg : word processors, spreadsheets, and web browsers
- The operating system- is a System software provides a platform for running and operating other software, namely application software. controls the hardware and coordinates its use among the various application programs for the various users

How is it different from other application software

Application software	Os
A computer program which is intended to perform some task classified along.	A system computer program that manages hardware and software resources and provides common services for computer program
It runs when the user desires to run the application.	it boots up when the user wants and run until the user switches off the machine.
It's examples are Photoshop, VLC player etc.	It's examples are Microsoft Windows, Linux, MAC
Users interact with application software while using specific applications.	Users never interact with system software as it functions in the background.
It is developed by using c++, c, assembly languages.	It is developed by using virtual basic, c++, c, java.

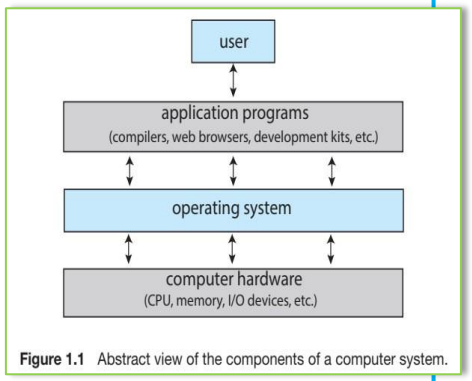


Figure 1.1 Abstract view of the components of a computer system.

Basic computer organization required for OS

RAM

Input devices

processor

Output devices

Hard disk

Computer organization is concerned with the way the hardware components operate and the way they are connected together to form the computer system.

The central processing unit (CPU) that processes data

The memory that holds the programs and data to be processed, and

I/O (input/output) devices as peripherals that communicate with the outside world.

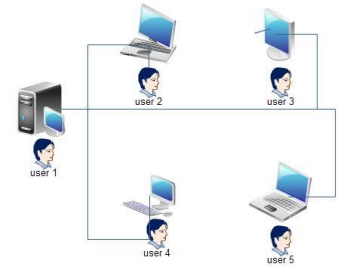
Types Of Os

- **Multitasking OS**

  Multi tasking refers to execution of multiple tasks.

- **Time-Sharing Operating Systems**

  Time-sharing operating system enables people located at a different terminal(shell) to use a single computer system at the same time.



- The processor time (CPU) which is shared among multiple users is termed as time sharing.

Mobile OS

- Mobile operating systems are those OS which is especially that are designed to power smartphones, tablets etc.

Types Of Os

- **Real Time Os**

A real-time operating system (RTOS) is an operating system with two key features: predictability and determinism. In an RTOS, repeated tasks are performed within a tight time boundary.Real-time systems are used when there are time requirements that are very strict like missile systems, air traffic control systems, robots, etc.

Air bag deployment depends on the rapid response time of a real-time embedded system with Hard RTOS

- **Hard RTOS**

These operating systems guarantee that critical tasks be completed within a range of time.

- **Soft RTOS**

This operating system provides some relaxation in the time limit. For example Multimedia systems, digital audio systems etc.

Different types of Os

EMBEDDED OS

An embedded system can be thought of as a computer hardware system having software embedded in it.

An embedded system can be an independent system or it can be a part of a large system.

An embedded system is a microcontroller or microprocessor based system which is designed to perform a specific task.

microwave ovens, washing machines and dishwashers,télévision sets, Vidéo Game consoles, set-top boxes, atms etc,Smart Tv

DESKTOP OS

The desktop OS is the environment where the user controls a personal computer .Ubuntu ,Windows 10

It is an operating system designed for usage on servers. It is utilized to give services to a large number of clients.

It is a very advanced operating system that can serve several clients simultaneously.

Important functions of an Operating System

- Process Management
- Memory Management
- File Management
- Device Management
- Protection and Security

Process Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time.

This function is called process scheduling.

An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.
- Providing mechanisms for process communication

Memory Management

- The operating system handles the responsibility of storing any data, system programs, and user programs in memory. This function of the operating system is called memory management.

Memory management activities

- Keeping track of which parts of memory are currently being used and by whom

  Deciding which processes to move into and out of memory

- Allocating and deallocating memory space as needed

File Management

- A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.
- An Operating System does the following activities for file management –
- Keeps track of information, location, uses, status etc.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Device Management

- An Operating System manages device communication via their respective drivers. It does the following activities for device management −
  - Keeps tracks of all devices. Program responsible for this task is known as the I/O controller.
  - Decides which process gets the device when and for how much time.
  - Allocates the device in the efficient way.
  - De-allocates devices.

Protection and Security

- The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.
- Monitors overall system health to help improve performance. records the response time between service requests and system response to having a complete view of the system health. This can help improve performance by providing important information needed to troubleshoot problems
- The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.

User space and Kernel space

When the computer system is run by user applications like creating a text document or using any application program, then the system is in user space

The system starts in kernel mode when it boots and after the operating system is loaded, it executes applications in user mode.

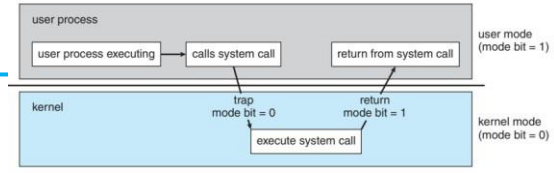


Figure 1.10 Transition from user to kernel mode.

Dual mode of operation

User mode

The operating system puts the CPU in user mode when a user program is in execution

When system runs a user application like creating a text document or using any application program, then the system is in user mode.

When CPU is in user mode, the programs don't have direct access to memory and hardware resources.

system will be in a safe state even if a program in user mode crashes.

Kernel mode

The operating system puts the CPU in kernel mode when it is executing in the kernel

At system boot time, the hardware starts in kernel mode.
on as well.
The operating system is then loaded and starts user applications in user mode.

The CPU can execute certain instruction only when it is in the kernel mode. **These instruction are called privilege instruction**
To provide protection to the hardware, we have privileged instructions which execute only in kernel mode

If a program crashes in kernel mode, the entire system will be halted.

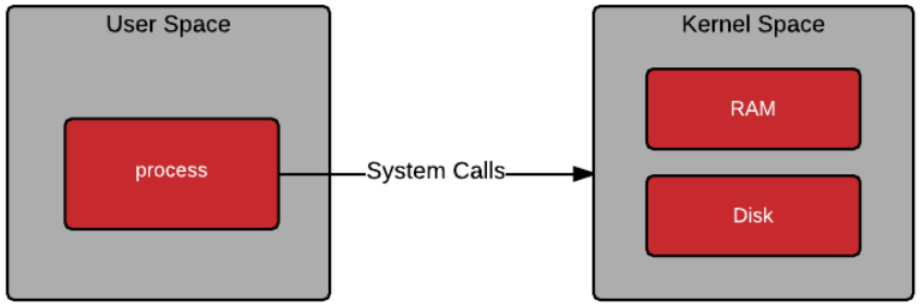
System Calls

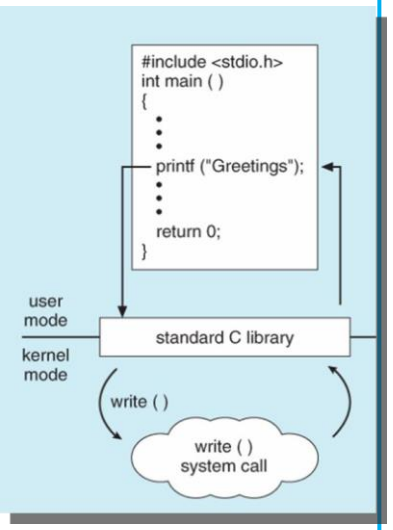
- A system call is a method for a computer program to request a service from the kernel of the operating system on which it is running.

  When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a system call.

- System calls are the only entry points into

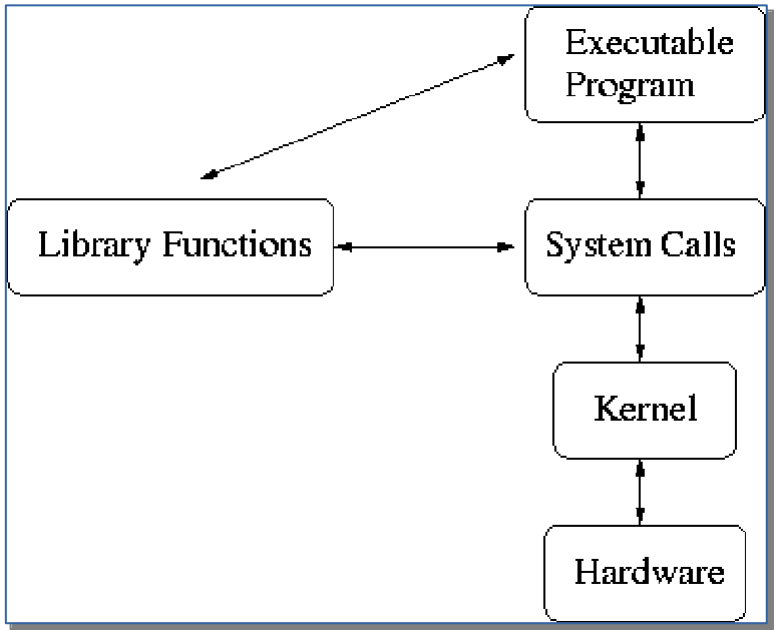
  the kernel system.





System Calls

- System calls are made by the user level programs in the following situations:
- Creating, opening, closing and deleting files in the file system.
- Creating and managing new processes.
- Creating a connection in the network, sending and receiving packets.
- Requesting access to a hardware device, like a mouse or a printer.



Introduction to Linux

Linux Topics

Introduction to Linux

Working basics of file system

Commands associated with files/directories & other basic commands.

Operators like redirection, pipe

What are file permissions and how to set them

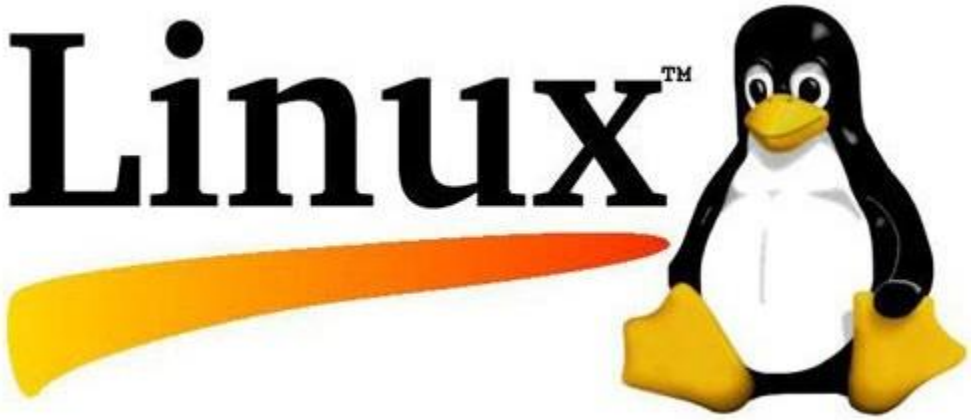
Permissions (chmod, chown, etc); access control list; network

commands (telenet, ftp, ssh, sftp, finger)

System variables like – PS1, PS2 etc. How to set them

LinuxOS

- Linux is the best-known and most-used open source operating system.
- Used in computers, servers, mobile devices and embedded devices etc



History

- **Linux Free and Open-Source Software**
- **Linux kernel** - Operating System Kernel released by **Linus Torvalds**
- **Linux** is a Unix-like operating system
- Published under **GNU General Public License**

Free and Open Source software
The users have the freedom to run, copy, distribute, study, change and improve the software

Tux - Symbol of Linux

Free and open-source software (FOSS)

FOSS :computer software that can be classified as both **free software** and **open-source software**

- **Free software**

  The freedom to run the software for any purpose Freedom to study

  Freedom to change

  Freedom to distribute the software

- **Open-source software**

  Computer softw are with its source code made available with a license .

  The copyright holder provides the rights to study, change, and distribute the

  software to anyone and for any purpose.

GNU General Public License

- The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU Project.
- FSF - non-profit organization founded by Richard Stallman to support the free software movement
- GNU's not Unix . Pronunciation ‘gnoo’
- GNU Project
- Linux Kernel Published under **GNU General Public License**

Linux Flavour/Distros

- Linux distributions are OSes based on the Linux Kernel

- Red Hat Enterprise Linux -Enterprise Operating System
  - – The Fedora Project -a community-supported free software project sponsered by Redhat
  - –The CentOS Project
- Debian
  - –Ubuntu
  - –Kali

Linux flavour/distros

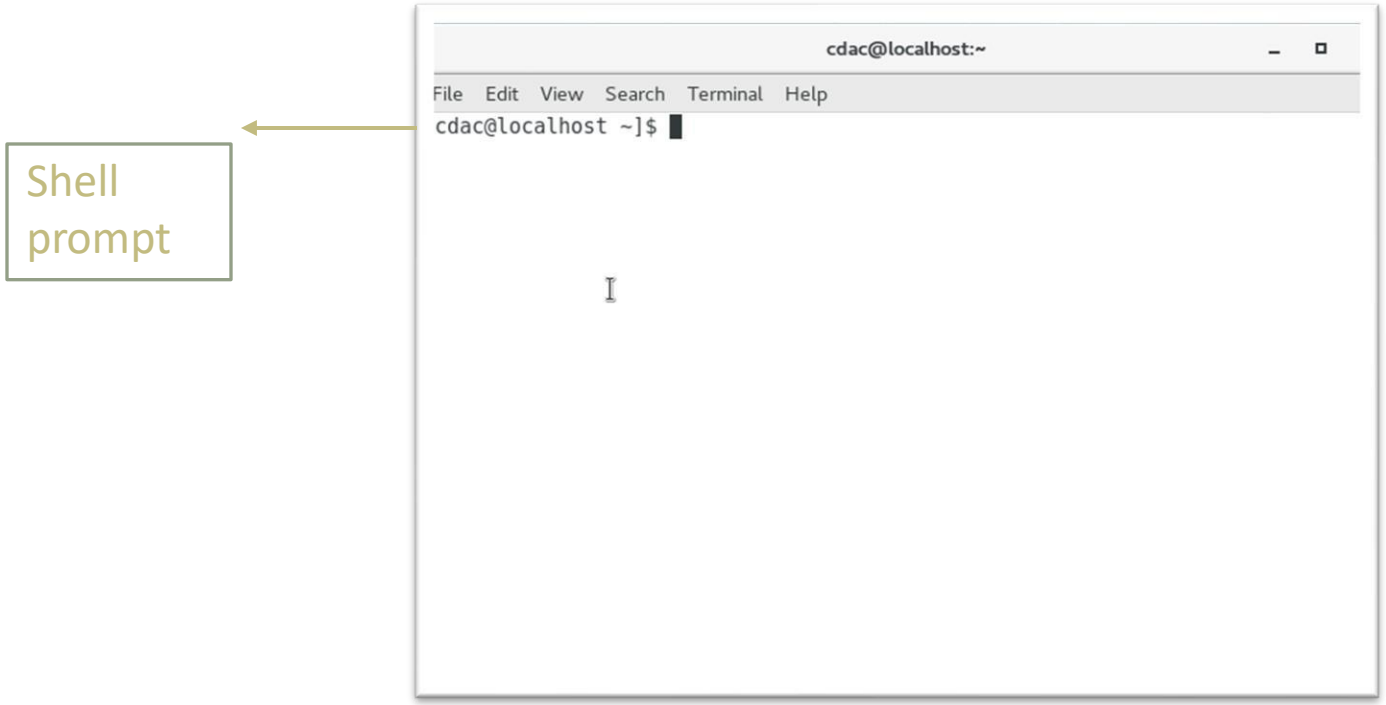
Kali Linux installation

- Installation using virtual box
- Memory/RAM 8 GB minimum, up to the system limit
- Hard Disk 4 GB minimum

Accessing Command line and Getting Help

The command line

- A text-based interface that allows you to enter commands, execute them, and view the results.
- The linux command line is provided by a program called the shell.

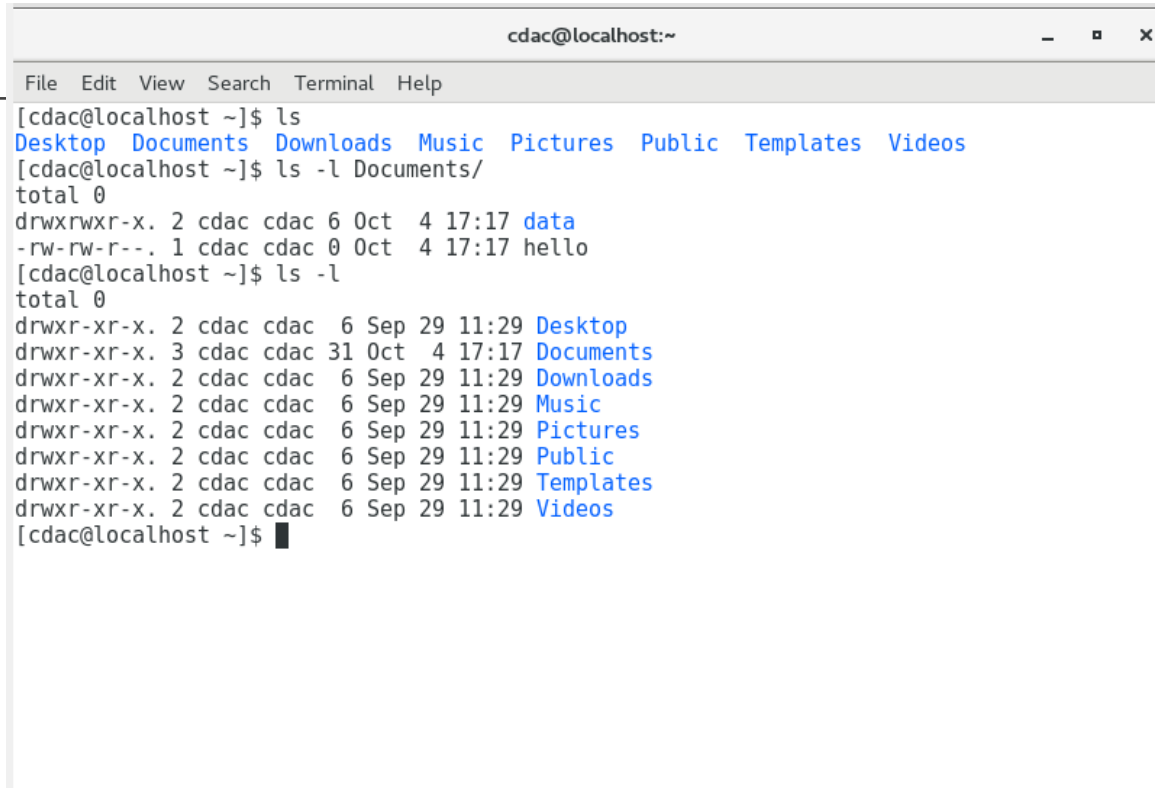


Running Commands

Commands syntax:

- ***command options arguments***

- Each item is separated by a space
- **Options** modify a command's behavior
  - –Single-letter options usually preceded by -
  - –Can be passed as -a -b -c or –abc
  - – Full-word options usually preceded by –
  - –Example: ls --help
- **Arguments** are filenames or other data needed by the command
- Multiple commands can be separated by ;



commands

Command	Meaning	Usage
ls	Displays a list of files in the current working directory	ls /home ls -l /home
cd directory	Change directories	cd home cd .. cd -
passwd	Change the password for the current user	passwd passwd bob
file filename	Display file type of file with name filename	file hello.txt
cat textfile	Throws content of textfile on the screen	cat hello.txt
pwd	Display present working directory	pwd
date	Display date and time	date
cal	Display calendar	cal
which	Command to locate executables in the system	which ls
exit or ctrl+D	terminates the current shell session	exit

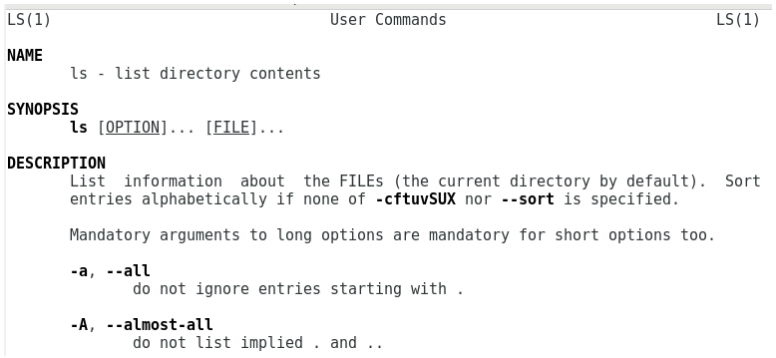
task	Keystroke
Locking the screen	Ct rl+Alt+L
logging out	select (User) > Log Out
To shut down the system Powering off or rebooting the system	Poweroff

Getting help

commands	description	usage
whatis keyword	Displays short descriptions of commands	whatis ls whatis head tail
man [<chapter>] <command>	Read built-in manual pages of a command	man 5 passwd man -a passwd man -f cat man -k printf man man
Command --help	short explanation about how to use the command and a list of available options.	ls --help

Reading manuals

•**Reading usage information**	**Navigating in man page**
•Arguments in [] are optional •Arguments in CAPS or <> are variables •eg:<filename> means "insert the filename you wish to use here" •Text followed by ... represents a list •x|y|z means "x or y or z" only one of them can be specified. • -abc means "any mix of -a, -b or -c"	•Navigate with arrows, PgUp, PgDn • /text searches for text • n/N goes to next/previous match •q quits



```
LS(1)                          User Commands                          LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).  Sort
       entries alphabetically if none of -cftuvSUX nor --sort is specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
```

BrowsingFiles in Filesystem

Linux File Hierarchy Concepts

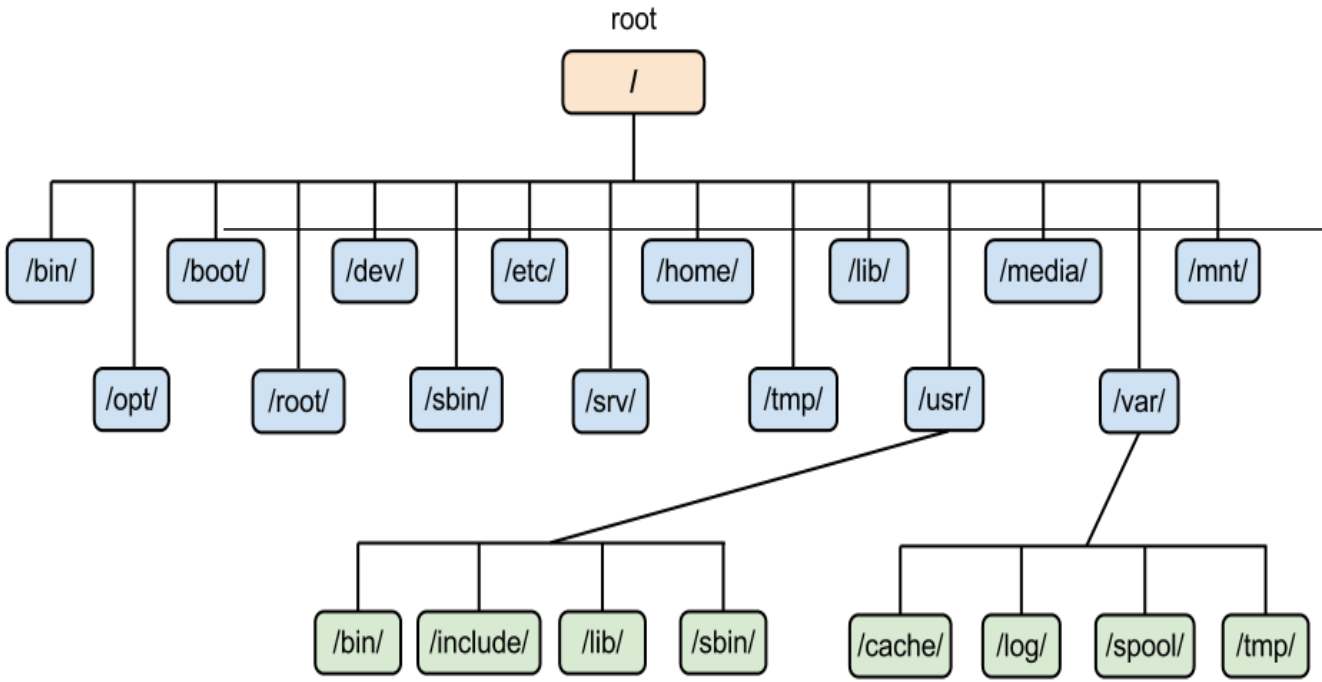
- Files and directories are organized into a single-rooted inverted tree structure
- This tree is inverted because the root of the tree is said to be at the top of the hierarchy,

  and the b ranches of directories and subdirectories stretch below the root.
- Filesystem begins at the root directory, represented by a / (forward slash) character.
- Names are case-sensitive
- Paths are delimited by /

- .. refers to parent directory of any particular directory
- . refers to current directory
- Files and directories starting with . are hidden.

e.g - filename = .abc.txt , .file1

Linux File System Layout



Home Directories:	/root,/home/*username*
User Executables	/bin, /usr/bin, /usr/local/bin
System Executables	/sbin, /usr/sbin, /usr/local/sbin
Other Mountpoints	/media, /mnt
Configuration:	/etc
Temporary Files	/tmp
Kernels and Bootloader:	/boot
Server Data Files that dynamically change (e.g. databases, cache directories, log files, pri nterspooled docume nts, and webs ite content)	/var, /srv
System Information:	/proc, /sys
Shared Libraries	/lib, /usr/lib, /usr/local/lib
Device files	/dev

man page hier(7)

Browsing files in filesystem

command	Description	Usage
cd directory	Change directory	To an absolute or relative path: • **cd** /home/cdac/Desktop • **cd** project/docs • cd .. • cd – • cd • cd ~
mkdir [option] dir_name	Create directory	mkdir folder1 folder2 mkdir -p food/fruit/citrus/oranges
rmdir [options] dir_name	Remove empty directory	rmdir folder1
rm –r dir_name	Remove non empty directory	
rm [option] file[s]	Remove files	rm file1 file2 rm *
touch file[s]	create empty files or update file timestamps	touch hello.txt

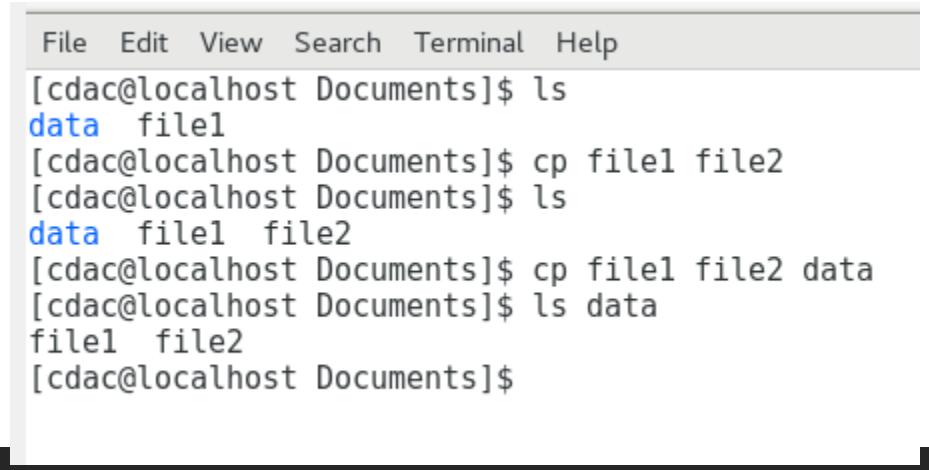
absolute path name
A path name with a forward slash (/) as the first character

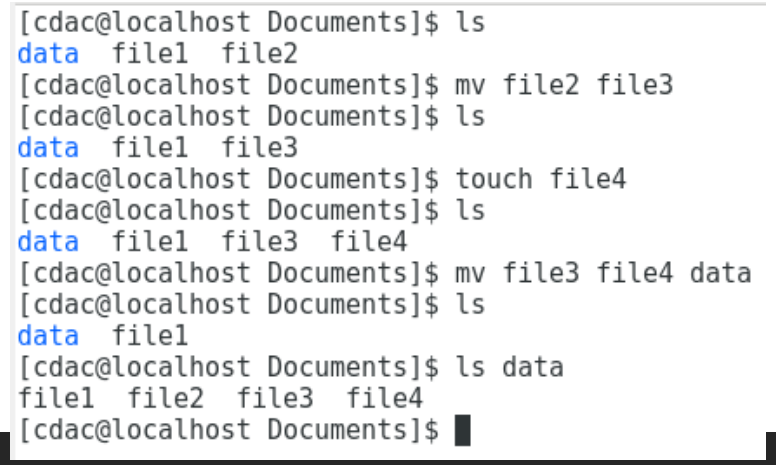
Relative Path specifying only the path necessary to reach the file from the working directory

```
File  Edit  View  Search  Terminal  Help
[cdac@localhost ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[cdac@localhost ~]$ cd Documents/data/
[cdac@localhost data]$ pwd
/home/cdac/Documents/data
[cdac@localhost data]$ cd /home/cdac
[cdac@localhost ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[cdac@localhost ~]$ cd /
[cdac@localhost /]$ ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
[cdac@localhost /]$
```

Cut,copy and paste..

Command	Description	Usage
ls [options] [files_or_dirs]	Lists the contents of the current directory or a specified directory	ls -a (include hidden files) ls -l (display extra information) ls -R (recurse through directories
cp [options] source destination	copy files and directories	cp –f hello.txt hello1.txt create copy in same directory with different name
cp [options] file1 file2 dest	More than one file may be copied if dest is a directory	cp –f hello.txt hello1.txt fold
mv [options] file destination	move and/or rename files and directories	mv old.txt new.txt -Rename mv [options] file1 file2 destination





Gathering Disk Space

- **df** command reports the system's disk space usage.
- **du** command displays the estimated amount of space being used by files in a directory, displaying t he disk usage of each subdirectory.

```
File  Edit  View  Search  Terminal  Help
[cdac@localhost ~]$ du -h
0       ./.mozilla/extensions/{ec8030f7-c20a-464f-9b0e-13a3a9e97384}
0       ./.mozilla/extensions
0       ./.mozilla/plugins
0       ./.mozilla/firefox/jzqm9xfn.default/gmp/Linux_x86_64-gcc3
0       ./.mozilla/firefox/jzqm9xfn.default/gmp
4.0K    ./.mozilla/firefox/jzqm9xfn.default/bookmarkbackups
0       ./.mozilla/firefox/jzqm9xfn.default/storage/permanent/chrome/idb/2918
sah.files
48K     ./.mozilla/firefox/jzqm9xfn.default/storage/permanent/chrome/idb
56K     ./.mozilla/firefox/jzqm9xfn.default/storage/permanent/chrome
56K     ./.mozilla/firefox/jzqm9xfn.default/storage/permanent
56K     ./.mozilla/firefox/jzqm9xfn.default/storage
0       ./.mozilla/firefox/jzqm9xfn.default/sessionstore-backups
8.0K    ./.mozilla/firefox/jzqm9xfn.default/datareporting/archived/2020-09
8.0K    ./.mozilla/firefox/jzqm9xfn.default/datareporting/archived/2020-10
16K     ./.mozilla/firefox/jzqm9xfn.default/datareporting/archived
24K     ./.mozilla/firefox/jzqm9xfn.default/datareporting
1.4M    ./.mozilla/firefox/jzqm9xfn.default/gmp-gmpopenh264/1.6
1.4M    ./.mozilla/firefox/jzqm9xfn.default/gmp-gmpopenh264
12K     ./.mozilla/firefox/jzqm9xfn.default/saved-telemetry-pings
19M     ./.mozilla/firefox/jzqm9xfn.default
19M     ./.mozilla/firefox
19M     ./.mozilla
```

```
File  Edit  View  Search  Terminal  Help
[cdac@localhost ~]$ df
Filesysten           1K-blocks    Used Available Use% Mounted on
/dev/mapper/rhel-root  17811456 3876636  13934820  22% /
devtmpfs                2435492       0   2435492   0% /dev
tmpfs                   2451404       0   2451404   0% /dev/shm
tmpfs                   2451404    9816   2441588   1% /run
tmpfs                   2451404       0   2451404   0% /sys/fs/cgroup
/dev/sda1               1038336  182348    855988  18% /boot
tmpfs                    490284       4    490280   1% /run/user/42
tmpfs                    490284      28    490256   1% /run/user/1000
[cdac@localhost ~]$ df -h
Filesysten           Size  Used Avail Use% Mounted on
/dev/mapper/rhel-root   17G  3.7G   14G  22% /
devtmpfs               2.4G     0  2.4G   0% /dev
tmpfs                  2.4G     0  2.4G   0% /dev/shm
tmpfs                  2.4G  8.9M  2.4G   1% /run
tmpfs                  2.4G     0  2.4G   0% /sys/fs/cgroup
/dev/sda1             1014M  179M  836M  18% /boot
tmpfs                  479M  4.0K  479M   1% /run/user/42
tmpfs                  479M   28K  479M   1% /run/user/1000
[cdac@localhost ~]$
```

Managing users

Command	Description	
adduser, userdel	Standard utilities for adding, modifying, and deleting user accounts.	
userdel -r	Delete existing user along with home directory	userdel –r bob
groupadd, groupmod, groupdel	Standard utilities for adding, modifying, and deleting groups.	

changing identities

su -	creates new shell as root	
sudo	command runs command as root	Sudo useradd bob
id,	displays the identity of the user running the session along with the list of groups they belong to.	
whoami ,w.	shows information on the current user	

Changing file permissions

- Symbolic & numeric method
- chmod [-R] mode file
  - –chmod 640 myfile
  - –chmod ugo+r myfile

Code	Meaning
u	user permissions
g	group permissions
o	permissions for others

Access modeCodes	Meaning
0 or -	The access right that is supposed to be on this place is not granted.
4 or r	read access is granted to the user category defined , to read a file or list a directory's contents
2 or w	write permission is granted to the user category , to write to a file or create and remove files from a directory
1 or x	execute permission is granted to the user category , to execute a program or change into a directory and do a long listing of the directory

```
[cdac@localhost Documents]$ ls -l
total 8
drwxrwxr-x. 2 cdac cdac  45 Oct  5 12:25 data
-rw-rw-r--. 1 cdac cdac   0 Oct  6 00:41 myscript.sh
-rwxrw-r--. 1 cdac cdac 244 Oct  6 00:10 usercreate.sh
-rwxrw-r--. 1 cdac cdac 113 Oct  6 00:15 userremove.sh
[cdac@localhost Documents]$ chmod u+x myscript.sh
[cdac@localhost Documents]$ ls -l
total 8
drwxrwxr-x. 2 cdac cdac  45 Oct  5 12:25 data
-rwxrw-r--. 1 cdac cdac   0 Oct  6 00:41 myscript.sh
-rwxrw-r--. 1 cdac cdac 244 Oct  6 00:10 usercreate.sh
-rwxrw-r--. 1 cdac cdac 113 Oct  6 00:15 userremove.sh
[cdac@localhost Documents]$
```

command to execute permission to user = chmod u+x
group=chmod g+x
other=chmod o+x

Input output redirection

Redirections	Operator	Description	Usage
Std output redirection	>	Redirects output of a command to file	cat file1 >file2
Pipe	|	Redirects output of one command as input to other	ls |grep hello.txt echo "test email" | mail -s "test" user@example.com
	>>	Redirect output and append it to file	cat file1>> file2
Input redirection	<	Redirects input	tr 'a-z' 'a-z' <hello.txt
Std error redirection	2>	Redirects error to file	ls 2> tmp ls > dirlist 2>&1
Tee command		Writing to output and files simultaneously	date | tee file1 file2
Sending Multiple Lines to STDIN	<<word	Redirect multiple lines from keyboard to STDIN with <<WORD	

Text processing tools

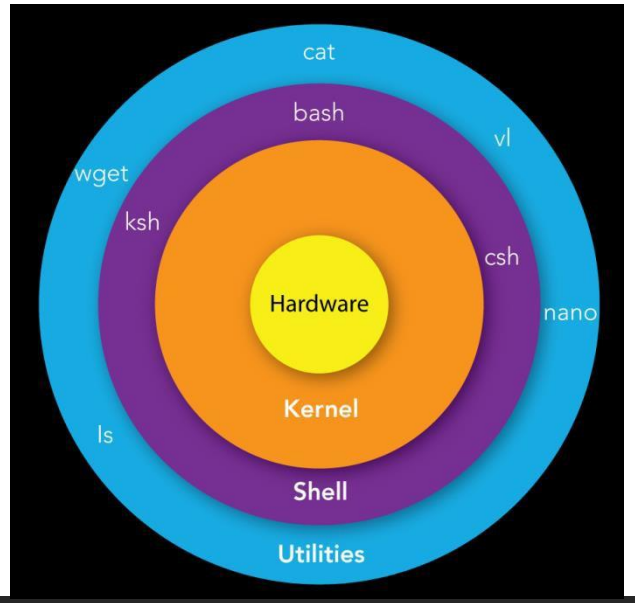
cat filenames	It reads data from files, and outputs their contents	cat hello.txt cat file1 file2
less filename	View file one page at a time	less hello.txt
head [option] filename	Display the first 10 lines of a file	head hello.txt head –n hello.txt
tail [option] filename	Display the last 10 lines of a file	tail hello.txt tail –f hello.txt
grep [options] PATTERN [FILE...]	print lines matching a pattern	grep 'john' /etc/passwd date --help | grep year grep –v 'bash' /etc/passwd -Ax -Bx
cut	Display specific columns of file	cut -d: -f1 /etc/passwd grep root /etc/passwd | cut -d: -f7 Use -f to specify the column to print Use -c to cut by characters

Shell.

•Shell :It is a command language interpreter
•Backward compatible with sh(standard unix shell).

•To find all available shells in your system type following command:
•$ cat /etc/shells

•echo $SHELL : Show which shell is used.



Software installation

Installing new software

- software comes in packages
- Extra software may be found on your installation CDs or on the Internet.
- Package formats
  - RPM packages
  - DEB (.deb) packages
- RPM
  - The redhat package manager, is a powerful package manager that can be used to install, update and remove packages
- DEB (.deb) packages
  - package format is the default on Debian GNU/Linux

Installing package using RPM

Rpm format <name>-<version>- <release>.<architecture>.rpm	To install a rpm package -v verbose message -h makes rpm print a progress bar	rpm -ivh <packagename>
	To query package, whether it is installed or not	rpm -q <packagename> rpm -qa | grep vim
	To uninstall a package	rpm -e <packagename>

Automating package management and updates

Purpose	**Debian/ubuntu**	**Redhat/centos/fedora**
Tool	Apt-get Advanced Package Tool	Yum Yellowdog Updater, Modified
Upgrade Installed Packages	apt-get upgrade	yum update
Find a Package	apt-cache search search_string	yum search search_string
Install a Package from Repositories	apt-get install package	yum install package
Install a Packagefrom the Local Filesystem	sudo dpkg -i package.deb	yum install package.rpm
Remove Installed Package	apt-get remove package	yum remove package

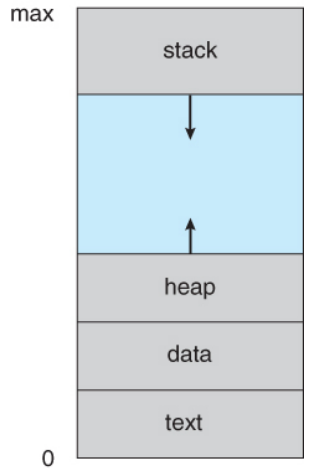
Operating System

Process Management

Process management

- A process is a program in execution.
- A process is an ‘active’ entity, as opposed to a program, which is considered to be a ‘passive’ entity.
- A single program can create many processes when run multiple times

How does a process look like in memory?



- When a program gets loaded into the memory, it is said to as a process.
- process memory is divided into four sections – stack, heap, text and data.

Stack

- The process Stack contains the temporary data such as method/function parameters, return address and local variables.

Heap

- This contains dynamically allocated memory to a process during its run time.

```
ptr =(int*)malloc(100*sizeof(int));
```

Text

- Code segment, also known as text segment contains The text section comprises the compiled program code

Data

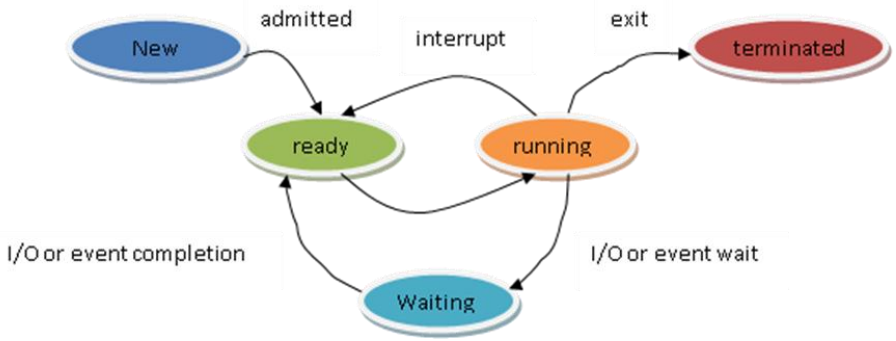
- This section contains the global variable

```
1 #include<stdio.h>
2 int a=8;
3 main()
4 {
5
6   int b=2;
7   printf("sum is %d ",b+a);
8
9
10 }
```

Process Life Cycle/state

When a process executes, it passes through different states.

States	Description
• **New:**	In this state, the process is being created. it is the program which is present in secondary memory that will be picked up by OS to create the process.
• **Ready:**	Process is Ready to run and is waiting for CPU to execute it. Processes that are ready for execution by the CPU are maintained in a queue for ready processes
• **Running:**	The instructions within the process are executed by CPU
• **Waiting:**	Whenever the process requests access to I/O or needs input from the user,the process enters a waiting state. The process continues to wait in the main memory and does not require CPU. Once the I/O operation is completed the process goes to the ready state
• **Terminated:**	In this state, the process has finished executing.



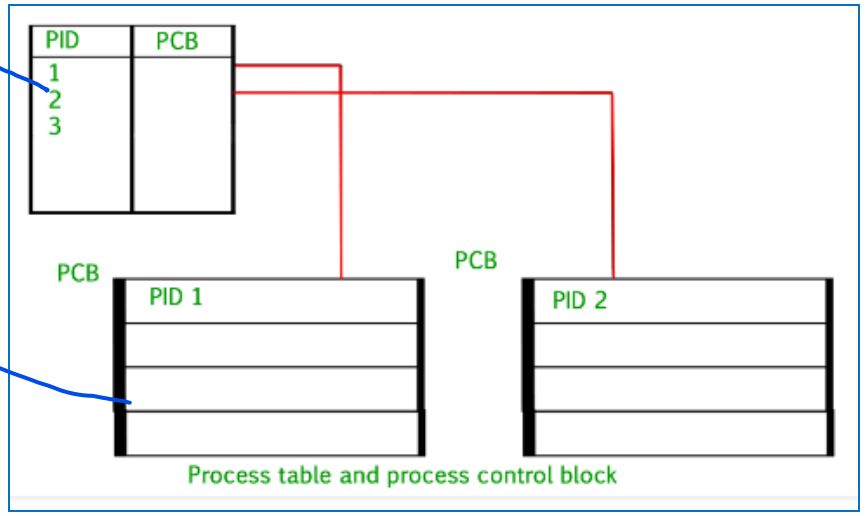
System Process

```
$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        Ss     0:03 init [5]
    2 ?        S      0:00 [migration/0]
    3 ?        SN     0:00 [ksoftirqd/0]
    4 ?        S<     0:05 [events/0]
    5 ?        S<     0:00 [khelper]
    6 ?        S<     0:00 [kthread]
  840 ?        S<     2:52 [kjournald]
  888 ?        S<s    0:03 /sbin/udevd --daemon
 3069 ?        Ss     0:00 /sbin/acpid
 3098 ?        Ss     0:11 /usr/sbin/hald --daemon=yes
```

- Each process is started by another process known as its parent process.
- A process so started is known as a child process.
- When Linux starts, it runs a single program, the prime ancestor and process number 1, init.

Process Table and Process Control Block (PCB)

- The **Process table** is like a data structure describing all of the processes that are currently loaded with.
- A **Process Control Block** is a data structure maintained by the Operating System to keep track of every process.
- The PCB is identified by an integer process ID (PID)
- Also called Task Controlling ,Task Struct,
- The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.



Process representation

- The kernel stores the list of processes in a circular doubly linked list called the task list.
- Within the Linux kernel, a process is represented by a structure called task_struct

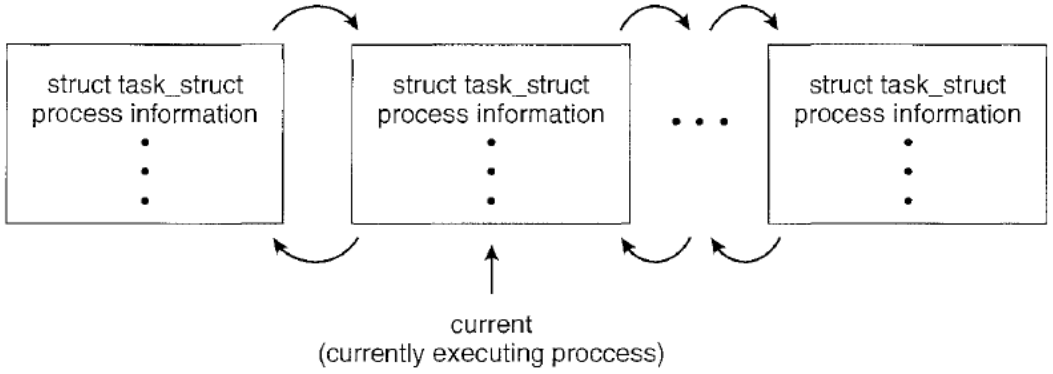


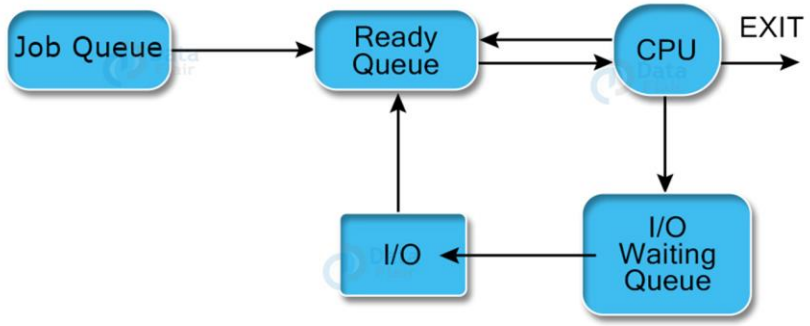
Figure 3.5 Active processes in Linux.

Process scheduling

- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process
- **Why do we need scheduling?**
  - A typical process involves both I/O time and CPU time.
  - In a uniprogramming system , time spent waiting for I/O is wasted and CPU is free during this time.
  - In multiprogramming systems, one process can use CPU while another is waiting for I/O.
  - This is possible only with process scheduling

Process Scheduling Queues

- The OS maintains all PCBs in **Process Scheduling Queues.**
- The OS maintains a separate queue for each of the process states.
- PCBs of all processes in the same execution state are placed in the same queue.
- When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.



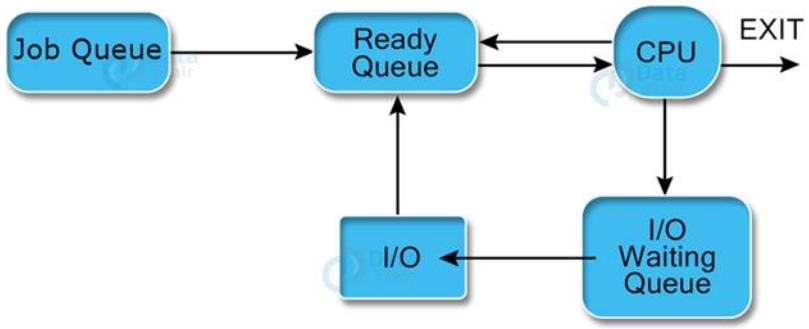
•**Job queue** - This queue keeps all the processes in the system. In starting, all the processes get stored in the job queue. It is maintained in the secondary memory.

•**Ready queue** –
This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue. This queue is generally stored as a linked list

•**Device queue**.- Processes waiting for a device to become available are placed in Device Queues. There are unique device queues available for each I/O device.

Process schedulers

- Schedulers selects the jobs to be submitted into the system and decide which process to run.
- Schedulers are of three types
  - **Long-Term Scheduler**
  - **Short-Term Scheduler**
  - **Medium-Term Scheduler**



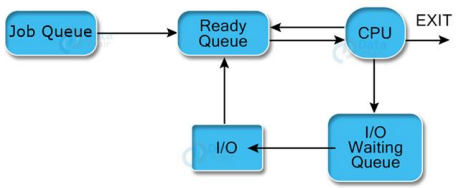
Process schedulers

Short-term scheduler (CPU scheduler)

- selects which process should be executed next and allocates CPU.
- Allocates Cpu to a process in memory
- Short-term scheduler is invoked frequently

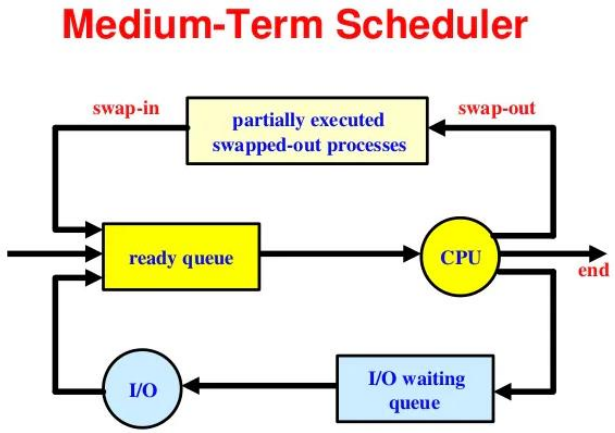
Long-term scheduler (Job scheduler)

- selects which processes should be brought into the ready queue
- Select a process from sec.memory to ready queue in memory
- Long-term scheduler is invoked infrequently (seconds, minutes)
- The long-term scheduler controls the degree of multiprogramming



Medium-term scheduler

- Medium-term scheduler can be added if degree of multiple programming needs to decrease
- Remove process from memory, store on disk, bring back in from disk to continue execution: swapping



I/O-bound process and CPU bound process

- Processes can be described as either:

I/O-bound process –

- If a process doesn't carry out many calculations, but it does do a lot of input/output operations, it's called an I/O-bound process.
- E g: office applications are mostly I/O bound

CPU-bound process –

- If a process does a lot operations using CPU, it's called a CPU-bound process
- E g: image resizing is a kind of CPU-bound task

Dispatcher

- **Dispatcher** is responsible for loading the process selected by Short-term scheduler on the CPU (Ready to Running State) Context switching is done by dispatcher only.

- A dispatcher does following:

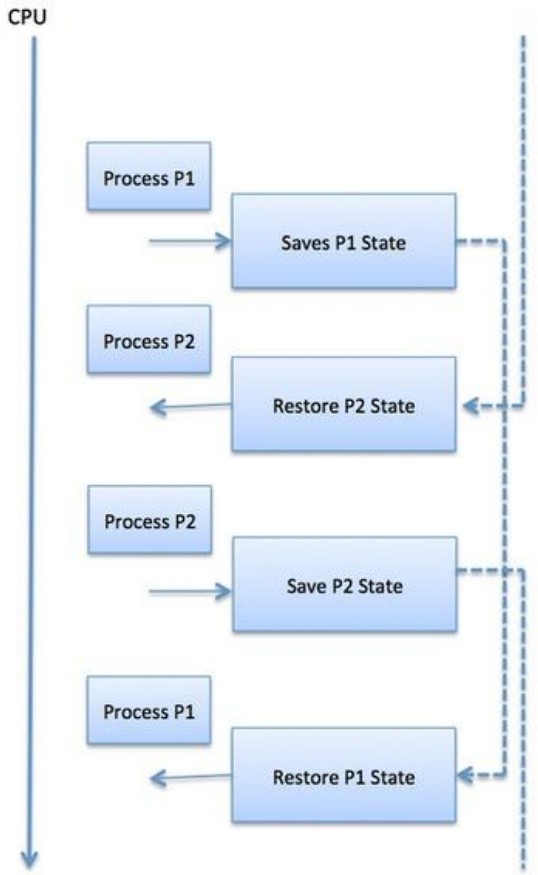
  1) Switching context.

  2) Switching to user mode.

  3) Jumping to the proper location in the newly loaded program.

Context switch

- A context switch is the mechanism to **store and restore the state or context of a CPU in process control block** so that a process execution can be resumed from the same point at a later time
- When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



Non-preemptive or preemptive

Preemptive scheduling

- The scheduling in which a running process can be interrupted if a high priority process enters the queue and is allocated to the CPU is called preemptive scheduling.

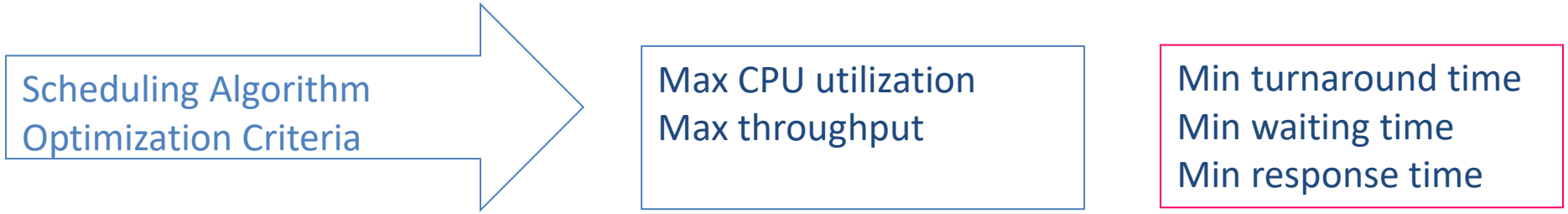
non-preemptive scheduling

- The scheduling in which a running process cannot be interrupted by any other process is called non-preemptive scheduling. Any other process which enters the queue has to wait until the current process finishes its CPU cycle.

Process scheduling

- A Process Scheduler schedules different processes to be assigned to the CPU based on particular **scheduling algorithms**
  - First-Come, First-Served (FCFS) Scheduling
  - Shortest-Job-Next (SJN) Scheduling
  - Priority Scheduling
  - Round Robin(RR) Scheduling
  - Multiple-Level Queues Scheduling
- **These algorithms are either non-preemptive or preemptive**

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit. For long processes, this rate may be one process per hour; for short transactions, it may be ten processes per second
- **Turnaround time** – amount of time to execute a particular process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/0.
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)



First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high
- Convoy effect - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

process arrived at t=0

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Non Premptive ,arrival time

PNo	AT	BT	CT ?	TAT ?	WT ?
1	0	4			
2	1	3			
3	2	1			
4	3	2			
5	4	5			

Completion Time:

At what time the process is going to get completed

Turn Around Time:

The time interval between the ct and at. CT-AT

The interval from the time of submission of a process to the time of completion

Waiting Time

Amount of time a process has been waiting in the ready queue

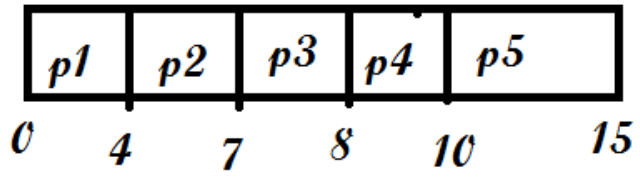
TAT=WT+BT So WT=TAT-BT

TAT=CT-AT

WT = TAT - BT

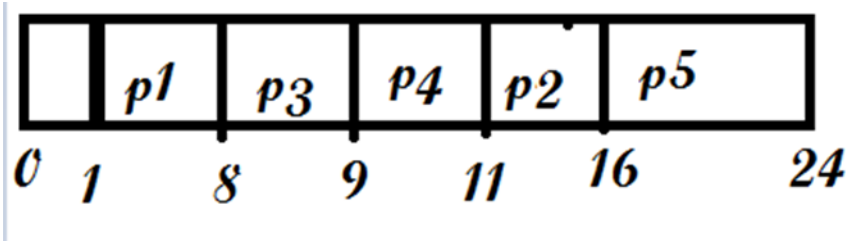
PNO	AT	BT	CT ?	TAT ?	WT ?
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

PNo	AT	BT	CT ?	TAT ?	WT ?
1	0	4			
2	1	3			
3	2	1			
4	3	2			
5	4	5			



Shortest-Job-Next (SJN) Scheduling

- Also called SJF
- From the process arrived it chooses the shortest job.
- Both premptive and non premptive



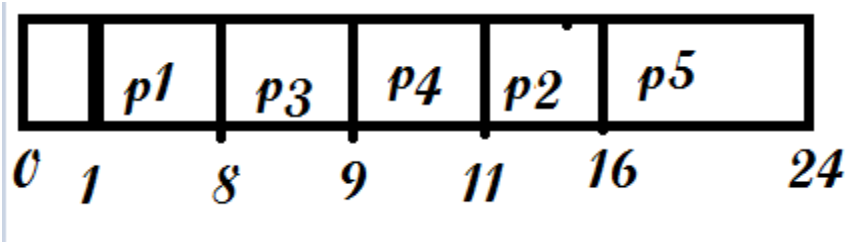
Non preemptibve

PNO	AT	BT	CT ?	TAT ?	WT ?
1	1	7			
2	2	5			
3	3	1			
4	4	2			
5	5	8			

TAT=CT-AT

WT = TAT - BT

PNO	AT	BT	CT ?	TAT ?	WT ?
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11



TAT=CT-AT

WT = TAT - BT

Priority scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

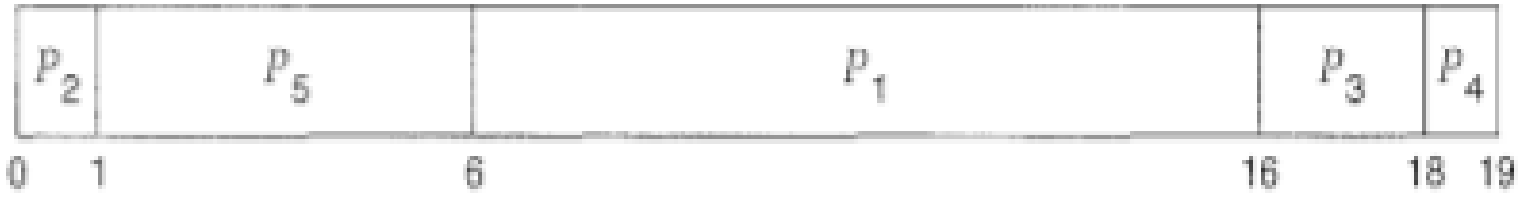
- Priority scheduling is a non-preemptive algorithm
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement

- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart:



Round Robin

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes

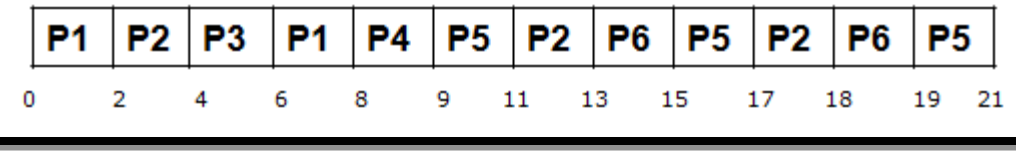
Time quantum :2

PROCESS	AT	BT
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

Example of RR with Time Quantum = 2

Time quantum :2		
PROCESS	**AT**	**BT**
P1	0	4
P2	1	5
P3	2	2
P4	3	1
P5	4	6
P6	6	3

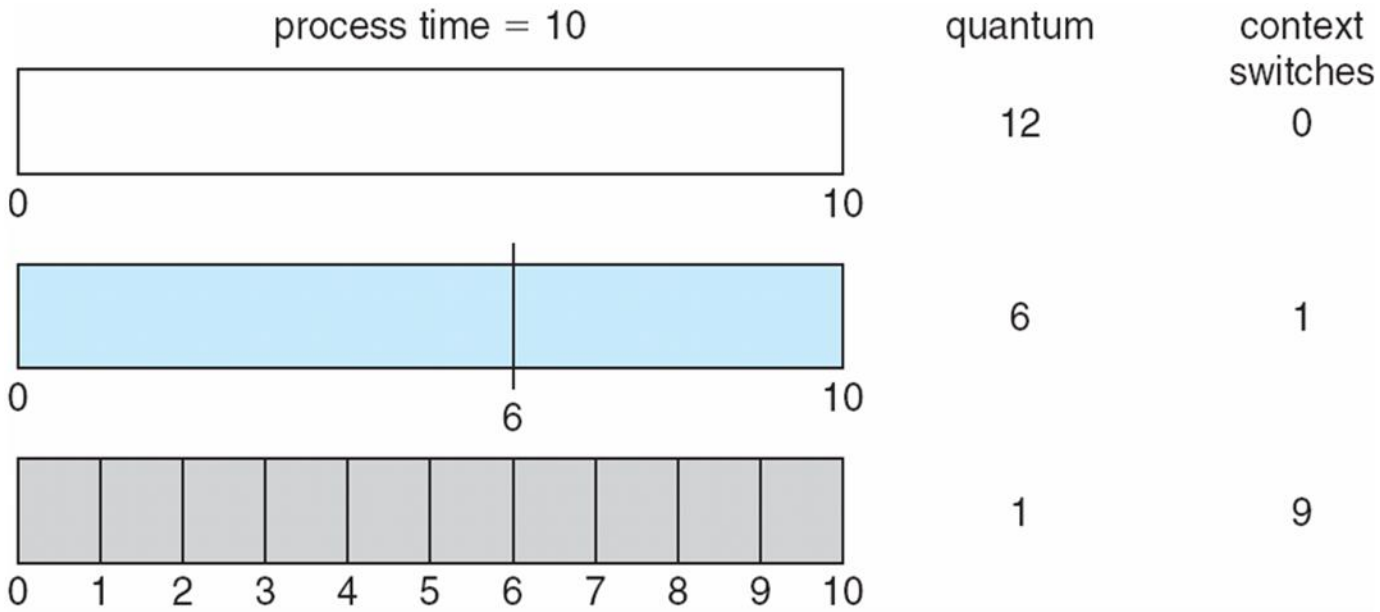
GANCHART



P1-P2-P3-P1-P4-P5-P2-P6-P5-P2-P6-P5
READY QUEUE

Ready queue		
Process	AT	BT
p1	0	4
P2	1	5
P3	2	2
p1		2
p4	3	1
p5	4	6
p2		3
p6	6	3
p5		4
p2		1
p6		1
p5		2

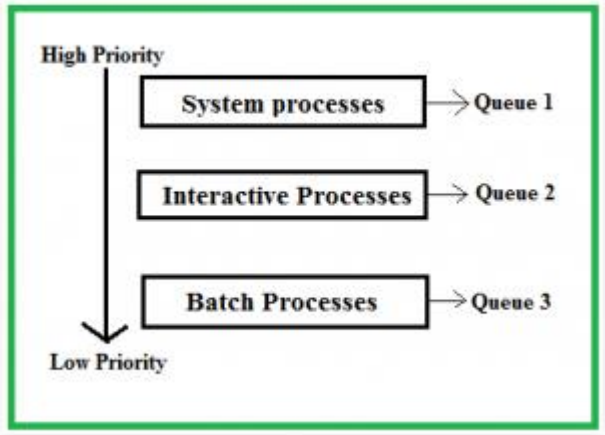
Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum

Multiple-Level Queues Scheduling

- Ready queue is partitioned into separate queues , eg:
  - foreground (interactive)
  - background (batch)
- Priorities are assigned to each queue.
- Each queue can have its own scheduling algorithms.
- Scheduling must be done between the queues:
  - serve all from system processes,foreground then from background.
  - Possibility of starvation.



Operations on Processes

System must provide mechanisms for:

- process creation
- process termination

Process Creation

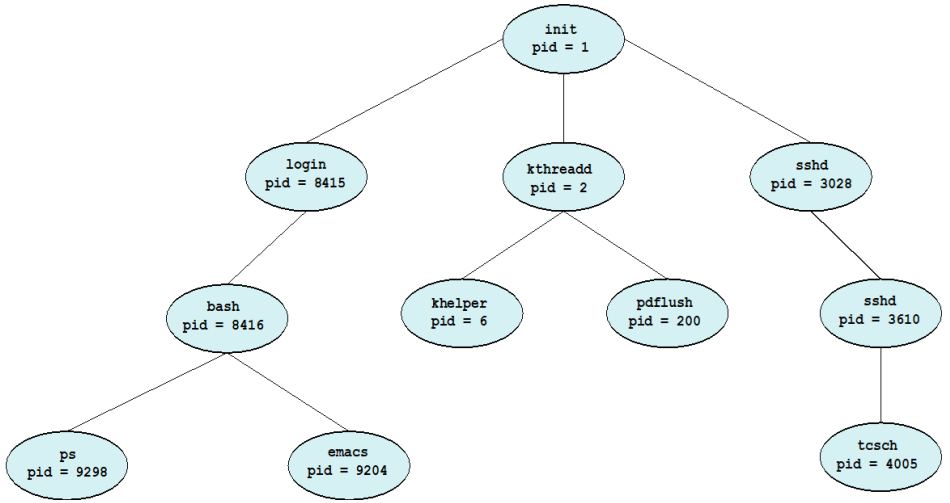
- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)

Resource sharing options

- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources

Execution options

- Parent and children execute concurrently
- Parent waits until children terminate



Commonly a process required resources like CPU time, memory, I/O devices and files to accomplish the task

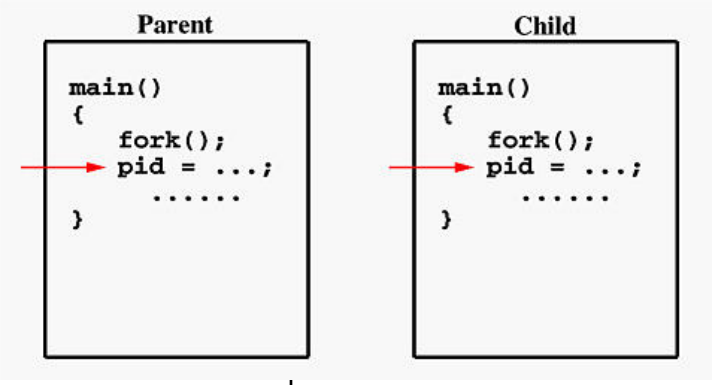
Process creation and termination system call

Process creation :fork,exec

Process wait: wait

Process termination: exit

Process creation



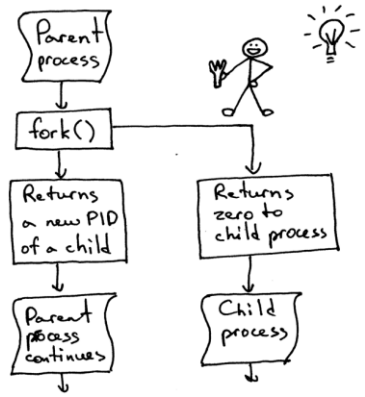
Process creation

fork()

- creates a new process by duplicating the calling process .
- The new process - child, is an exact duplicate of the calling process,
- Calling process- parent,

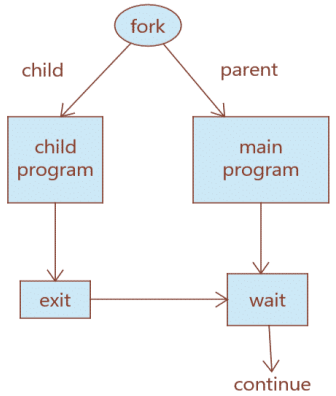
Return Value

- Negative Value: creation of a child process was unsuccessful.
- Zero: Returned to the newly created child process.
- Positive value: Returned to parent or caller.(returns the process identifier (pid) of the child process in the parent,)



Process creation

exec()	**exec** #include <unistd.h> • replaces the current running process with a new process. • An exec function replaces the current process with a new process specified by the pathor fileargument.
	•int execlp(const char *file, const char *arg, ..., NULL); •process which is a copy of the one that calls it, while exec replaces the current process image with another (different) one.

Process termination exit()	void exit(int status) Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
Wait()  	#include <sys/wait.h> pid_t wait (int *status_ptr) or pid = wait(&status); • The parent process wait for termination of a child process • The call returns status information and the pid of the terminated process –This status value is zero if the child process explicitly returns zero status. wait(&status) –If it is not zero, it can be analyzed with the status analysis macros. WEXITSTATUS(stat) –The status_ptr pointer may also be NULL, where, wait(NULL) ignores the child's return status. • If no parent waiting (did not invoke **wait()**) process is a **zombie** • If parent terminated without invoking **wait** , process is an **orphan** • A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

Zombie and orphan process

Zombie

A zombie process is a process whose execution is completed but it still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status. Once this is done using the wait system call, the zombie process is eliminated from the process table.

Orphan

If parent terminated without invoking **wait** , process is an **orphan**

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

End

Operating System: Signals &Threads

Threads

signal

- Signals are standardized messages sent to a running program to trigger specific behavior, such as quitting or error handling
- When a signal is sent, the operating system interrupts the target process' normal flow of execution to deliver the signal. If the process has previously registered a signal handler, that routine is executed. Otherwise, the default signal handler is executed. [1]

Signal Names and Values

Signal Name	Signal Number	Description
SIGHUP	1	Hang up detected on controlling terminal or death of controlling process
SIGINT	2	Issued if the user sends an interrupt signal (Ctrl + C)
SIGQUIT	3	Issued if the user sends a quit signal (Ctrl + D)
SIGFPE	8	Issued if an illegal mathematical operation is attempted
SIGKILL	9	If a process gets this signal it must quit immediately and will not perform any clean-up operations
SIGALRM	14	Alarm clock signal (used for timers)
SIGTERM	15	Software termination signal (sent by kill by default)

- Kill -l

Sending Signals

When you press the Ctrl+C key, a SIGINT is sent to the process and as per defined default action process terminates.

- use the kill command
- kill -signal pid
- int kill(pid_t pid, int signum);

Handling Signals

- **int signal(int signum, void (*handler)(int))**
- //(SIGINT, SIG_DFL); //Setting SIGINT to default behavoiur
- the first argument is the signal number, such as SIGSTOP or SIGINT,
- second is a reference to a handler function whose first argument is an int and returns void.
- Default Actions of Signals
  - Each signal has a **default action**.
  - The default action for a signal is the action that a process performs when it receives a signal.
  - **Term :** The process will terminate
  - **Ign :** The process will ignore the signal
  - **Stop :** The process will stop, like with a Ctrl-Z

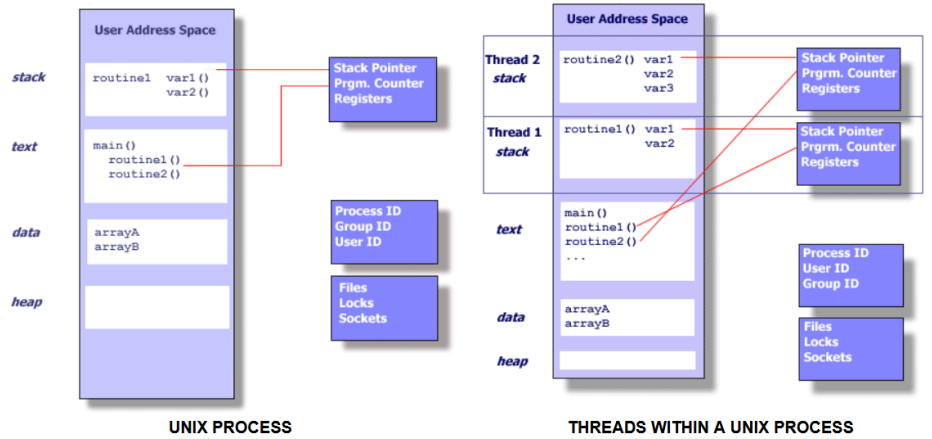
Threads

- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
- A thread is also known as lightweight process.
- The idea is to achieve parallelism by dividing a process into multiple threads.
- For example, in a browser, multiple tabs can be different threads.
- MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Process vs Thread?

- The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.

- Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.



Advantages of Thread over Process

1. **Responsiveness:** If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. **Faster context switch:** Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3. **Effective utilization of multiprocessor system:** If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. **Resource sharing:** Resources like code, data, and files can be shared among all threads within a process.

Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

Types of threads

- **User levelthread**
  - management done by user-level threads library
  - Three primary thread libraries:
  - POSIX Pthreads posix-portable operating system interface
  - Windows threads
  - Java threads
- **Kernel level thread**
  - kernel threads are implemented by OS.

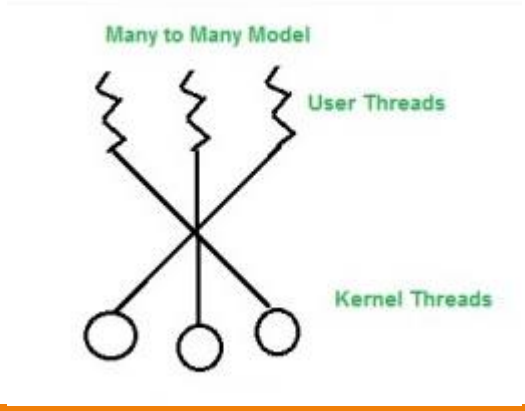
Multi threading models

Many operating systems support kernel thread and user thread in a combined way

- Many to many model.
- Many to one model.
- one to one model

Many to many

- In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads.
- Number of kernel level threads are specific to the machine.
- advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread.
- Thus, System doesn’t block if a particular thread is blocked.

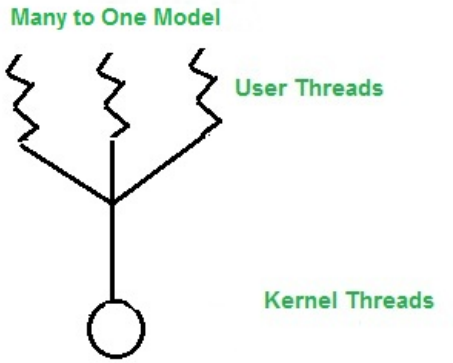


Many to One Model

multiple user threads mapped to one kernel thread.

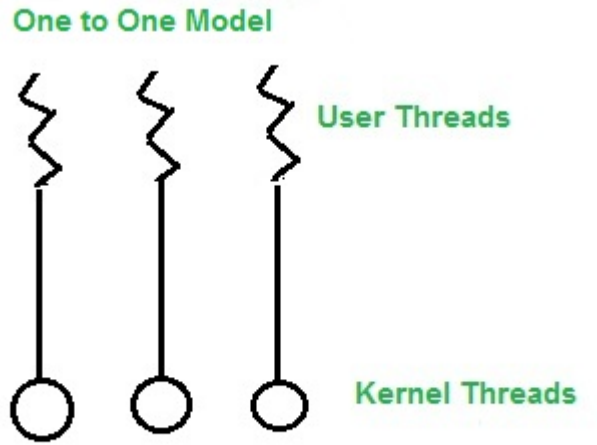
In this model when a user thread makes a blocking system call entire process blocks.

As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time.



One to One Model

- In this model, one to one relationship between kernel and user thread.
- In this model multiple thread can run on multiple processor.
- Problem with this model is that creating a user thread requires the corresponding kernel thread.



Thread usage

To Creates a thread	**int pthread_create(pthread_t *thread, pthread_attr_t *attr, void*(*start_routine)(void *), void *arg);** • **pthread_t variable, in which the thread ID of the new thread is stored** • **pthread_attr_t *attr this object controls details of how the thread interacts with the rest of the program NULL -default attribute.** • **void*(*start_routine)(void *),:The function the thread has to start executing** • **void *arg:the arguments that are to be passed to this function.** • **Eg: pthread_create(&tid2,NULL,thread2,NULL);**

Thread termination and waiting

When a thread terminates, it calls the pthread_exit function This function terminates the calling thread	void pthread_exit(void *retval);
pthread_join is the thread equivalent of wait that processes use to collect child processes.	int pthread_join(pthread_t th, void **thread_return); • first-the thread for which to wait • second -return value of the thread on which we want to the parent thread to wait. • pthread_join(tid1,NULL);

End

Operating System: Process Synchronisation
Semaphores

Process Synchronization

- Processes are categorized as one of the following two types:

- **Independent process** : execution of one process does not affects the execution of other processes.

- **Cooperative process :** execution of one process affects the execution of other processes.

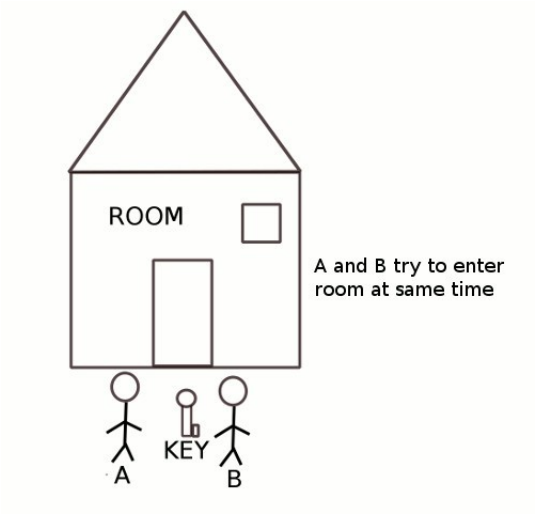
  - Process synchronization problem arises in the case of cooperative process also because resources are shared in cooperative processes.

  - **Race condition**

  - Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place.

Critical Section

- The Critical section code can be accessed by only one process at a time
- In the Critical section, there are many variables and functions that are shareable among different processes.




```
do
{
        Entry  Section

        critical section

        Exit  Section

        Remainder Section
}
while(True);
```

Any solution to the critical section problem must satisfy three requirements

Software solution to CSP

Semaphore

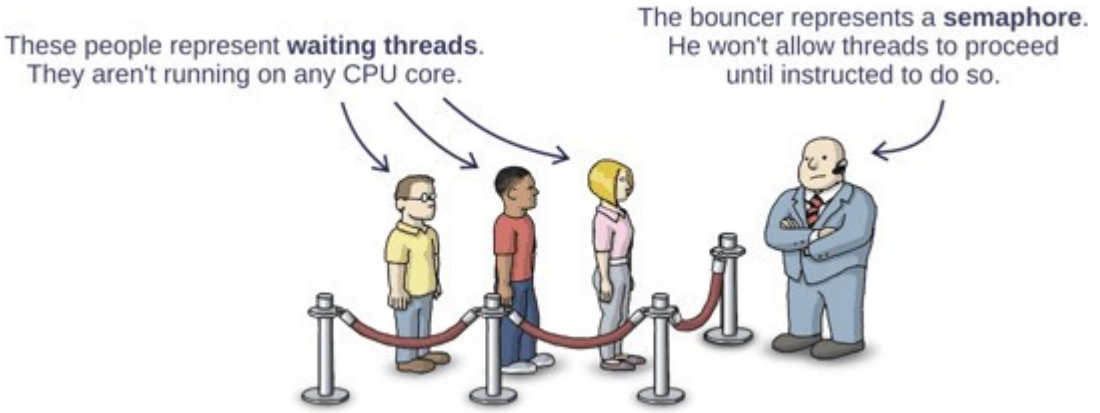
- **Mutual Exclusion** : If a process is executing in its critical section, then no other process is allowed to execute in the critical section.

- **Progress** : If a process is not executing its own critical section, then it should not stop any other process to access the Critical Section.

- **Bounded Waiting :** Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

Semaphores

- A semaphore is a variable used to help threads to work together without interfering with each other.



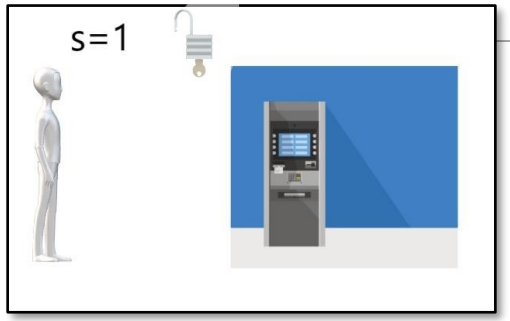
Semaphore

- A semaphore is an integer variable, which can be accessed only through two operations wait() and signal().
- Two types of semaphores : binary semaphores and counting semaphores.
- **Binary semaphores**
  - Binary semaphores take only the values between 0 to 1.
  - Its value is initialized to 1
- **Counting semaphores.**
  - Counting semaphores have the non-negative integer value.
  - the semaphore count is used to indicate the number of available resources.
  - It can be used to control access to a given resource

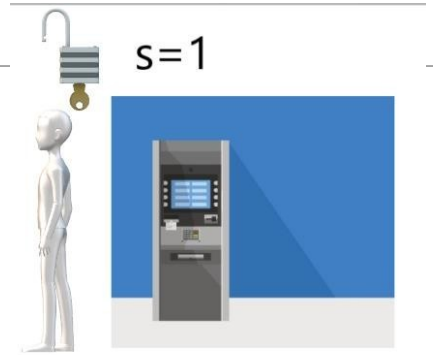
```
wait(S)
{
    while (S<=0);//no operation
    S--;
}
```

```
signal(S)
{
S++;
}
```

Binary Semaphore working

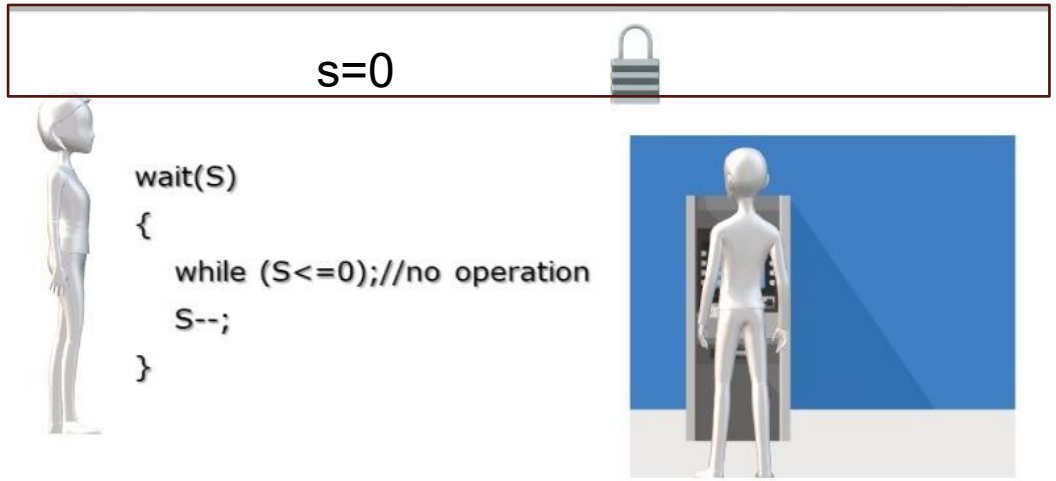



```
wait(S)
{
   while (S<=0);//no operation
   S--;
}
```




```
signal(S)
{
S++;
}
```

s=0



```
wait(S)
{
   while (S<=0);//no operation
   S--;
}
```

S=1

```
signal(S)
{
S++;
}
```

```
wait(S)
{
   while (S<=0);//no operation
   S--;
}
```

S=0

Counting semaphore

Semaphore implementation

POSIX Semaphore

#include <semaphore.h>

Semaphore datatype sem t

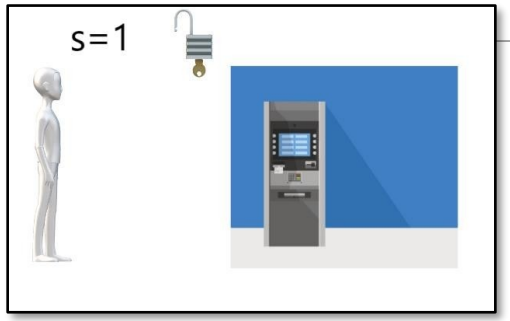
Semaphores in Linux
POSIX-semaphore:
sem_init(),sem_wait(),sem_post(),sem_getvalue(),
sem_destroy()
SystemV -semaphore
semget(),semop(),semctl()

Semaphore implementation

sem_init() initialize an semaphore	int sem_init(sem_t *sem, int pshared, unsigned int value); #include <semaphore.h> pshared : • indicates whether this semaphore is to be shared between the threads of a process, or between processes • 0, semaphore is shared between the threads of a process, else nonzero Value :specifies the initial value for the semaphore. sem_init() returns 0 on success; on error, -1 is returned,
sem_wait(sem_t *sem);	lock a semaphore If the semaphore's value is greater than zero, then the decrement and lock semaphore. If the semaphore currently has the value zero, then the call blocks until becomes grater than 0
int sem_post(sem_t *sem);	sem_post - unlock a semaphore
int sem_destroy(sem_t *sem);	sem_destroy - destroy an semaphore

Binary Semaphore working

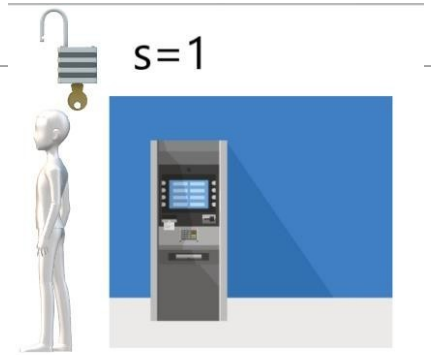
sem_init(&sem, 0, 1);



sem_wait(&sem);

```
wait(S)
{
   while (S<=0);//no operation
   S--;
}
```

sem_post(&sem);




```
signal(S)
{
S++;
}
```

For counting sem

```c
#include <semaphore.h>

sem_t s;
int main() {
  sem_init(&s, 0, 10); // returns -1 (=FAILED) on OS X
  sem_wait(&s); // Could do this 10 times without blocking
  sem_post(&s); // Announce that we've finished (and one more resource item is availa
  sem_destroy(&s); // release resources of the semaphore
}
```

For binary semaphore

```c
sem_t s;
sem_init(&s, 0, 1);

sem_wait(&s);
// Critical Section
sem_post(&s);
```

Mute

- A *mutex* is a mutual exclusion lock. Only one thread can hold the lock.
- Mutexes are used to protect data or other resources from concurrent access

#include <pthread.h> int pthread_mutex_init(mutex, pthread_mutexattr_t *attr);	Initialize a mutex .
int pthread_mutex_lock(pthread_mutex_t *mutex);	Lock a mutex.
int pthread_mutex_unlock(pthread_mutex_t *mutex);	Unlock a mutex.

Producer-Consumer problem

- The Producer-Consumer problem is a classic problem this is used for multi-process synchronization i.e. synchronization between more than one processes.
- In the producer-consumer problem, there is one Producer that is producing something and there is one Consumer that is consuming the products produced by the Producer. The producers and consumers share the same memory buffer that is of fixed-size.
- Criteria's that should be satisfied
- If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer.
- If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer.
- The producer and consumer should not access the buffer at the same time.

Solution

problems can be solved with the help of semaphores

```
void producer() {
    while(T) {
        produce()
        wait(E)
        wait(S)
        append()
        signal(S)
        signal(F)
    }
}
```

```
void consumer() {
    while(T) {
        wait(F)
        wait(S)
        take()
        signal(S)
        signal(E)
        use()
    }
}
```

In the producer-consumer problem, three semaphore variables can be used to solve problem

while() is used to produce data, again and again

produce() function is called to produce data by the producer.

wait(E) will reduce the value of the semaphore variable "E" by one,when the producer produces something and value of the empty space in the buffer.

wait(S) is used to set the semaphore variable "S" to "0" so that no other process can enter into the critical section.

append() function is used to append the newly produced data in the buffer.

signal(s) is used to set the semaphore variable "S" to "1"

signal(F) is used to increase the semaphore variable "F" by one because after adding the d ata into the buffer

Thank you

Operating System: Threads

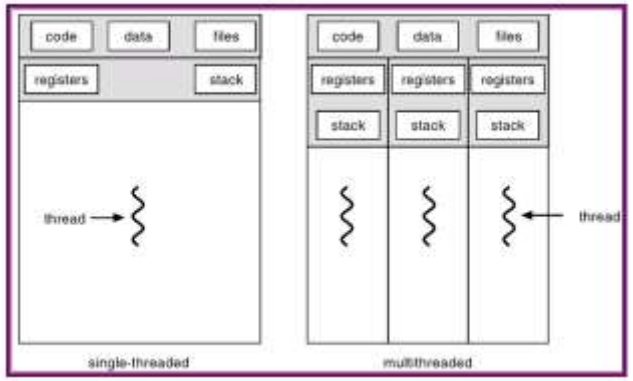
Threads

Threads

- A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.
- A thread is also known as lightweight process.
- The idea is to achieve parallelism by dividing a process into multiple threads.
- For example, in a browser, multiple tabs can be different threads.
- MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc.

Process vs Thread?

- The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.
- Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals).
- But, like process, a thread has its own program counter (PC),stack space.



Advantages of Thread over Process

1. **Responsiveness:** If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. **Faster context switch:** Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.

3. **Effective utilization of multiprocessor system:** If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. **Resource sharing:** Resources like code, data, and files can be shared among all threads within a process.

Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.

Three primary thread libraries:

POSIX Pthreads

Windows threads

Java threads

Types of threads

- **Kernel level thread**
  - kernel threads are implemented by OS.

User level thread

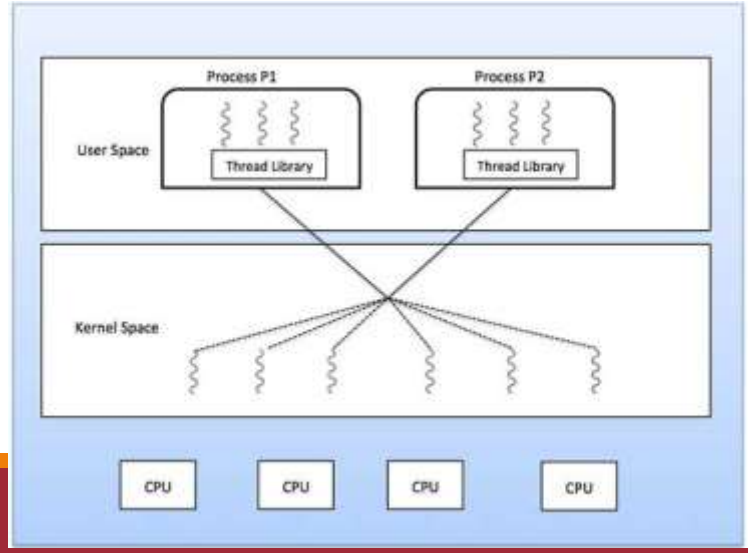
Multi threading models

- All the threads must have a relationship between them (i.e., user threads and kernel threads).
- Many operating systems support kernel thread and user thread in a combined way

- Many to many model.
- Many to one model.
- one to one model

Many to many

- In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads.
- Number of kernel level threads are specific to the machine.
- advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread.
- Thus, System doesn’t block if a particular thread is blocked.

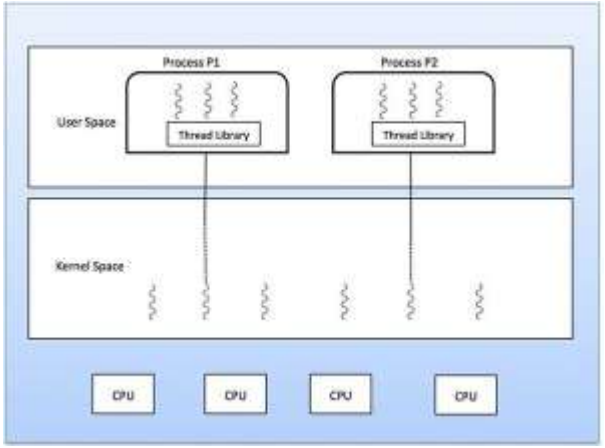


Many to One Model

multiple user threads mapped to one kernel thread.

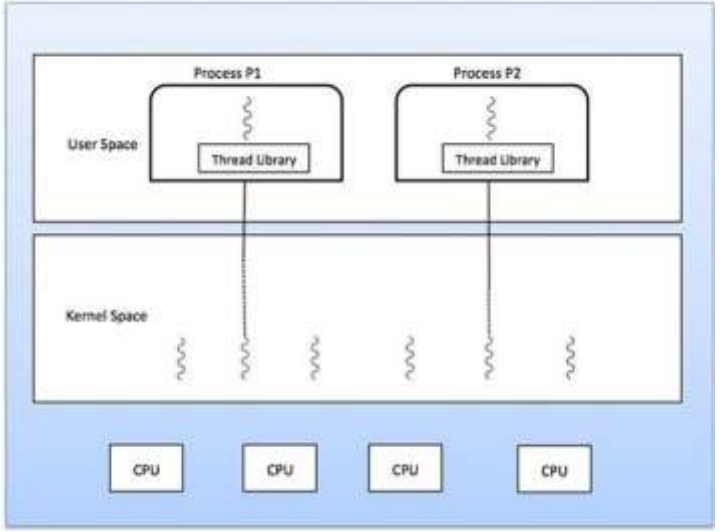
In this model when a user thread makes a blocking system call entire process blocks.

As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time.



One to One Model

- In this model, one to one relationship between kernel and user thread.
- In this model multiple thread can run on multiple processor.
- Problem with this model is that creating a user thread requires the corresponding kernel thread.



Thread Implementation

To create a thread

To Creates a thread	**#include <pthread.h>** pthread_create (thread, attr, start_routine, arg) • **thread** the thread ID of the new thread is stored • **attr** it controls details of how the thread interacts with the rest of the program NULL -default attribute . • **start_routine:**The function the thread has to start executing • **arg:**the arguments that are to be passed to this function.

To compile in Linux

gcc filename.c -lpthread

Thread termination and waiting

When a thread terminates, it calls the pthread_exit function This function terminates the calling thread	#include <pthread.h> pthread_exit (status)
pthread_join is the thread equivalent of wait that processes use to collect child processes.	int pthread_join(thread,thread_return); • first-the thread for which to wait • second -return value of the thread on which we want to the parent thread to wait. • pthread_join(tid1,NULL);

End

Operating System: Memory management

Functionality of memory management

- Keep track of every memory allocation
- Track whether a memory is allocated or not
- Track how much memory is allocated.
- It takes the decision which process will get memory and when.
- Updates the status of memory location when it is freed or allocated.
- Protection : ensuring that user address space should not use kernel address space.

Memory management

Two main functions of os in memory management

- How the process in secondary memory will allocated to main memory .
- The address translation.
    - logical address is genearated by cpu can be used to access sec memory, but to access main memory it require physical address.

Goal of memory management

- Space utilization
  - Fragmentation : there is chance of loss memory due to fragmentation
  - Keep fragmentation as small as possible.
- How to run larger program in smaller memory area
  - Assume pgm 1000kb -run process in 500kb.
  - Done by using virtual memory.

How the process in secmemory will allocated to main memory

- Main memory usually has two partitions –
  - Low Memory – Operating system resides in this memory.
  - High Memory – User processes are held in high memory.

- **Memory management technique.**
- **Contiguous**
- **Non contiguous**
  - Process parts are stored distributed in memory
  - Segmentation ,Paging

Memory allocation policies

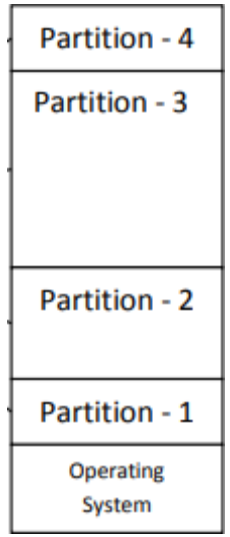
- **Non –contiguous :**
  - break the process into pieces and Keep them in different areas of memory.
  - Segmentation ,Paging
- **Contiguous**
  - Process as unit is completely stored .
  - Allocation of process brought from secmemory to main memory must be placed contiguous.

  •Contiguous allocation is one early method
  - It suffers from **external fragmentation** where u have the requested memory is available ,but they are not contiguous for ur process to store.

    •Fixed size partioning & Variable size partitioning

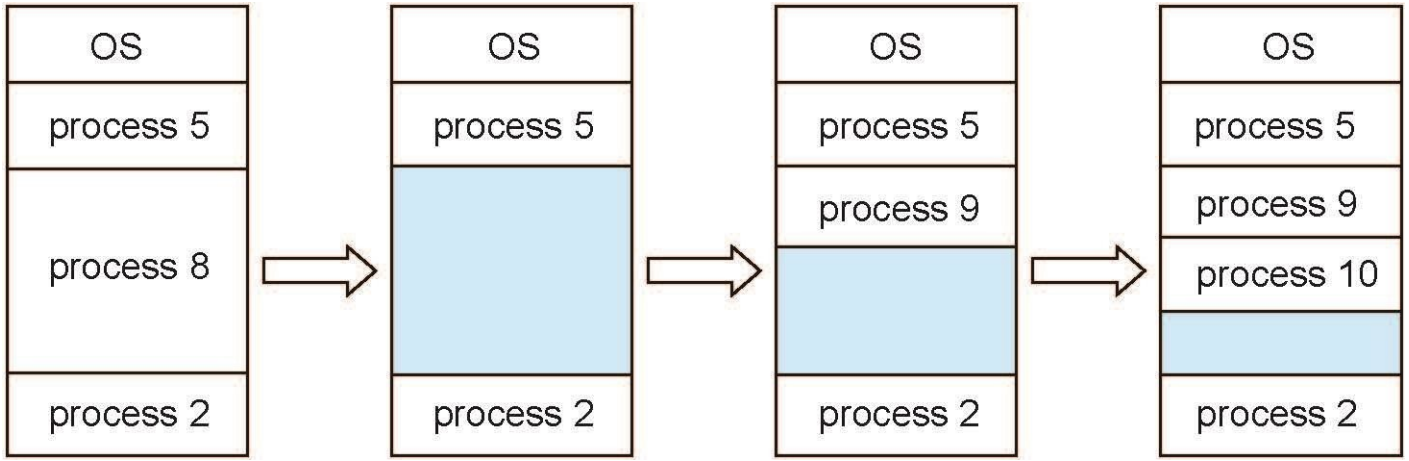
Contiguous memory allocation policy :Fixed size partioning

- For memory allocation ,memory is divided into fixed size partitions of different size
- Fixed size will be fixed which cannot be changed
- When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process
- Degree of multiprogramming is bound by no of partitions.
- 1 process= 1 partition



- If a process of 4kb it will be kept in the 5k sized so rest 1kb is waste ,which is known as **internal fragmentation…**
- Partition cannot be reused i.e only 1 process allowed in 1 partition

- **Variable size partitioning**
  - large block of memory is available as the process arrives its allocated
  - The entire partition is not divided into fixed size.so it does not suffers from internal fragmentation.
  - Hole – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it



Dynamic Storage-Allocation
FIRST | BEST | WORST | algorithms

- In both fixed size and variable partition schemes there are 3 algorithms which can be used to allocation.
- They are first fit,best fit,worst fit.
- **First-fit**:
  - Allocate the ***first*** hole that is big enough
- **Best-fit**:
  - Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
  - Produces the smallest leftover hole
- **Worst-fit**:
  - Allocate the ***largest*** hole; must also search entire list
  - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

First fit

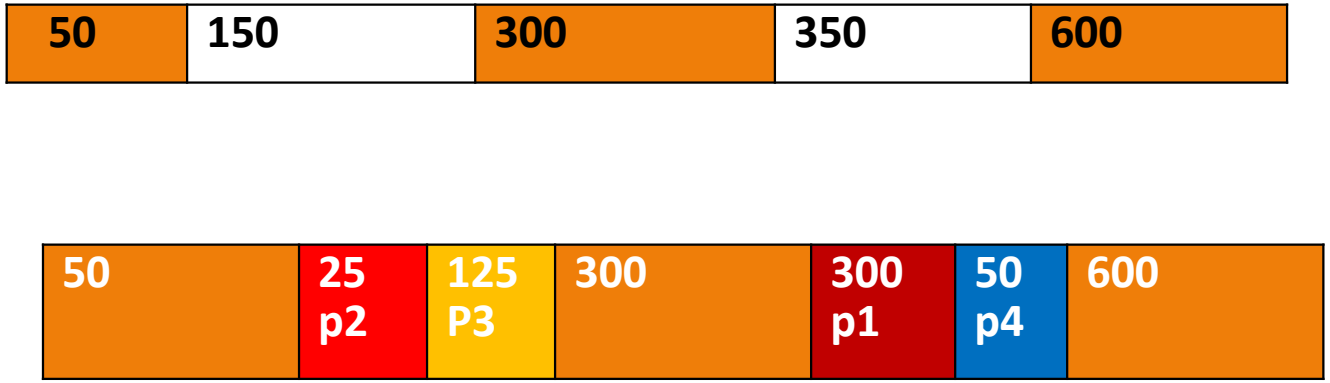
- First fit: searches the location starting from beginning and select a hole that big enough to hold our request

P1-300

P2-25

P3-125

P4-50



Best fit

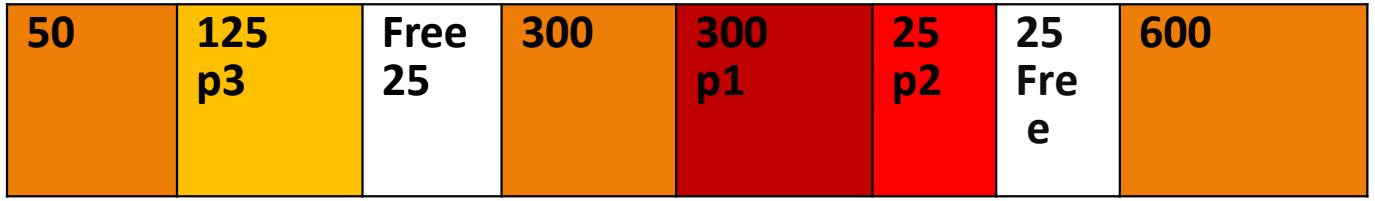
Searches the entire list . then choose the smallest hole that is big enough to allocate the process

P1-300

P2-25

P3-125

P4-50



here last p4 cannot be fitted here it suffers from external fragmentation.

The best fit is the good algorithm for fixed size partitioning and best fit perform worst for variable size.

Worst fit

search entire memory and allocate largest block.

P1-300

P2-25

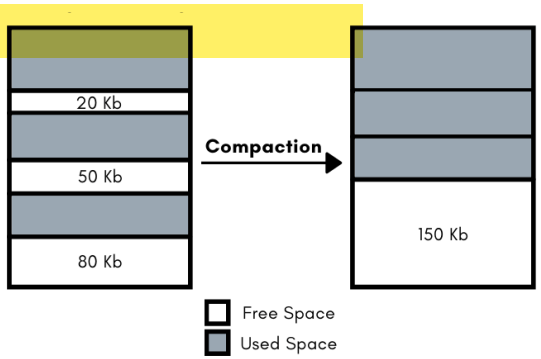
P3-125

P4-50

Compaction

- Compaction is a technique to collect all the free memory present in form of
- fragments into one large chunk of free memory, which can be used to run other processes.

- It does that by moving all the processes towards one end of the memory and
- all the available free space towards the other end of the memor
contiguous.
- Compaction is one of the solutions to External Fragmentation.



Address translation in contiguous memory allocation

Two main functions of os in memory management

- How the process in secondary memory will allocated to main memory .
- **The address translation.**
    - logical address is generated by cpu can be used to access sec memory, but to access main memory it require physical address.

Address translation

- The cpu access main memory, but the address it (cpu) generates is **logical address which can be used to access secondary memory**.
- The **main memory can be accessed using physical address** so the logical address has to be converted to physical address.

- In **contiguous policy** it is easy to know the address because we take a process from sec memory in a contiguous fashion and place it in the main memory so we know the base address we could calculate the rest address.

- In **non contiguous** it will be difficult ,since the entire process is divided into different fragments and placed in the memory .

Logical vs. Physical Address Space

Logical address – generated by the CPU; also referred to as **virtual address**

Physical address – address seen by the memory unit

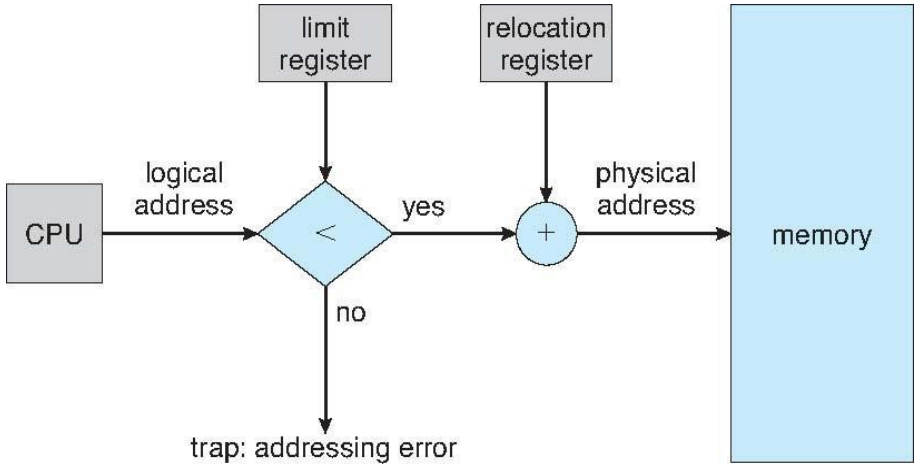
During compile-time and load-time ,memory address binding logical and physical addresses are same.

differ in execution-time address-binding scheme

Logical address space is the set of all logical addresses generated for a program

Physical address space is the set of all physical addresses generated for a program

Address translation in contiguous memory allocation



The process in s.m (sec memory) is loaded to main memory.

Memory-Management Unit

Does the runtime mapping logical to physical address space.

Done with the help of relocation register.

relocation register contains the base address of the process in main memory .

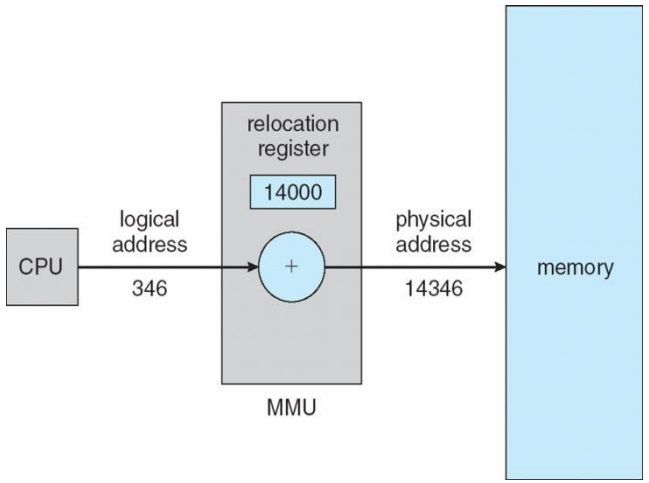
The logical address generated by CPU can be added with **the base address** in the RR to get the **physical address.**

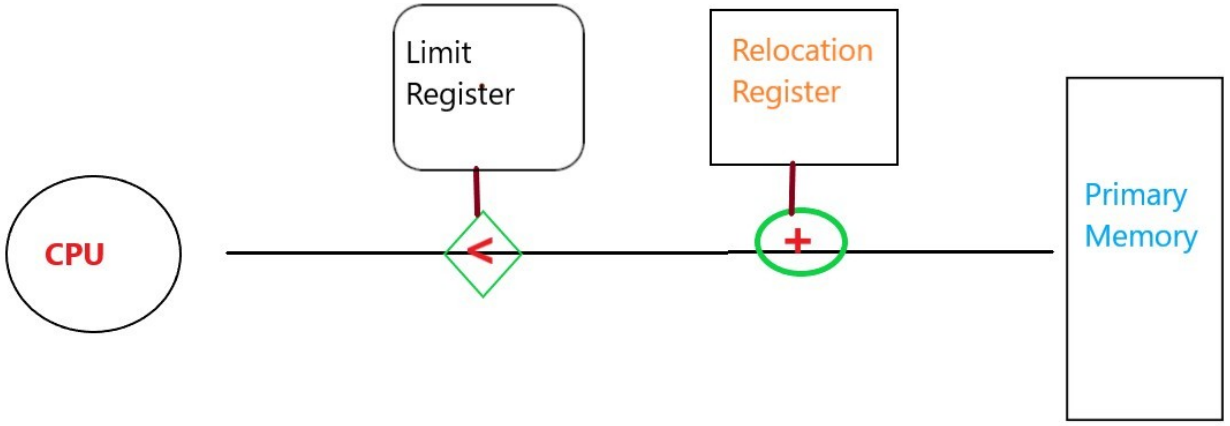
Limit register contains the size of the process in main memory.

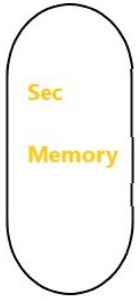
If the system illegally or intentionally produce a logical address,which gets added with BA and produce a physical address , which is not the address of current process .it will become an illegal access to other process location so for these reason we have a limit register LR.

If the system illegally or intentionally produce a logical address,which gets added with BA and produce a physical address , which is not the address of current process .it will become an illegal access to other process location so for these reason we have a **limit register LR.**

Relocation register

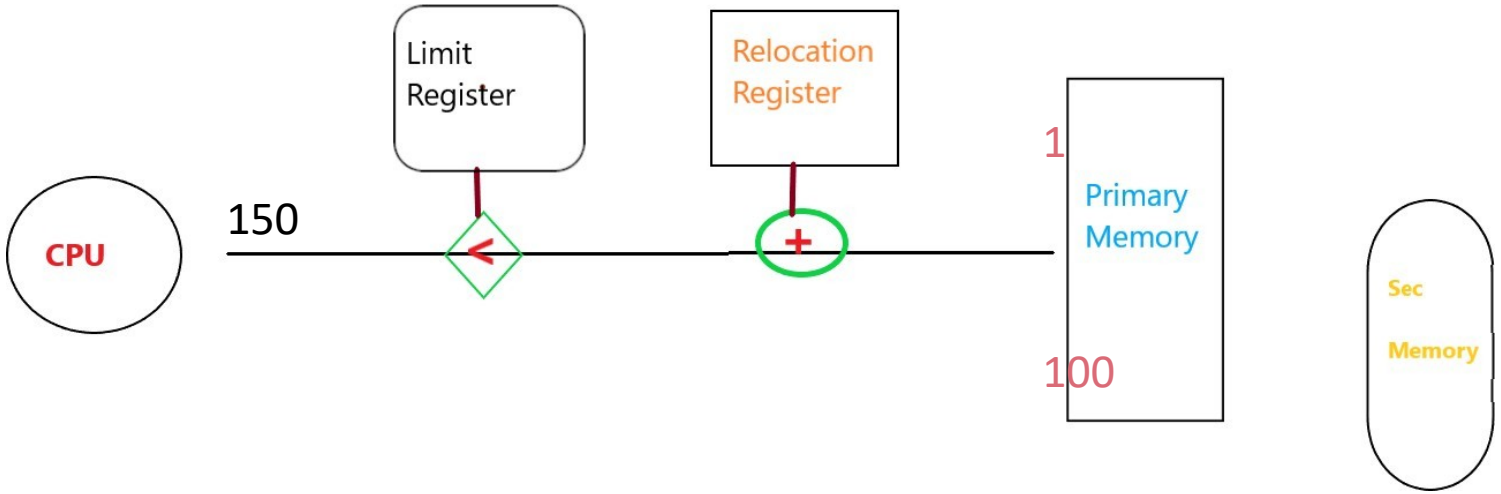






Adress translation

but if cpu generate 150 it cannot be added because if we add it will be greater and some other location so its not allowed.



Numericals

		LR	RR
P0	450	500	1200
P1	300	275	550
P2	210	212	880

Disadvantage of contiguous poilcy

External Fragmentation – total memory space exists to satisfy a request, but it is not contiguous

Internal Fragmentation – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

Non- Contiguous memory management policy

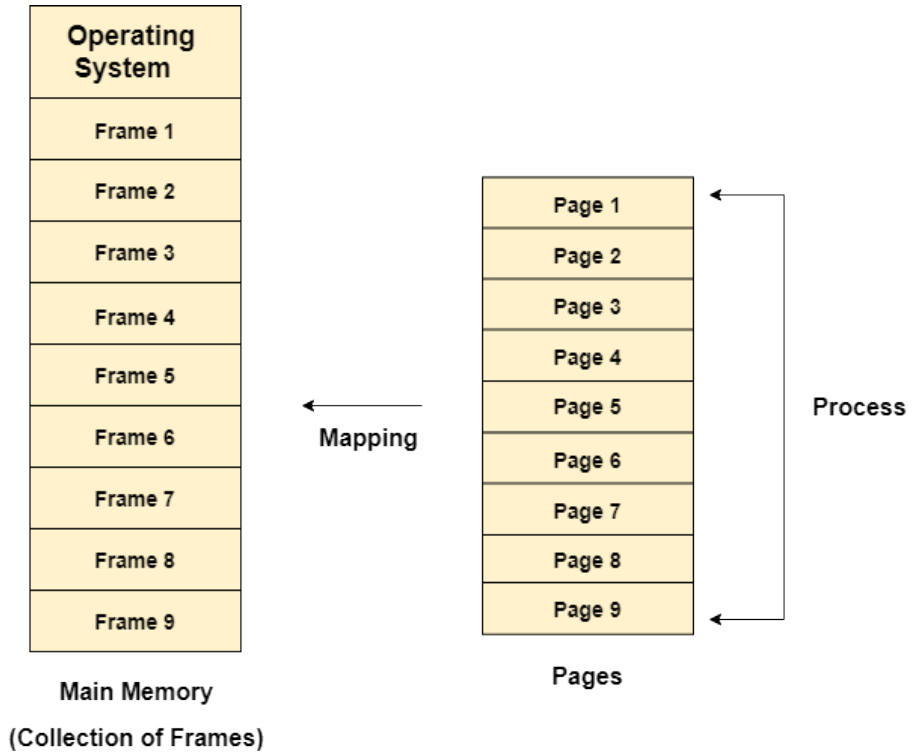
In Non- Contiguous memory management policy two approaches

Paging

Segmentation

paging

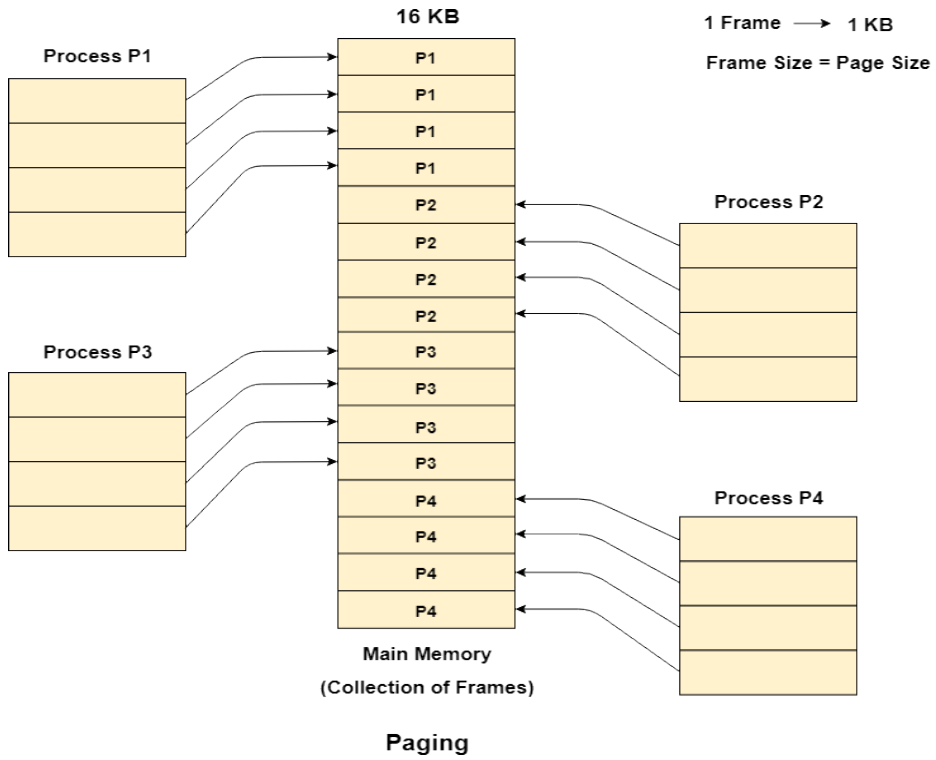
- The sec memory is divided into fixed size same partitions or blocks ,which are called **pages.**
- The main memory is also divided into fixed size partitions of same size called **frames.**
- Both size of frame and page will be same i.e if page is 1kb the frame also will be 1kb.



Pages and frames

- There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.
- The process in sec memory can be splitted into pages and placed in the main memoy
- frames.

eliminate the external fragmentation.



Paging

The address translation with paging.

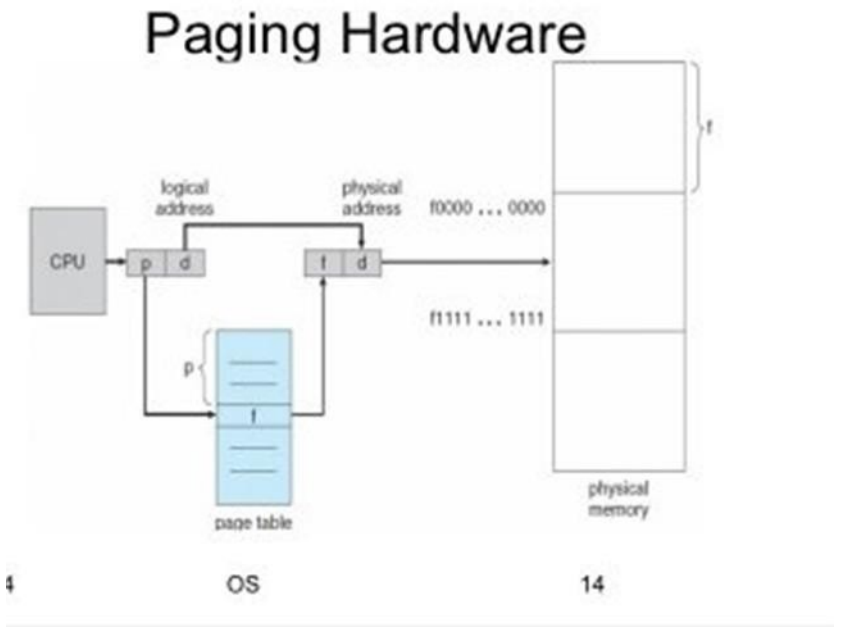
- The cpu generates logical address which can be used to access sec.memory.
- The logical address can be divided into page number (p)+ instruction offset(d).
- the cpu generates address in such a format say it want to access the pagenumber 2 instruction (offset)number 24,
- We can find it from main memory in different way.

Method one

- **Method one**
- the base address of the process is given in this scenario (the address of first page).
- in this approach the first page contain pointer to sec.page and sec.page contain pointer to next ,like link list.
- so there is no need of knowing all the page number.
- So by looking at page1 we can find where page 2 is stored and then find instruction 24.
- But this mechanism the access will be very slow.

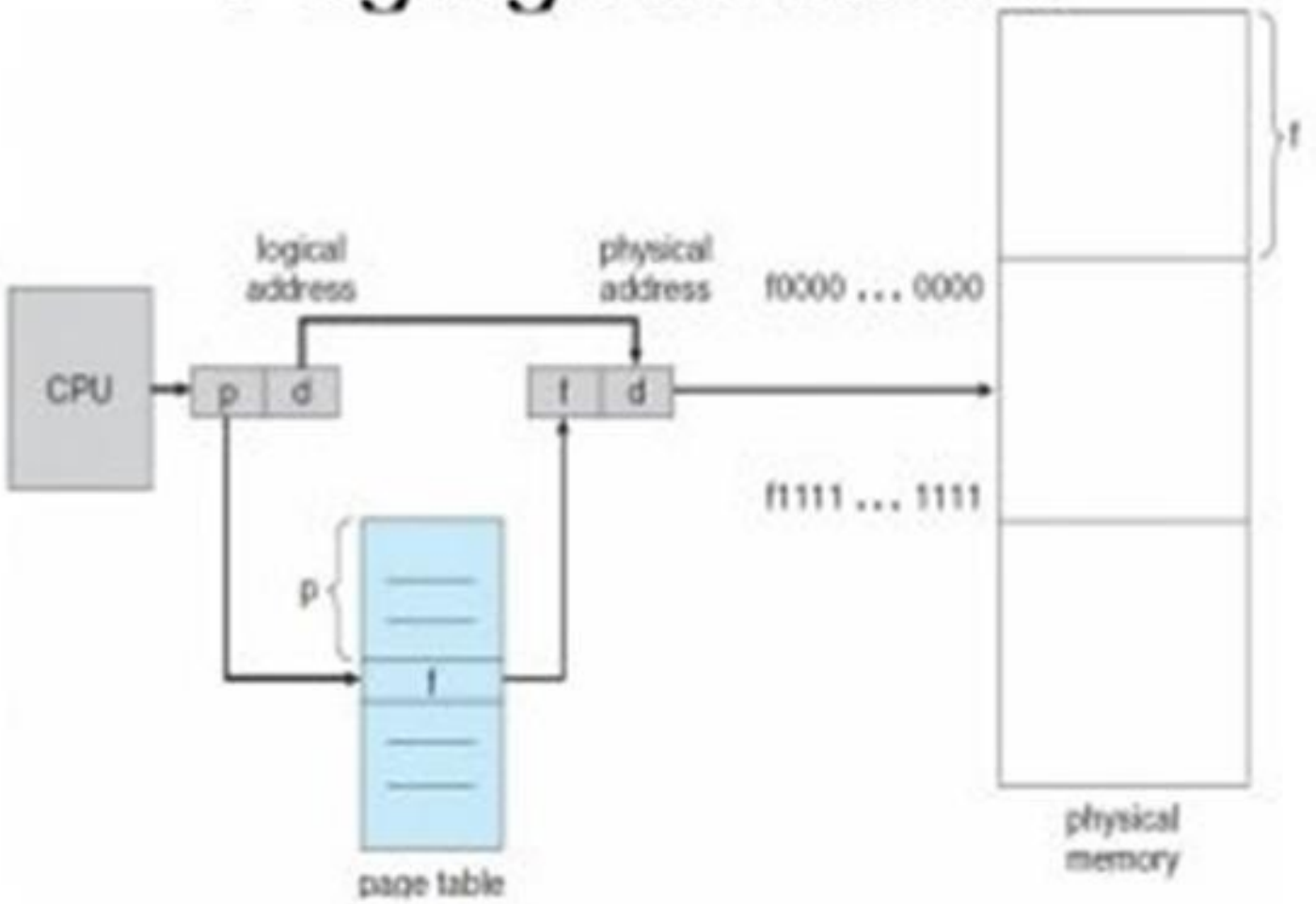
Pagetable

- **Method 2**:In this method we have a pagetable which is a datastructure (not a hardware) which contains the base address of each pages of a process in main memory.
- Each process will be having separate page table.
- The no of pages a process has that much entry
will be there in page table.
- The page table is nothing but the index to framenumber of page stored in main memory
- page number generates a logical address, it will access the pagetable.
- From page table at index p2 we get frame number(or base address where the page is stored in mainmemory) add it with d (instruction offset) to get the physical address to access the mainmemory.



How does it knows the address of page table.?from the PTBR.(a register holding the address of page table)it is stored in PCB Process Control Block.

Paging Hardware



Advantage

- The paging mechanism is actually introduce to reduce external fragmentation,
- Disadvantage
- but by paging mechanism it require 2 time access memory if you want to access something from memory.so it increases the time.

Translation look aside buffer

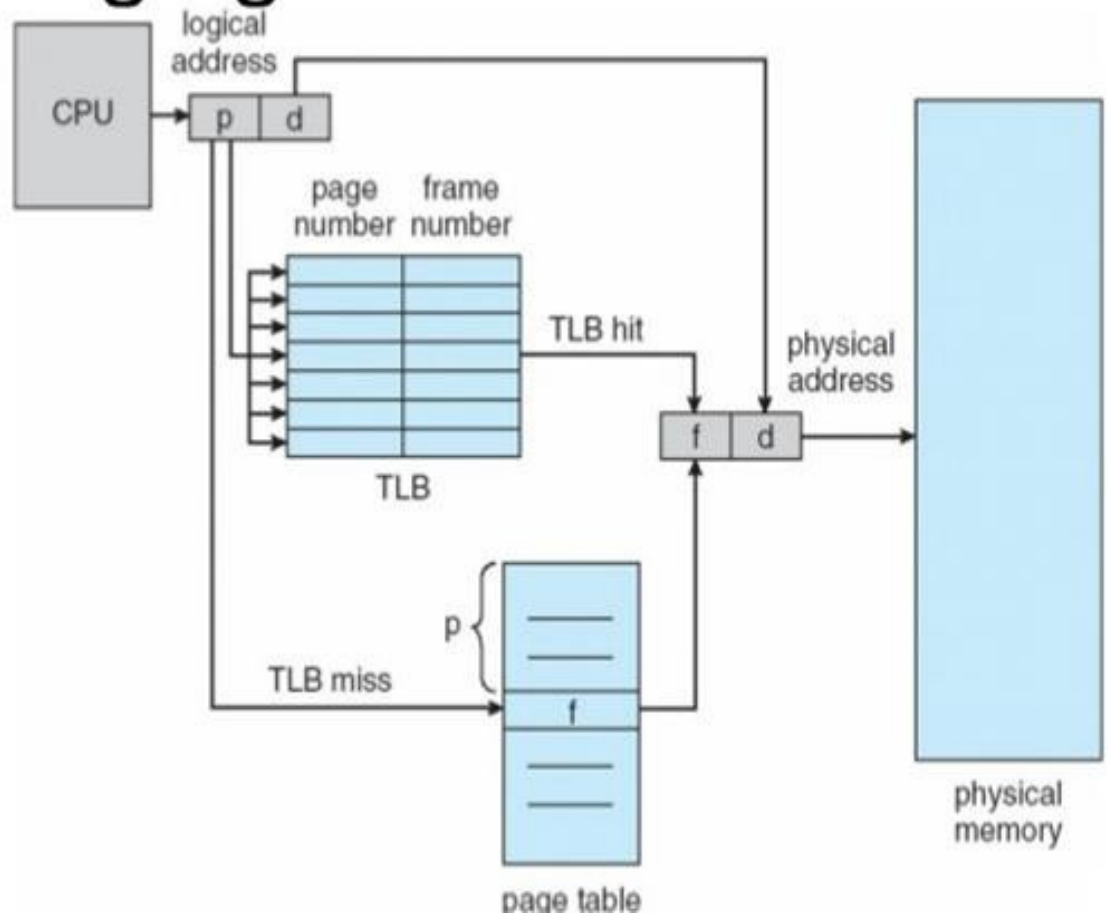
- To reduce this timing we can use Translation look aside buffer.small fast look up hardware cache.
- When a page is accessed from pagetable first time. Its info will be stored in the TLB.
- The tlb contain page number and framenumber.
- The TLB contains only a few of the page-table entries.

- When a logical address is generated by the CPU, its page number is presented to the TLB.
- **If the page number is found(TLB hit)**,its frame number is immediately available and is used to access memory

- **If the page number is not in the TLB (TLB miss)**
- page table is accessed and frame number is obtained and memory is accessed .
- In addition, add the page number and frame number to the TLB, so that they will be found quickly on the next reference.

- If the TLB is already full of entries, the operating system must select one for replacement.
- During the context switch the TLB will be cleared and a new process pages will be kept in TLB.

Paging Hardware With TLB

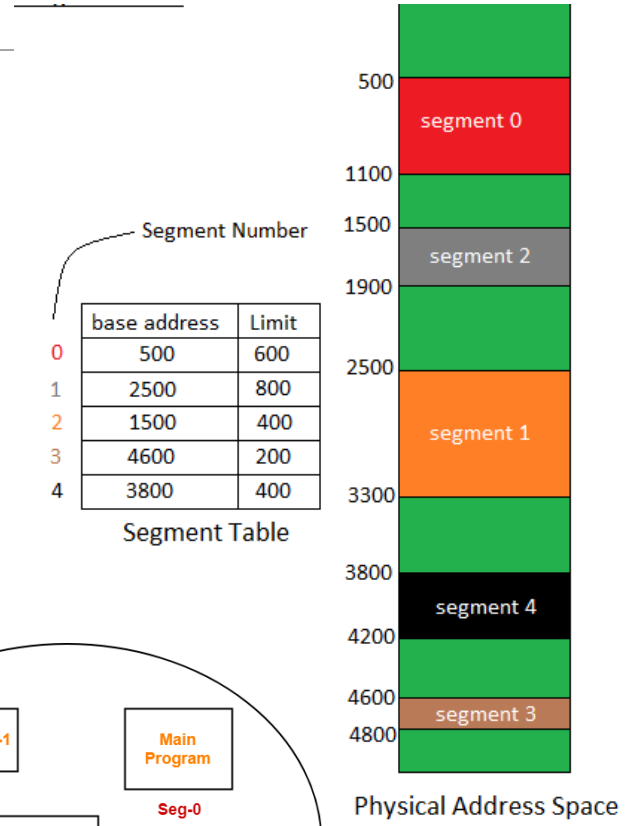


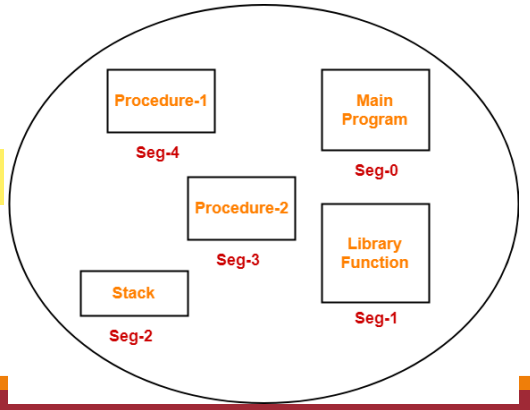
- When a logical address is generated by the CPU, its page number is presented to the TLB.
- **If the page number is found(TLB hit)**,its frame number is immediately available and is used to access memory

- If the page number is not in the TLB (**TLB miss**)
- page table is accessed and frame number is obtained and memory is accessed .
- In addition, add the page number and frame number to the TLB, so that they will be found quickly on the next reference.

Segmentation

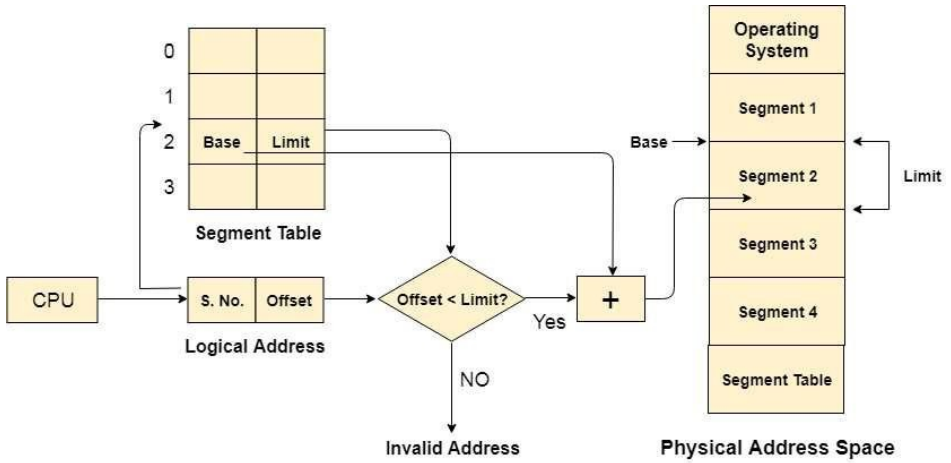
- In segmentation, secondary memory and main memory are divided into partitions called segments of unequal size.
- The size of partitions depend on the length of modules.
- **Segment Table-** Segment table is a table that stores the information about each segment of the process.
  - **Base Address:** It contains the starting physical address where the segments reside in memory.
  - **Limit:** It specifies the length or size of
    - the segment.
- No internal fragmentation, but external fragmentation

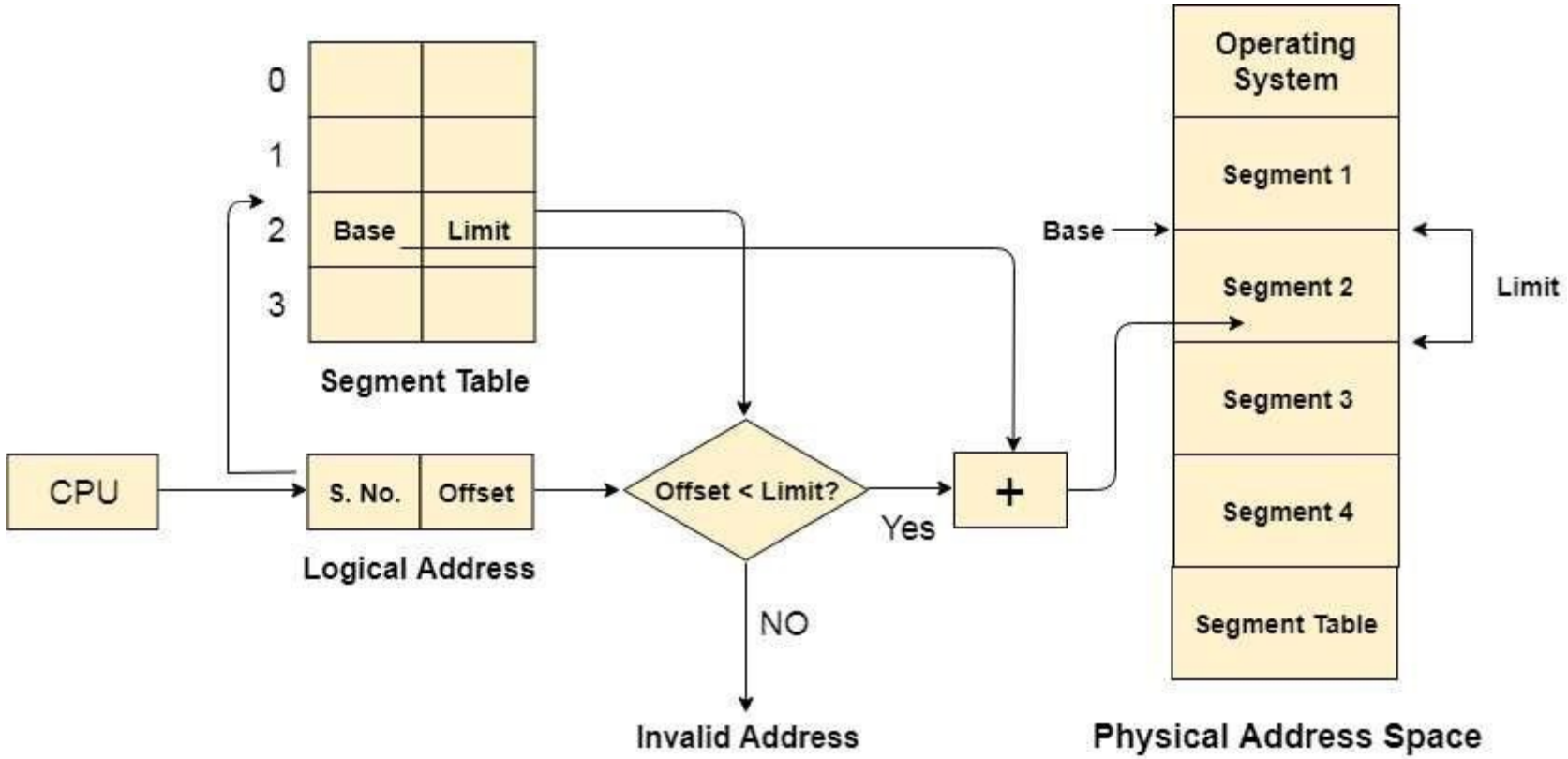




Segmentation Address translation

- **cpu generates->**Segment Number and Segment Offset (logical address)
- For the generated **segment number**, corresponding entry is located in the segment table
- **segment offset** is compared with the **limit** (size) of the segment.
- If valid ,**segment offset** is added with the **base address** of the segment.




0
1
2
3
Base
Limit
Segment Table
CPU
S. No.
Offset
Logical Address
Offset < Limit?
Yes
NO
Invalid Address
+
Operating System
Segment 1
Base
Segment 2
Limit
Segment 3
Segment 4
Segment Table
Physical Address Space

Virtual memory

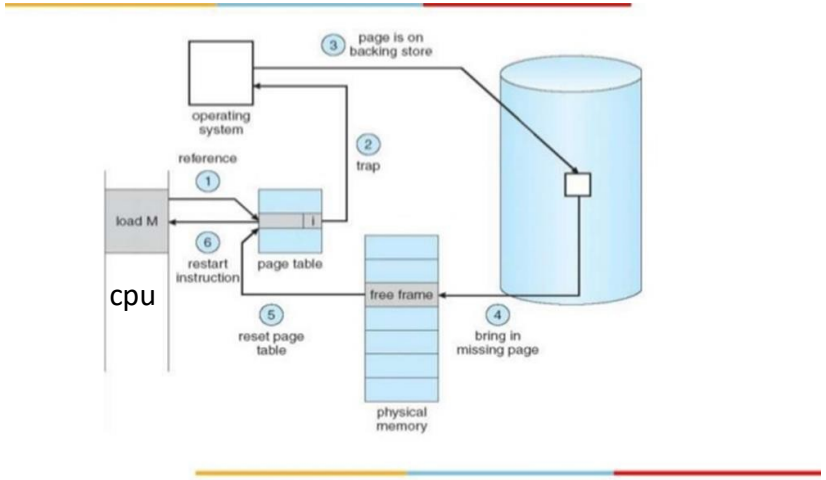
- Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory
- When the RAM is low,
- With virtual memory, the system looks at RAM for areas that have not been used recently and copy them onto the hard disk. This frees up space in RAM to load the new application.
- The area of the hard disk that stores the RAM image is called a page file. It holds pages of RAM on the hard disk, and the operating system moves data back and forth between the page file and RAM. On a Windows machine, page files have a .SWP extension.

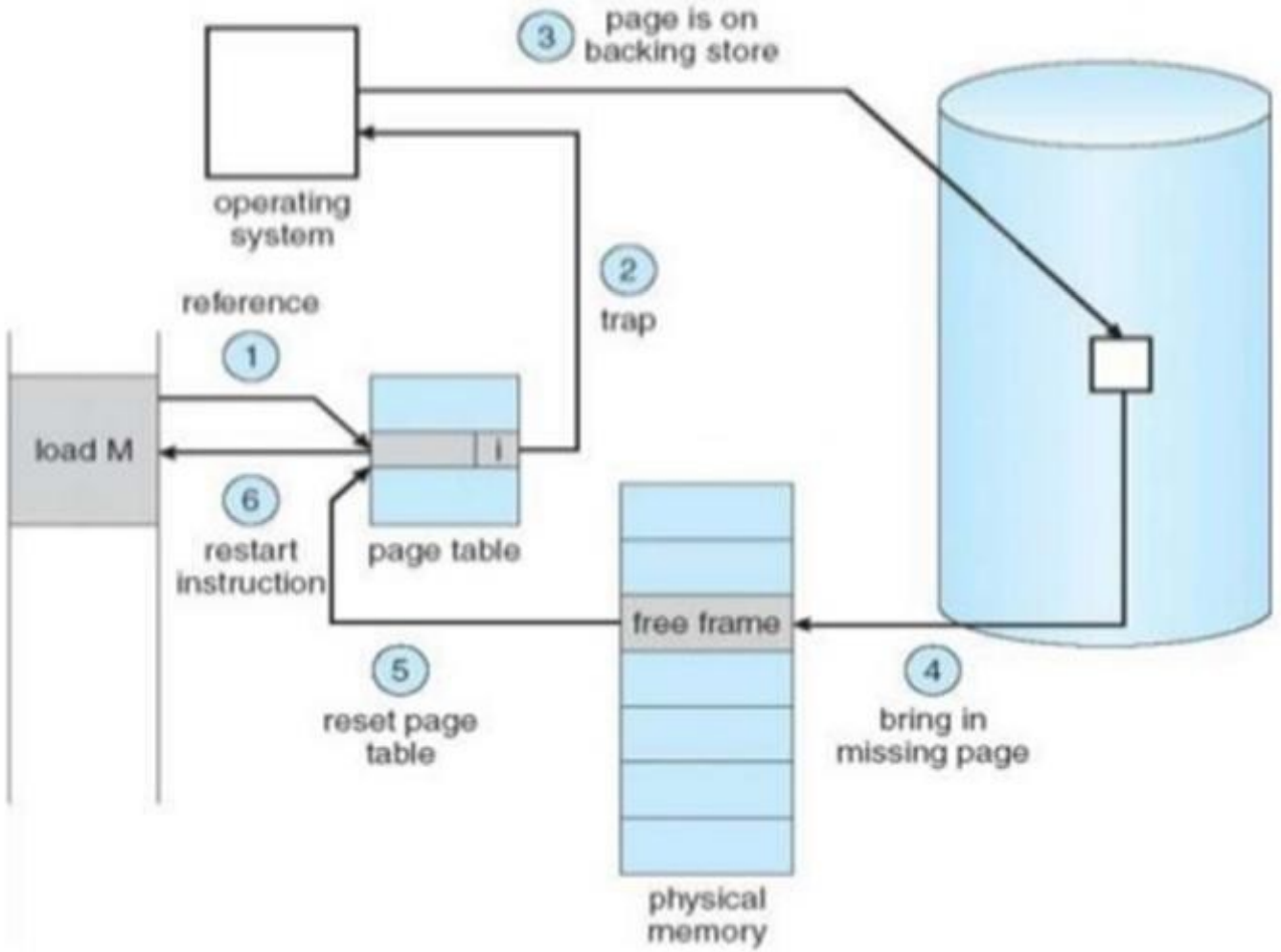
Demand Paging

The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.

How the page is brought

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault. Page fault
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
5. The page table will updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.




page is on backing store
operating system
reference
trap
load M
page table
restart instruction
free frame
reset page table
bring in missing page
physical memory

DirtyBit

In order to reduce the page fault service time, a special bit called the dirty bit will be associated with each page .

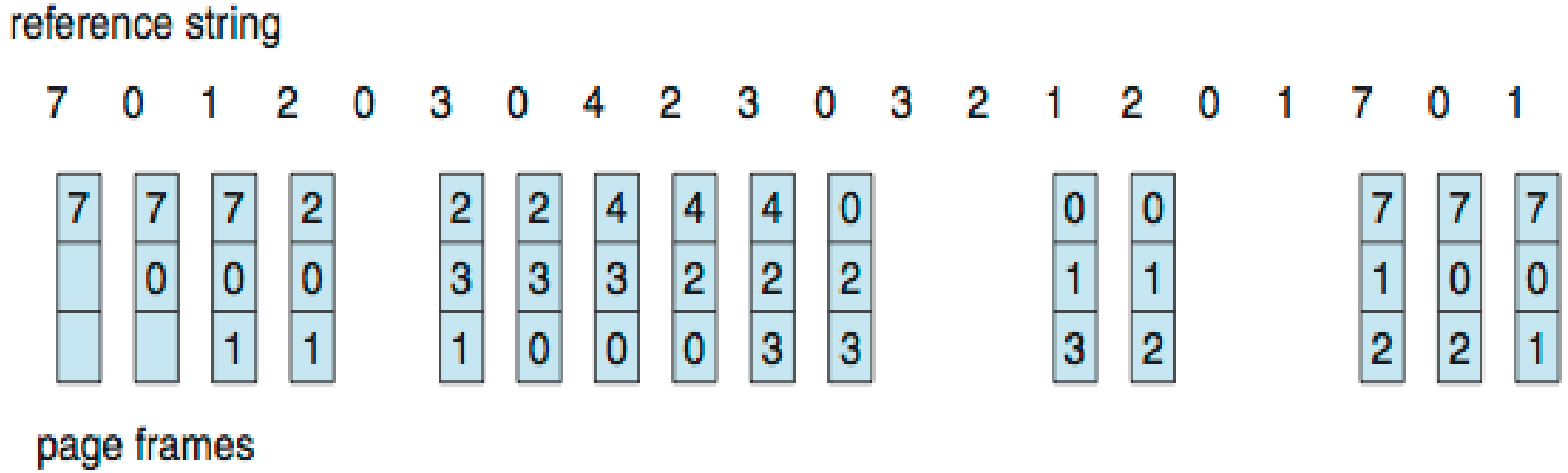
The dirty bit is set to 1 whenever it is modified.while selecting a victim page using page replacement algorithm. This value is tested ,if it is set to 1 means that page has modified after swapped in ,so the page have to be written into sec memory.

Page replacement algorithms

1. Find the location of the page requested by ongoing process on the disk.
2. Find a free frame. If there is a free frame, use it.
3. **If there is no free frame, use a page-replacement algorithm** to select any existing frame to be replaced, such frame is known as victim frame.
4. Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
5. Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
6. Restart the process that was waiting for this page.

First-In-First-Out (FIFO) Algorithm

- This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal
- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



Belady's Anomaly

- FIFO suffers from belady's anomaly.
- belady's anomaly:
  - increasing the number of page frames results in an increases the number of page faults for a given memory access pattern.
- 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	1	1	2	3	4	1	1	1	2	5	5
	2	2	3	4	1	2	2	2	5	3	3
		3	4	1	2	5	5	5	3	4	4
PF	PF	PF	PF	PF	PF	PF	X	X	PF	PF	X

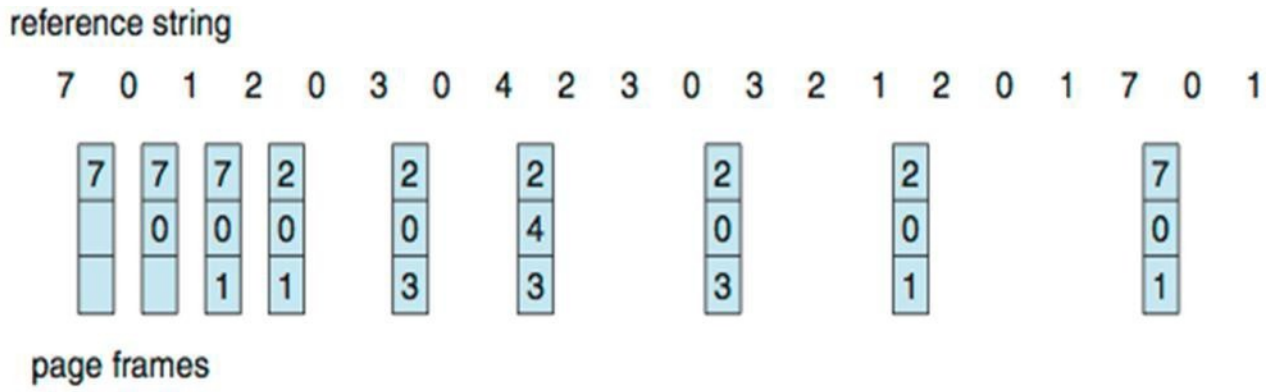
3 frames

1	1	1	1	1	1	2	3	4	5	1	2
	2	2	2	2	2	3	4	5	1	2	3
		3	3	3	3	4	5	1	2	3	4
			4	4	4	5	1	2	3	4	5
PF	PF	PF	PF	X	X	PF	PF	PF	PF	PF	PF

4 frames

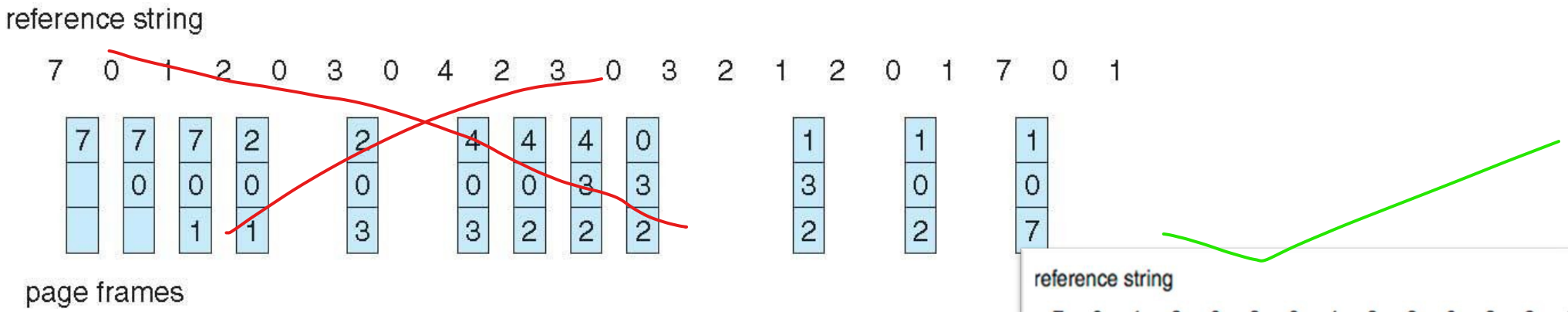
Optimal Algorithm

- Replace page that will not be used for longest period of time
- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
- 9 page faults
- **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**

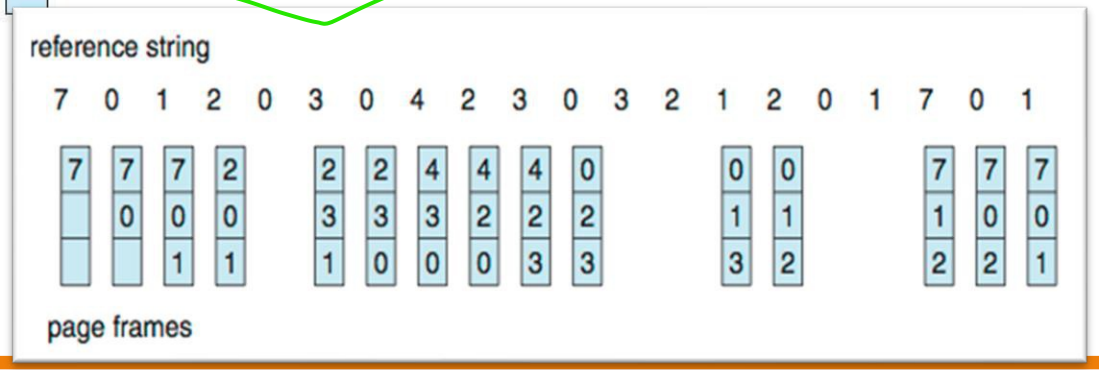


LRU

- this algorithm replaces the page which has not been referred for a long time. This algorithm is just opposite to the optimal page replacement algorithm. In this, we look at the past instead of staring at future.
- Associate time of last use with each page
- **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used



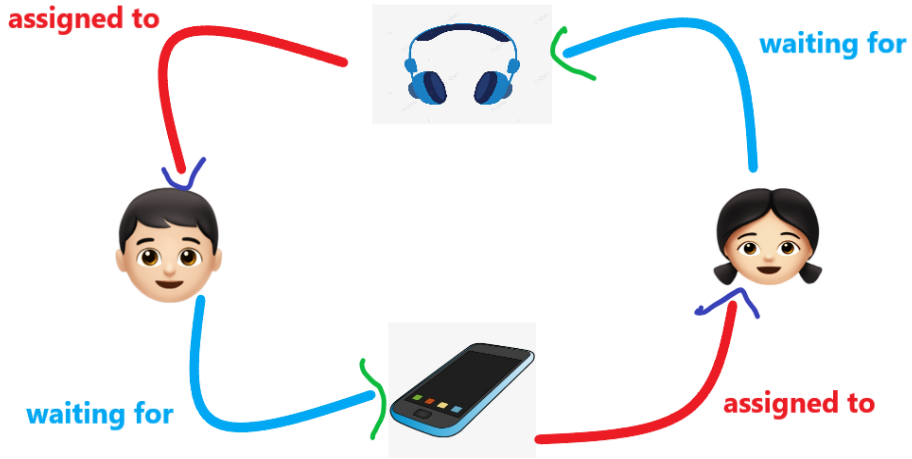
Thrashing

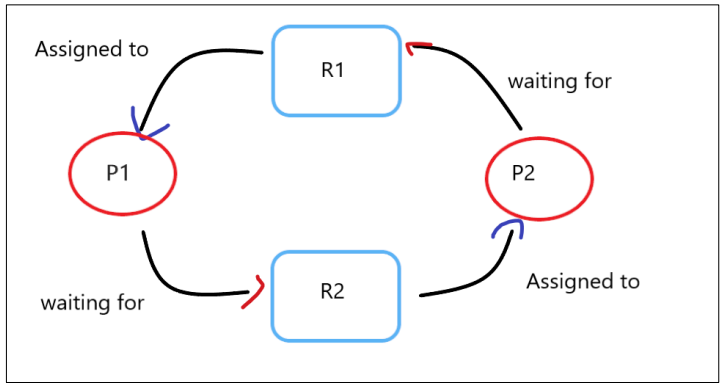
- A process that is spending more time paging than executing is said to be thrashing. In other words it means, that the process doesn't have enough frames to hold all the pages for its execution, so it is swapping pages in and out very frequently to keep executing. Sometimes, the pages which will be required in the near future have to be swapped out.

DEADLOCK IN OS

DEADLOCK

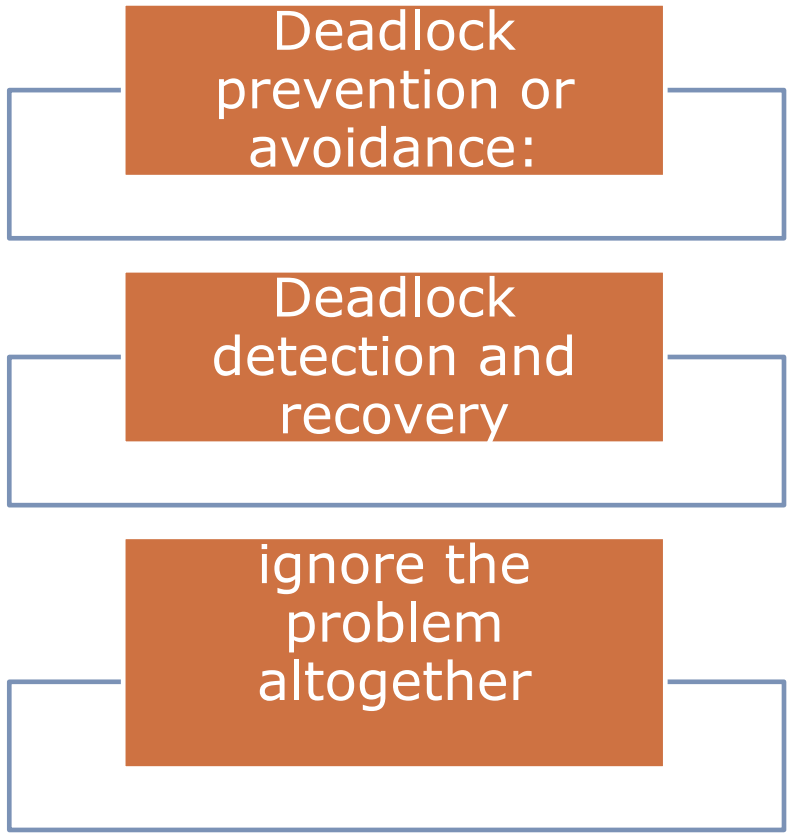
- Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.





Process 1 is holding Resource 1 and waiting for resource 2
which is acquired by process 2, and process 2 is waiting for resource 1

METHODS FOR HANDLING DEADLOCK



To ensure that deadlocks never occur, the system can use eit
Deadlock prevention or a deadlock-avoidance scheme.

Allow the system to enter a deadlocked state,
detect it, and recover

ignore the problem altogether and pretend that
deadlocks never
occur in the system.

NECESSARY CONDITIONS FOR DEADLOCKS

- **Mutual Exclusion**

  A resource can only be shared in mutually exclusive manner. Only one process can access resource at a time

- **Hold and Wait**

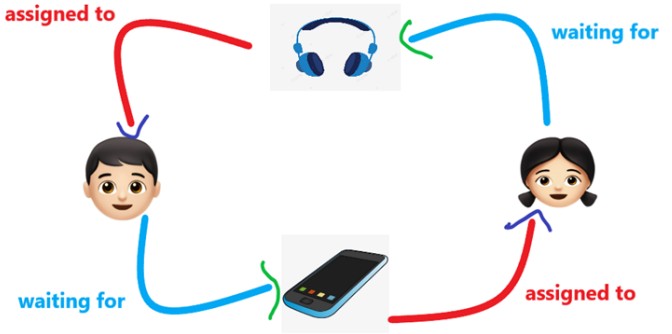
  A process waits for some resources while holding another resource at the same time.

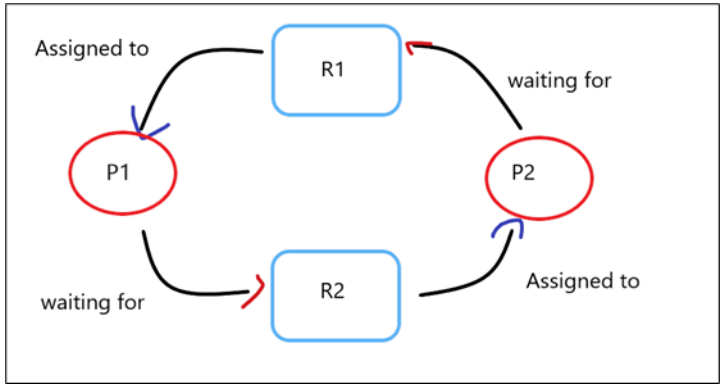
- **No preemption**

  The process which once scheduled will be executed till the completion. A resource can be released only voluntarily by the process holding it after that process has finished its task

- **Circular Wait**

  All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process





Deadlock happens only when Mutual Exclusion, hold and wait, No preemption and circular wait holds simultaneously

- To ensure that deadlocks never occur, the system can use either a deadlock prevention or a deadlock-avoidance scheme.

DEADLOCK PREVENTION

- Deadlock can be prevented by eliminating any of the necessary conditions for deadlock.

Eliminate Mutual Exclusion

- if we can be able to violate resources behaving in the mutually exclusive manner then the deadlock can be prevented.

Eliminate No Premption

- Preempt resources from the process when resources required by other high priority processes.

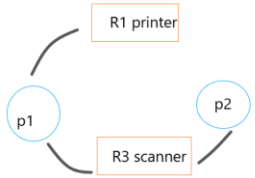
Eliminate Hold and Wait:

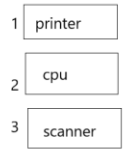
- In this condition, processes must be stopped from holding single or multiple resources while simultaneously waiting for one or more others.

Eliminate Circular Wait

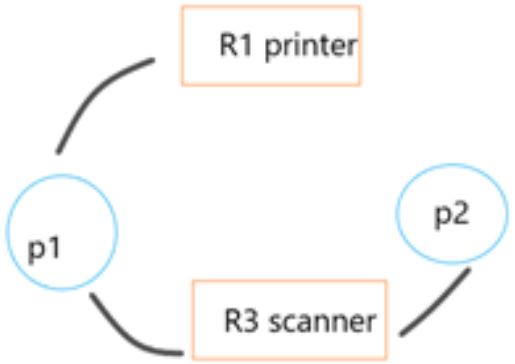
- This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed

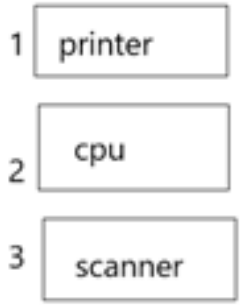
. Process can request for resource in an increasing order only. Set of numbers are assigned to resources.





ELIMINATE CIRCULAR WAIT





- This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed
- Process can request for resource in an increasing order only. Set of numbers are assigned to resources.

Deadlock Avoidance

- The request for any resource will be granted only if the resulting state of the system doesn't cause any deadlock in the system.
- For every resource allocation ,the safe state or unsafe state is checked
- Its done using bankers algorithm

BANKERS ALGORITHM

- Bankers's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.
- Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not.
- the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers. The bank would try to be in safe state always

BANKERS ALGORITHM

Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

1. If the request made by the process is less than equal to max need to that process.
2. If the request made by the process is less than equal to the freely available resource in the system.

Available

indicating the number of available resources of each type.

Allocation :

defines the number of resources of each type currently allocated to each process.

Max

defines the maximum demand of each process in a system

Need (Remaining):

indicates the remaining resource need of each process

BANKERS ALGORITHM:: TOTAL AVAILABLE IS A=10,B=5,C=7

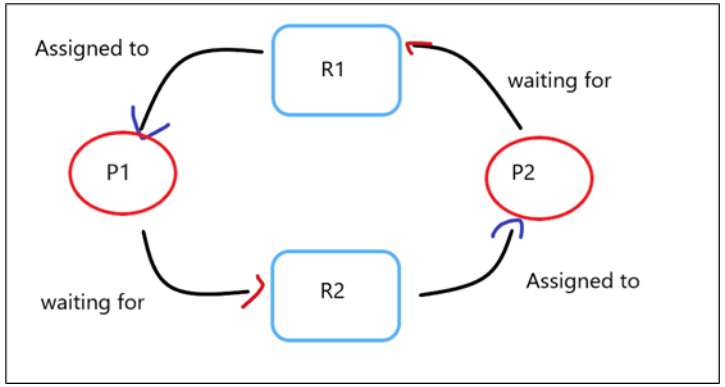
process	Allocated			Max.Need			Available			Need [maxneed – allocation]		
	A	B	C	A	B	C	A	B	C	A	B	C
p1	0	1	0	7	5	3	3	3	2	7	4	3
p2	2	0	0	3	2	2	5	3	2(2)	1	2	2
p3	3	0	2	9	0	2	7	4	3 (4)	6	0	0
p4	2	1	1	4	2	2	7	4	5 (5)	2	1	1
p5	0	0	2	5	3	3	7	5	5 (1)	5	3	1
	P2→P4→P5→P1→P3						10	5	7 (3)			

DEADLOCK DETECTION AND RECOVERY IN OS

- If a system does not employ either a deadlock-prevention or deadlock-avoidance algorithm, then there are chances of occurrence of a deadlock.
- In order to get rid of deadlocks, The OS periodically checks the system for any deadlock.
- In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.
- The OS can detect the deadlocks with the help of Resource allocation graph.

DEADLOCK DETECTION

- If resources have single instance:
- In this case for Deadlock detection can be done by checking for cycle in the Resource Allocation Graph. Presence of cycle in the graph is the sufficient condition for deadlock.

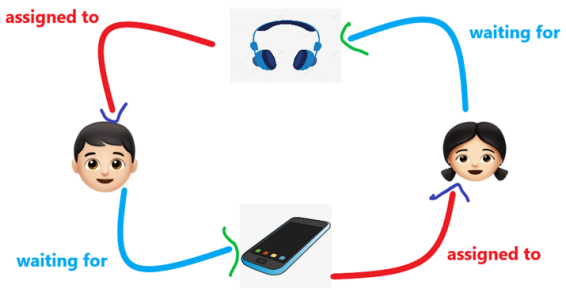


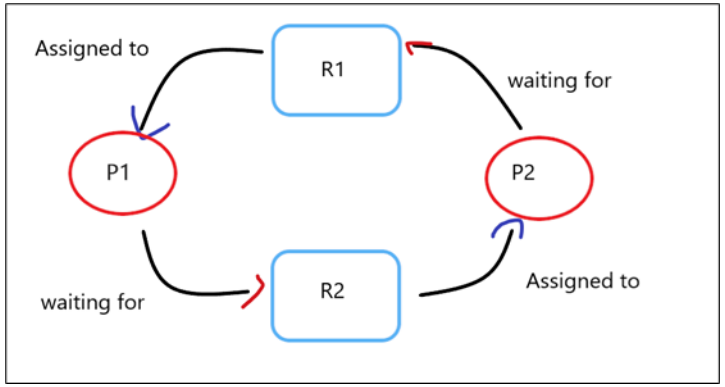
DEADLOCK RECOVERY

Recovery methods

1. **Killing the process:** killing all the process involved in the deadlock. Killing process one by one. After killing each process check for deadlock again keep repeating the process till system recover from deadlock.

2. **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, preempted resources are allocated to other processes so that there is a possibility of recovering the system from deadlock.





STARVATION

- Starvation is the problem that occurs when high priority processes keep executing and low priority processes get blocked for indefinite time
- In starvation resources are continuously utilized by high priority processes. Problem of starvation can be resolved using Aging. In Aging priority of long waiting processes is gradually increased.

Deadlock	Starvation
All processes keep waiting for each other to complete and none get executed	High priority processes keep executing and low priority processes are blocked
Resources are blocked by the processes	Resources are continuously utilized by high priority processes
It can be prevented by avoiding the necessary conditions for deadlock	It can be prevented by Aging

Approaches to Interprocess Communication

The different approaches to implement interprocess communication are given as follows −

- **Pipe**

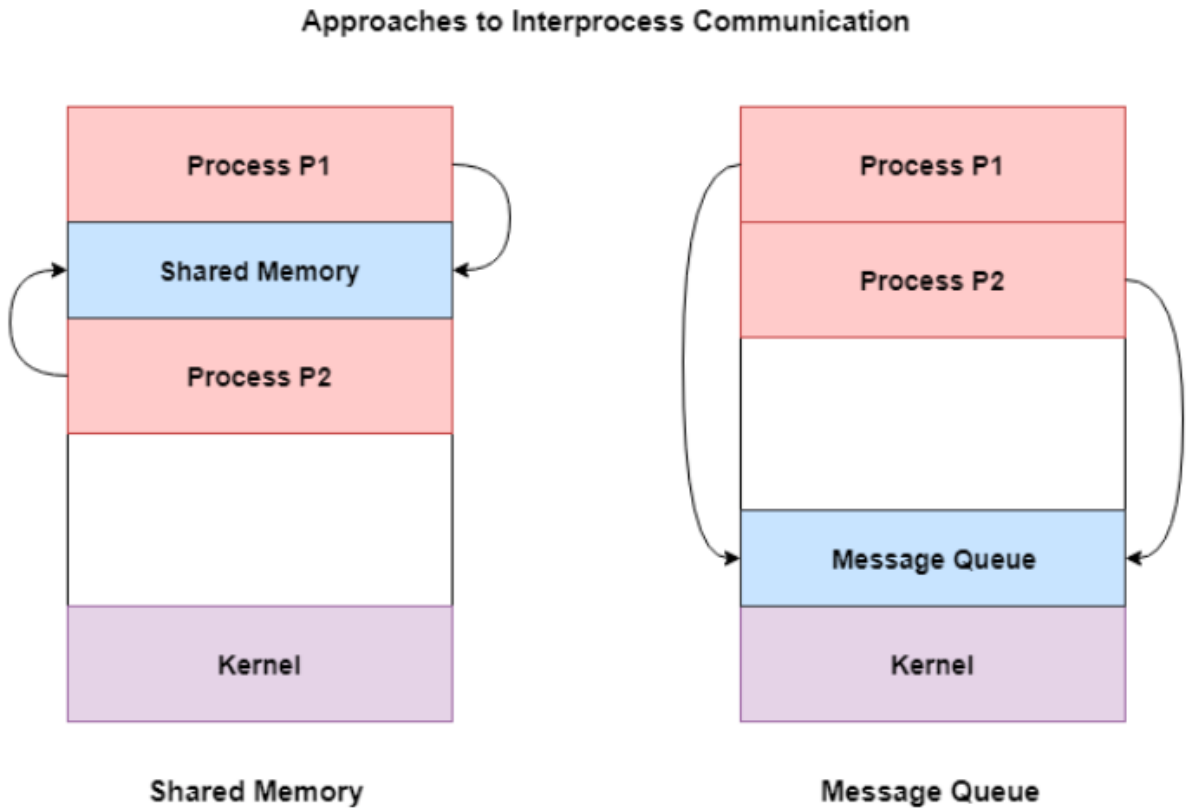
A pipe is a data channel that is unidirectional. Two pipes can be used to create a two-way data channel between two processes. This uses standard input and output methods. Pipes are used in all POSIX systems as well as Windows operating systems.

- **Shared Memory**

Shared memory is the memory that can be simultaneously accessed by multiple processes. This is done so that the processes can communicate with each other. All POSIX systems, as well as Windows operating systems use shared memory.

•Message Queue

Multiple processes can read and write data to the message queue without being connected to each other. Messages are stored in the queue until their recipient retrieves them. Message queues are quite useful for interprocess communication and are used by most operating systems.



Named Pipe or FIFO

- It is an extension to the traditional pipe concept on Unix. A traditional pipe is “unnamed” and lasts only as long as the process.
- A named pipe, however, can last as long as the system is up, beyond the life of the process. It can be deleted if no longer used.
- Usually a named pipe appears as a file and generally processes attach to it for inter-process communication. A FIFO file is a special kind of file on the local storage which allows two or more processes to communicate with each other by reading/writing to/from this file.
- A FIFO special file is entered into the filesystem by calling *mkfifo()* in C. Once we have created a FIFO special file in this way, any process can open it for reading or writing, in the same way as an ordinary file. However, it has to be open at both ends simultaneously before you can proceed to do any input or output operations on it.