DEPARTMENT OF COMPUTER SCIENCE AND ENGG.

SYSTEM PROGRAMMING AND SYSTEM ADMINISTRATION

SEM-6TH

SECTION-B (NOTES)

INTRODUCTION

Macros are used to provide a program generation facility through macro expansion.
Many languages provide build in facilities for writing macros like PL/I, C, Ada AND C++. Assembly languages also provide such facilities.
When a language does not support build in facilities for writing macros what is to be done?
A programmer may achieve an equivalent effect by using generalized preprocessors or software tools like Awk of Unix.

A MACRO

Def: A macro is a unit of specification for program generation through expansion.
A macro consists of
# a name,
# a set of formal parameters and
# a body of code.

The use of a macro name with a set of actual parameters is replaced by some code generated from its body. This is called macro expansion.

Two kinds of expansion can be identified.

CLASSIFICATION OF MACROS

1. **Lexical expansion**:
   Lexical expansion implies replacement of a character string by another character string during program generation. Lexical expansion is to replace occurrences of formal parameters by corresponding actual parameters.

2. **Semantic expansion**:
   Semantic expansion implies generation of instructions tailored to the requirements of a specific usage. Semantic expansion is characterized by the fact that different uses of a macro can lead to codes which differ in the number, sequence and opcodes of instructions.

Eg: Generation of type specific instructions for manipulation of byte and word operands.

The following sequence of instructions is used to increment the value in a memory word by a constant.

1. Move the value from the memory word into a machine register.
2. Increment the value in the machine register.
3. Move the new value into the memory word.

Since the instruction sequence MOVE
ADD
MOVE
may be used a number of times in a program, it is convenient to define a macro named INCR.
Using Lexical expansion the macro call INCR
A,B,AREG can lead to the generation of a MOVE

ADD
MOVE
 instruction sequence to increment A by the value of B using AREG to perform the arithmetic.
Use of Semantic expansion can enable the instruction sequence to be adapted to the types of A and B.

For example an INC instruction could be generated if
A is a byte operand and B has the value „1‟.

HOW DOES MACRO DIFFER FROM SUBROUTINE ?

Macros differ from subroutines in one fundamental respect.
Use of a macro name in the mnemonic field of an assembly statement leads to its
Expansion whereas use of subroutine name in a call instruction leads to its execution. So there is difference in Size & Execution Efficiency.

Macros can be said to trade program size for execution efficiency.

MACRO DEFINITION AND CALL

A macro definition is enclosed between a macro header statement and a macro end statement.
Macro definitions are typically located at the start of a program.
A macro definition consists of.
1 A macro prototype statement
2 One or more model statements
3 Macro preprocessor statements

The macro prototype statement declares the name of a macro and the names and kinds of its parameters.
It has the following syntax <macro name> [< formal parameter spec > [,..]]
Where <macro name> appears in the mnemonic field of an assembly statement and formal parameter spec> is of the form &<parameter name> [<parameter kind>]

MACRO CALL

A macro is called by writing the macro name in the mnemonic field.
Macro call has the following syntax. <macro name> [<actual parameter spec>[,..]]
Where an actual parameter resembles an operand specification in an assembly language statement.

MACRO EXPANSION

Macro call leads to macro expansion.
During macro expansion, the macro call statement is replaced by a sequence of assembly statements. How to differentiate between „the original statements of a program‟ and „the statements resulting from macro expansion‟ ?
Ans: Each expanded statement is marked with a „+‟ preceding its label field.
Two key notions concerning macro expansion are
A. Expansion time control flow
: This determines the order in which model statements are visited during macro expansion.
B. Lexical substitution
: Lexical substitution is used to generate an assembly statement from a model statement.

DEFAULT SPECIFICATION OF PARAMETERS

A default value is a standard assumption in the absence of an explicit specification by the programmer. Default specification of parameters is useful in situations where a parameter has the same value in most calls. When the desired value is different from the default value, the desired value can be specified explicitly in a macro call.
The syntax for formal parameter specification, as follows:
&<parameter name> [<parameter kind> [<default value>]]

MACROS WITH MIXED PARAMETER LISTS

A macro may be defined to use both positional and keyword parameters.
In such a case, all positional parameters must precede all keyword parameters.
example in the macro call
SUMUP A, B, G=20, H=X
A, B are positional parameters while G, H are keyword parameters.
Correspondence between actual and formal parameters is established by applying the rules governing positional and keyword parameters.

OTHER USES OF PARAMETERS

The model statements have used formal parameters only in operand field. However, use of parameters is not restricted to these fields.Formal parameters can also appear in the label and opcode fields of model statements.

Example:

MCRO CALC
&X, &Y, &OP=MULT, &LAB= &LAB
MOVER AREG, &X &OP AREG, &Y
MOVEM AREG, &X MEND
Expansion of the call CALC A, B, LAB=LOOP leads to the following code:
+ LOOP MOVER AREG, A
+ MULT AREG, B
+ MOVEM AREG, A

NESTED MACRO CALLS

A model statement in a macro may constitute a call on another macro.
Such calls are known as nested macro calls. Macro containing the nested call is the outer macro and, Macro called is inner macro.
They follow LIFO rule. Thus, in structure of nested macro calls, expansion of latest macro call (i.e inner macro) is completed first.

ADVANCED MACRO FACILITIES:

Advanced macro facilities are aimed to supporting semantic expansion. Used for:
Performing conditional expansion of model statements and in writing expansion time loops.
These facilities can be grouped into following.
1. Facilities for alteration of flow of control during expansion.
2. Expansion time variables.
3. Attributes of parameters

Unix:

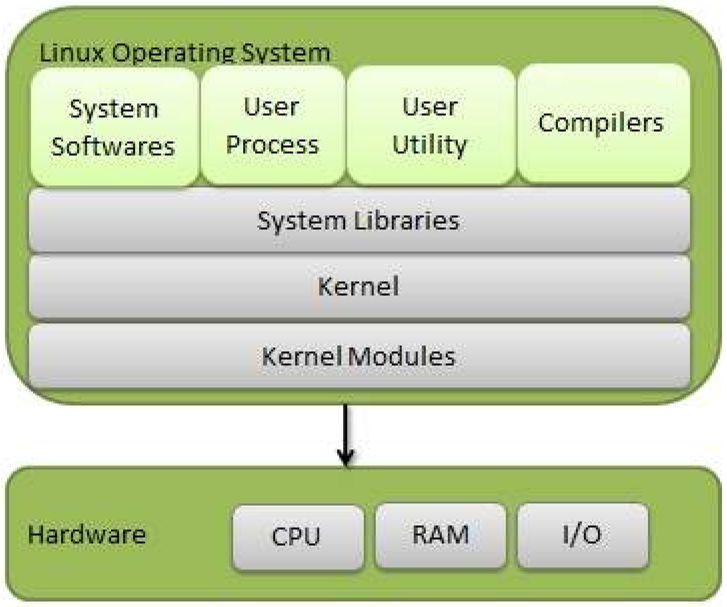
Linux is one of popular version of UNIX operating System. It is open source as its source code is freely available. It is free to use. Linux was designed considering UNIX compatibility. Its functionality list is quite similar to that of UNIX.

Components of Linux System

Linux Operating System has primarily three components

- **Kernel** − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.

- **System Utility** − System Utility programs are responsible to do specialized, individual level tasks.



Kernel Mode vs User Mode

Kernel component code executes in a special privileged mode called **kernel mode** with full access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each processes and provides system services to processes, provides protected access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in **User Mode** which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low level tasks.

Basic Features

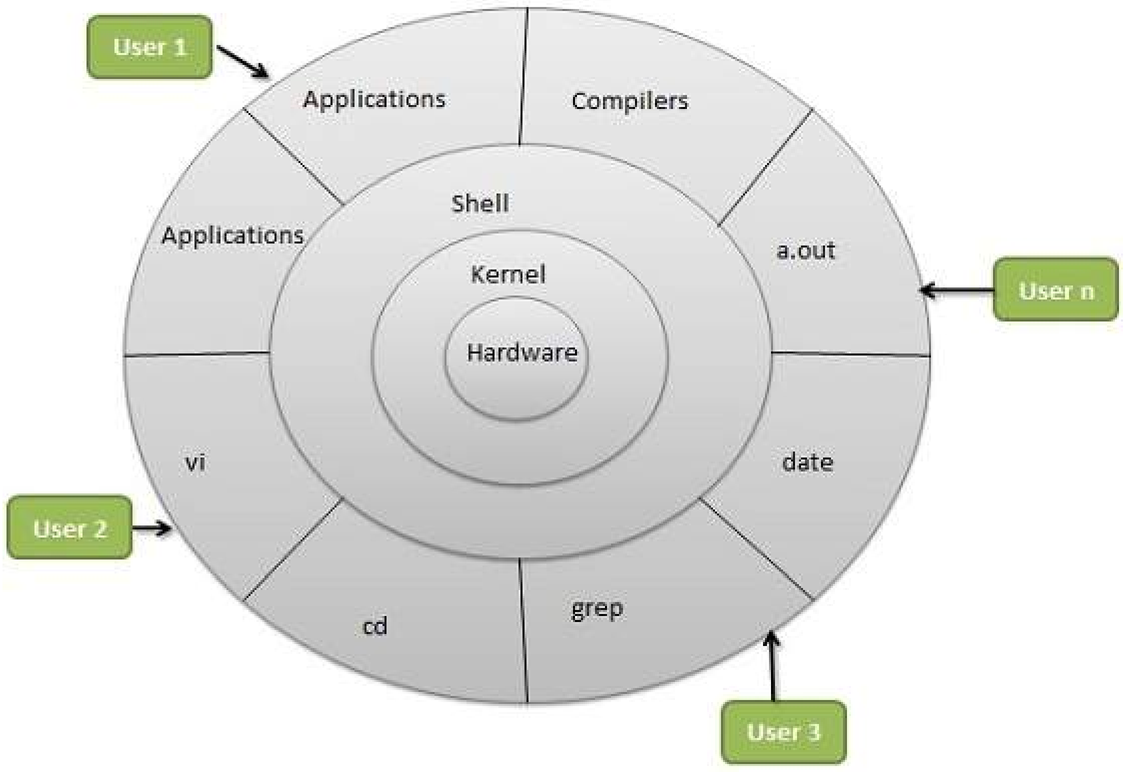
Following are some of the important features of Linux Operating System.

- **Portable** − Portability means software can works on different types of hardware in same way. Linux kernel and application programs supports their installation on any kind of hardware platform.
- **Open Source** − Linux source code is freely available and it is community based development project. Multiple teams work in collaboration to enhance the capability of Linux operating system and it is continuously evolving.

- **Multi-User** − Linux is a multiuser system means multiple users can access system resources like memory/ ram/ application programs at same time.
- **Multiprogramming** − Linux is a multiprogramming system means multiple applications can run at same time.
- **Hierarchical File System** − Linux provides a standard file structure in which system files/ user files are arranged.
- **Shell** − Linux provides a special interpreter program which can be used to execute commands of the operating system. It can be used to do various types of operations, call application programs. etc.
- **Security** − Linux provides user security using authentication features like password protection/ controlled access to specific files/ encryption of data.

Architecture

The following illustration shows the architecture of a Linux system −



The architecture of a Linux System consists of the following layers −

- **Hardware layer** − Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** − It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** − An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.

- **Utilities** − Utility programs that provide the user most of the functionalities of an operating systems.

File

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Structure

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File Type

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

- **Character special files** − data is handled character by character as in case of terminals or printers.
- **Block special files** − data is handled in blocks as in the case of disks and tapes.

File Access Mechanisms

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files −

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

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 on the basis of a particular strategy.

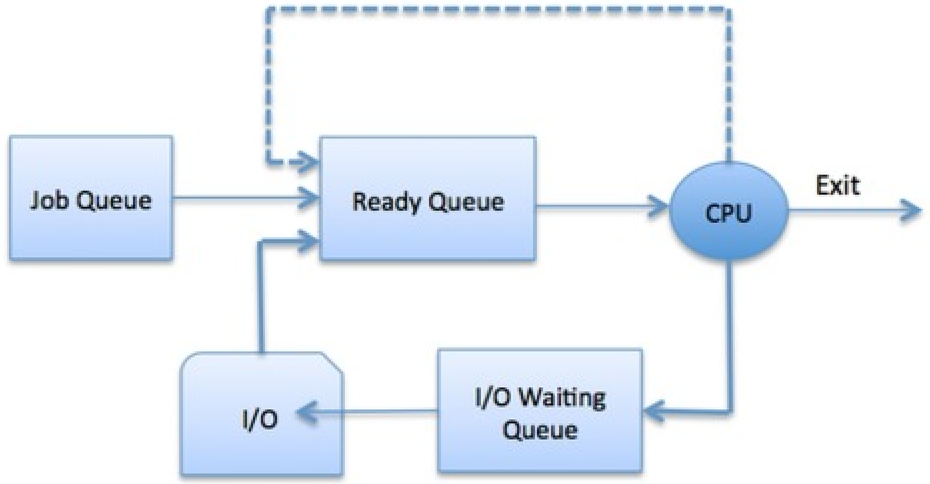
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and 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.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **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.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Two-State Process Model

Two-state process model refers to running and non-running states which are described below −

S.N.	State & Description
1	**Running** When a new process is created, it enters into the system as in the running state.

Not Running

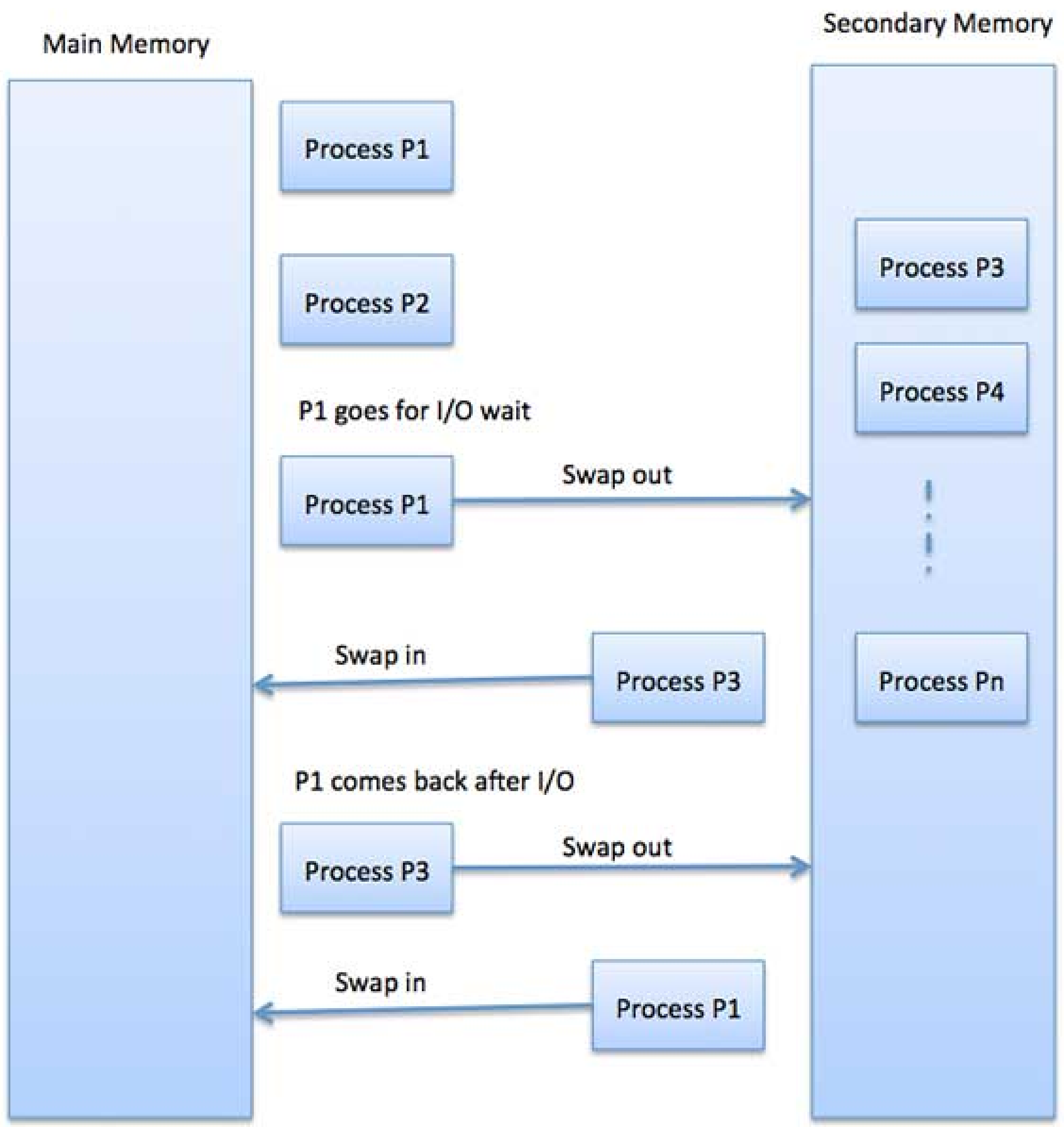
2 Processes that are not running are kept in queue, waiting for their turn to execute. Each entry in the queue is a pointer to a particular process. Queue is implemented by using linked list. Use of dispatcher is as follows. When a process is interrupted, that process is transferred in the waiting queue. If the process has completed or aborted, the process is discarded. In either case, the dispatcher then selects a process from the queue to execute.

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.**also known as a technique for memory compaction**.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

```
2048KB / 1024KB per second
= 2 seconds
= 2000 milliseconds
```

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Memory Allocation

Main memory usually has two partitions −

- **Low Memory** − Operating system resides in this memory.
- **High Memory** − User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

Single-partition allocation

1 In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.

Multiple-partition allocation

2 In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. 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.

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

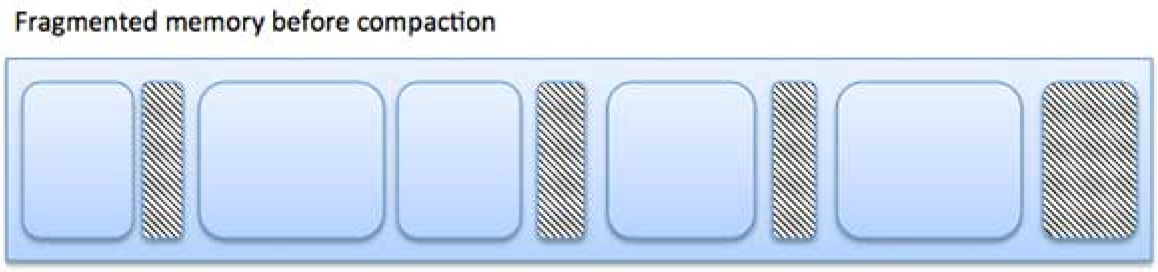
External fragmentation

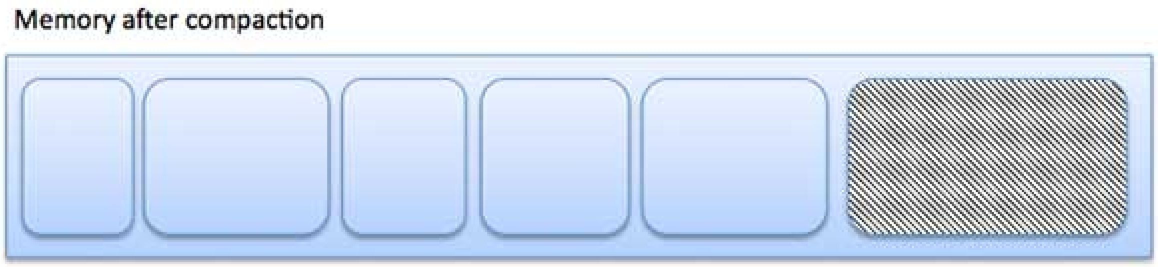
Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −





External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

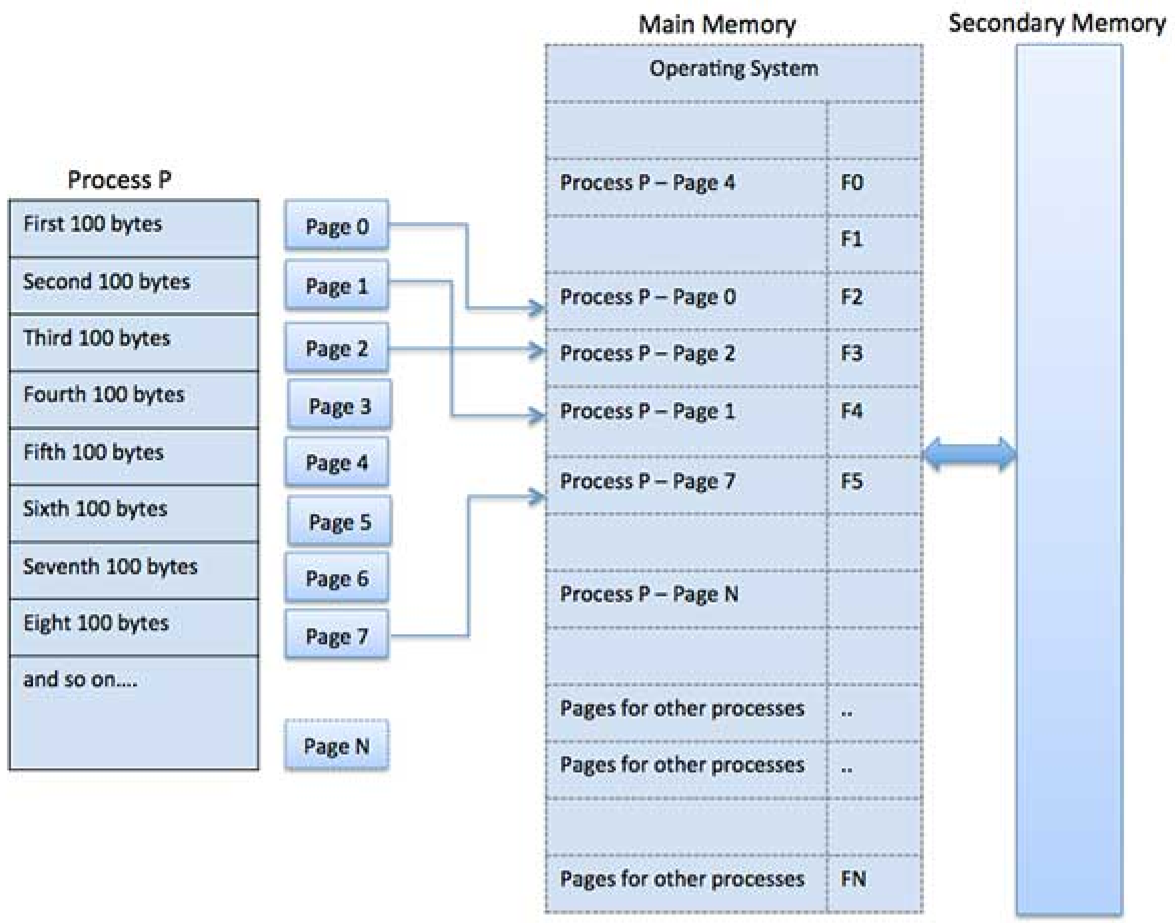
The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

Page address is called **logical address** and represented by **page number** and the **offset**.

```
Logical Address = Page number + page offset
```

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

```
Physical Address = Frame number + page offset
```

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.

Advantages and Disadvantages of Paging

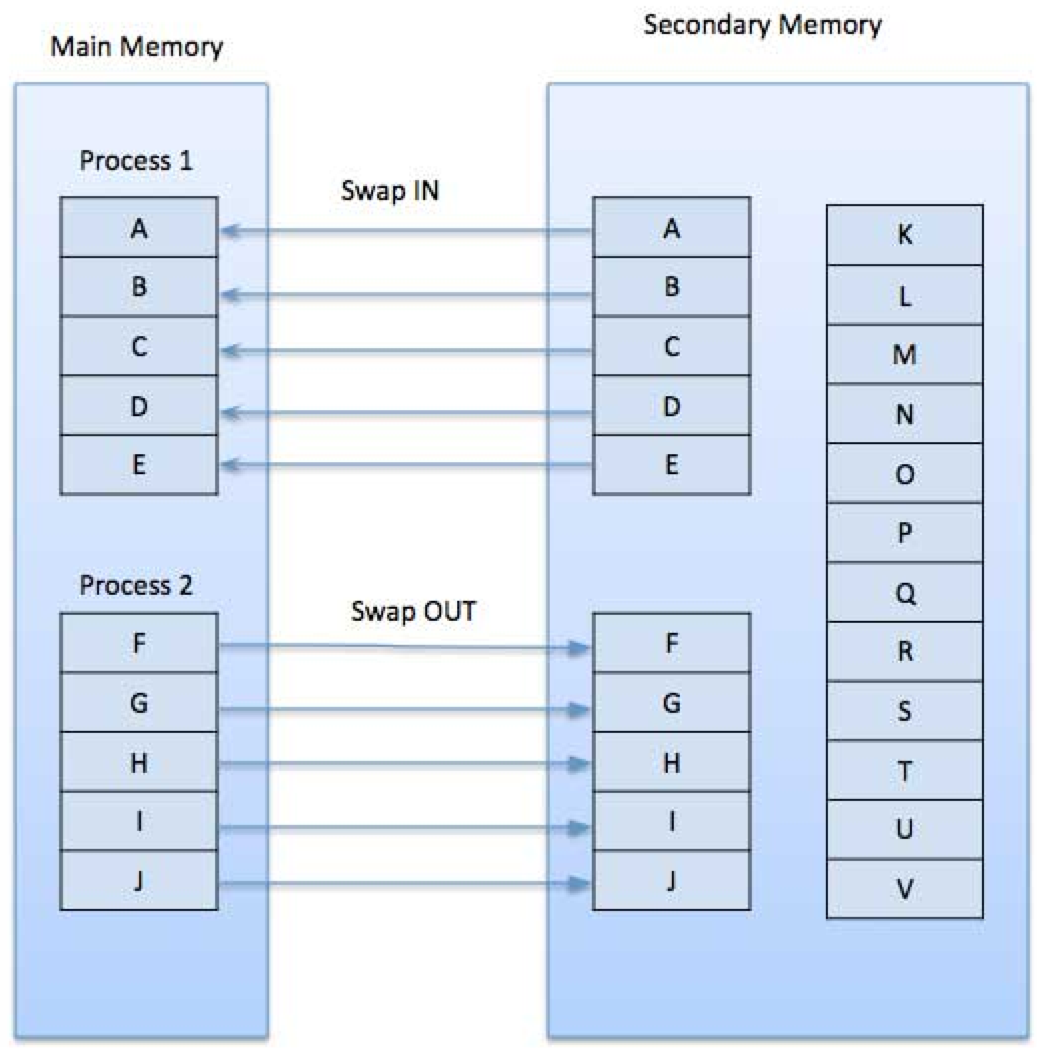
Here is a list of advantages and disadvantages of paging −

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.

- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96

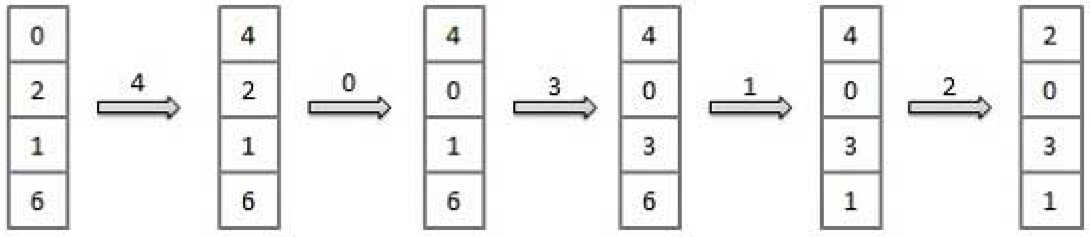
- If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses :x x x x x x x x x



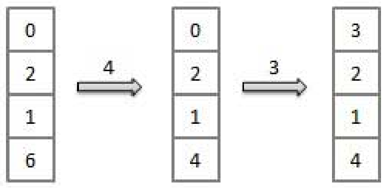
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

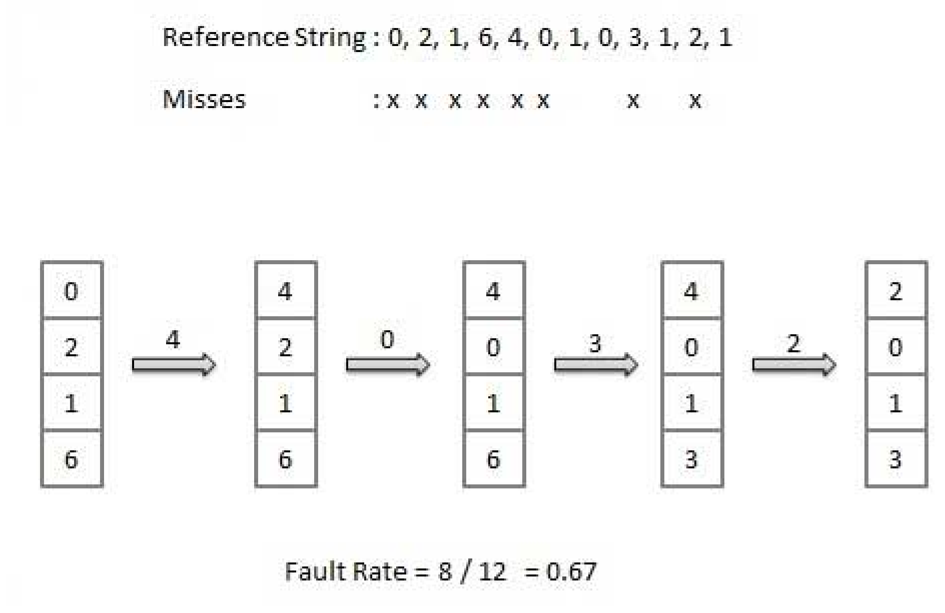
Misses :x x x x x x



Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses : x x x x x x x x
0
2
1
6
4
4
2
1
6
0
4
0
1
6
3
4
0
1
3
2
2
0
1
3
Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Unix File System

The Unix file system (UFS; also called the Berkeley Fast File System, the BSD Fast File System or FFS) is a file system used by many Unix and Unix-like operating systems. It is a distant descendant of the original file system used by Version 7 Unix.

A UFS volume is composed of the following parts:

- A few blocks at the beginning of the partition reserved for boot blocks (which must be initialized separately from the filesystem)
- A superblock, containing a magic number identifying this as a UFS filesystem, and some other vital numbers describing this filesystem's geometry and statistics and behavioral tuning parameters
- A collection of cylinder groups. Each cylinder group has the following components:
    - A backup copy of the superblock
    - A cylinder group header, with statistics, free lists, etc., about this cylinder group, similar to those in the superblock
    - A number of inodes, each containing file attributes
    - A number of data blocks

Inodes are numbered sequentially, starting at 0. Inode 0 is reserved for unallocated directory entries, inode 1 was the inode of the bad block file in historical UNIX versions, followed by the inode for the root directory, which is always inode 2 and the inode for the lost+found directory which is inode 3.

Directory files contain only the list of filenames in the directory and the inode associated with each file. All file metadata is kept in the inode.

- All of the files in the UNIX file system are organized into a multi-leveled hierarchy called a directory tree.
- A family tree is an example of a hierarchical structure that represents how the UNIX file system is organized. The UNIX file system might also be envisioned as an inverted tree or the root system of plant.
- At the very top of the file system is single directory called "root" which is represented by a / (slash). All other files are "descendents" of root.
- The number of levels is largely arbitrary, although most UNIX systems share some organizational similarities. The "standard" UNIX file system is discussed later.

Example:

```
             / (root)
             |
   ---------------------
   |         |         |
  /bin      /usr      /tmp
              |
              |
      -------------------------------------------------------------
      |        |        |                                        |
     /public  /misc    /staff                                /students
      |                  |                                       |
 ------------         -------------------       ----------------------
 |          |         |     |     |     |       |      |      |      |
/software  /doc      /john /mary /bill /carl   /tom  /dick  /mary  /
```

File Types

The UNIX filesystem contains several different types of files:

- Ordinary Files
  - Used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.
  - Always located within/under a directory file
  - Do not contain other files
- Directories
  - Branching points in the hierarchical tree
  - Used to organize groups of files
  - May contain ordinary files, special files or other directories
  - Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
  - All files are descendants of the root directory, (named /) located at the top of the tree.
- Special Files
  - Used to represent a real physical device such as a printer, tape drive or terminal, used for Input/Ouput (I/O) operations
  - Unix considers any device attached to the system to be a file - including your terminal:
    - By default, a command treats your terminal as the standard input file (stdin) from which to read its input
    - Your terminal is also treated as the standard output file (stdout) to which a command's output is sent
    - Stdin and stdout will be discussed in more detail later
  - Two types of I/O: character and block
  - Usually only found under directories named /dev

- Pipes
    - UNIX allows you to link commands together using a pipe. The pipe acts a temporary file which only exists to hold data from one command until it is read by another
    - For example, to pipe the output from one command into another command:

```
who | wc -l
```

This command will tell you how many users are currently logged into the system. The standard output from the who command is a list of all the users currently logged into the system. This output is piped into the wc command as its standard input. Used with the -l option this command counts the numbers of lines in the standard input and displays the result on its standard output - your terminal.

The ping Utility

The **ping** command sends an echo request to a host available on the network. Using this command, you can check if your remote host is responding well or not.

The ping command is useful for the following −

- Tracking and isolating hardware and software problems.
- Determining the status of the network and various foreign hosts.
- Testing, measuring, and managing networks.

Syntax

Following is the simple syntax to use the ping command −

```
$ping hostname or ip-address
```

The above command starts printing a response after every second. To come out of the command, you can terminate it by pressing **CNTRL + C** keys.

Example

Following is an example to check the availability of a host available on the network −

```
$ping google.com
PING google.com (74.125.67.100) 56(84) bytes of data.
64 bytes from 74.125.67.100: icmp_seq = 1 ttl = 54 time = 39.4 ms
64 bytes from 74.125.67.100: icmp_seq = 2 ttl = 54 time = 39.9 ms
64 bytes from 74.125.67.100: icmp_seq = 3 ttl = 54 time = 39.3 ms
64 bytes from 74.125.67.100: icmp_seq = 4 ttl = 54 time = 39.1 ms
64 bytes from 74.125.67.100: icmp_seq = 5 ttl = 54 time = 38.8 ms
--- google.com ping statistics ---
22 packets transmitted, 22 received, 0% packet loss, time 21017ms
rtt min/avg/max/mdev = 38.867/39.334/39.900/0.396 ms
```

The ftp Utility

Here, **ftp** stands for **F**ile **T**ransfer **P**rotocol. This utility helps you upload and download your file from one computer to another computer.

The ftp utility has its own set of Unix-like commands. These commands help you perform tasks such as −

- Connect and login to a remote host.
- Navigate directories.
- List directory contents.
- Put and get files.
- Transfer files as **ascii**, **ebcdic** or **binary**.

Syntax

```
$ftp hostname or ip-address
```

The above command would prompt you for the login ID and the password. Once you are authenticated, you can access the home directory of the login account and you would be able to perform various commands.

The following tables lists out a few important commands −

put filename

Uploads filename from the local machine to the remote machine.

get filename

Downloads filename from the remote machine to the local machine.

mput file list

Uploads more than one file from the local machine to the remote machine.

The telnet Utility

There are times when we are required to connect to a remote Unix machine and work on that machine remotely. **Telnet** is a utility that allows a computer user at one site to make a connection, login and then conduct work on a computer at another site. Once you login using Telnet, you can perform all the activities on your remotely connected machine. The following is an example of Telnet session −

```
C:>telnet amrood.com
Trying...
Connected to amrood.com.
Escape character is '^]'.
```