

INDALA COLLEGE OF ENGINEERING
(Affiliated to Mumbai University & Approved by AICTE, New Delhi)
Kalyan- Murbad Road ,At. Bapsai, Thane, 421103

Operating System Lab Manual

Department of Computer Engineering

IV Semester

2022-23

EXPERIMENT NO:-1

AIM:-To study and implement the internal commands of Linux like ls, chdir, mkdir, chown, chmod, chgrp, ps, etc

THEORY:-

WHAT IS LINUX?

Linux is an Operating System's Kernel. You might have heard of UNIX. Well, Linux is a UNIX clone. But it was actually created by Linus Torvalds from Scratch. Linux is free and open-source, that means that you can simply change anything in Linux and redistribute it in your own name! There are several Linux Distributions, commonly called "distros". A few of them are:

- Ubuntu Linux
- Red Hat Enterprise Linux
- Linux Mint
- Debian
- Fedora

1. Command - ls

Name	list command
Purpose	functions in the Linux terminal to show all of the major directories filed under a given file system, will also show the user all of the folders stored in the specified folder. Options: a- used to list all the files including the hidden files. c - list all the files columnwise. d - list all the directories. m - list the files separated by commas. p - list files include „/" to all the directories. r - list the files in reverse alphabetical order. f - list the files based on the list modification date. x - list in column wise sorted order.
Syntax	\$ ls – options <arguments>
Example	ls /applications

OUTPUT:

```
/root# ls -a
.          .bash_history  .bash_profile  test.lua
..         .bash_logout  .shrc          tests
/root# _
```

2. Command – chmod

Name	change mode command
Purpose	Permissions can be changed by owner of the file Symbolic modes User(u) - the owner of the file Group(g) - users who are members of the file's group Others(o) - users who are not the owner of the file or members of a group All(a) - three of the above; is the same as ugo Read(r) - read a file or list a directory's contents Write(w) - write to a file or directory Execute(x) - execute a file or recurse a directory tree
Syntax	\$ chmod ug+x file
Example	\$ chmod 400 sample.txt

OUTPUT:

```
[ 1.844226] serio: i8042 KBD port at 0x60,0x64 irq 1
[ 1.844226] serio: i8042 AUX port at 0x60,0x64 irq 12
[ 1.853890] mice: PS/2 mouse device common for all mice
[ 1.871274] rtc_cmos rtc_cmos: rtc core: registered rtc_cmos as rtc0
[ 1.879418] rtc0: alarms up to one day, 114 bytes nvram
[ 1.893857] input: AT Translated Set 2 keyboard as /devices/platform/i8042/ser
rio0/input/input0
[ 1.920169] RAMDISK: ext2 filesystem found at block 0
[ 1.920169] RAMDISK: Loading 4719KiB [1 disk] into ram disk... done.
[ 3.569789] VFS: Mounted root (ext2 filesystem) on device 1:0.

/root# chmod 400sample.txt
BusyBox v1.21.1 (2013-10-16 20:06:36 CEST) multi-call binary.

Usage: chmod [-Rcvf] MODE[,MODE]... FILE...

Each MODE is one or more of the letters ugoa, one of the
symbols += and one or more of the letters rwxst

    -R    Recurse
    -c    List changed files
    -v    List all files
    -f    Hide errors

/root#
```

3. Command - Sort

Name	sort command
Purpose	sort command can be used to get sorted content
Syntax	\$ sort file
Example	<p>Let's say you have a file, data.txt, which contains the following ASCII text: apples</p> <p>oranges</p> <p>pears</p> <p>kiwis</p> <p>bananas</p> <p>To sort the lines in this file alphabetically, use the following command: \$ sort data.txt</p>

OUTPUT:

```

root@/root:~# sort data.txt
apples
bananas
kiwis
oranges
pears
root@/root:~# _
```

4. Command - chdir

Name	chdir command- change directory
Purpose	chdir is the system function for changing the current working directory.
Syntax	\$ chdir name of the directory
Example	chdir /home/user/www

OUTPUT:

```
[ 1.102952] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
[ 1.374573] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550
[ 1.510097] brd: module loaded
[ 1.584930] loop: module loaded
[ 1.584930] Uniform Multi-Platform E-IDE driver
[ 1.584930] ide-gd driver 1.18
[ 1.601410] ide-cd driver 5.00
[ 1.608799] serio: i8042 KBD port at 0x60,0x64 irq 1
[ 1.608799] serio: i8042 AUX port at 0x60,0x64 irq 12
[ 1.615738] mice: PS/2 mouse device common for all mice
[ 1.642262] input: AT Translated Set 2 keyboard as /devices/platform/input/input0
[ 1.650022] rtc_cmos rtc_cmos: rtc core: registered rtc_cmos
[ 1.650022] rtc0: alarms up to one day, 114 bytes nvram
[ 1.676370] RAMDISK: ext2 filesystem found at block 0
[ 1.676370] RAMDISK: Loading 4719KiB [1 disk] into ram disk.
[ 3.315464] VFS: Mounted root (ext2 filesystem) on device 1:0.

/root# mkdir home
/root# chdir home
home# mkdir user
home# chdir user
user# mkdir www
user# chdir www
www#
```

5. Command - mkdir

Name	mkdir command –make directory
Purpose	Create the DIRECTORY(ies), if they do not already exist.
Syntax	\$ mkdir directory_name
Example	\$ mkdir mydir \$ls

OUTPUT:

```
found)
[ 0.980290] io scheduler noop registered (default)
[ 1.228832] Non-volatile memory driver v1.3
[ 1.228326] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
[ 1.508506] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550
[ 1.658431] brd: module loaded
[ 1.730674] loop: module loaded
[ 1.730674] Uniform Multi-Platform E-IDE driver
[ 1.730674] ide-gd driver 1.18
[ 1.747453] ide-cd driver 5.00
[ 1.753368] serio: i8042 KBD port at 0x60,0x64 irq 1
[ 1.753368] serio: i8042 AUX port at 0x60,0x64 irq 12
[ 1.759277] mice: PS/2 mouse device common for all mice
[ 1.779596] rtc cmos rtc cmos: rtc core: registered rtc cmos as rtc0
[ 1.779596] rtc0: alarms up to one day, 114 bytes nvram
[ 1.796141] input: AT Translated Set 2 keyboard as /devices/platform/input/input0
[ 1.819294] RAMDISK: ext2 filesystem found at block 0
[ 1.819294] RAMDISK: Loading 4719KiB [1 disk] into ram disk...
[ 3.368620] VFS: Mounted root (ext2 filesystem) on device 1:0.

/root# mkdir mydir
/root# ls
mydir  test.lua  tests
/root# _
```

6. Command - chown

Name	chown
Purpose	To change owner, change the user and/or group ownership of each given File to a new Owner.
Syntax	chown [options] new_owner object(s)
Example	The following would transfer the ownership of a file named <i>file1</i> and a directory named <i>dir1</i> to a new owner named <i>alice</i> : chown alice file1 dir1

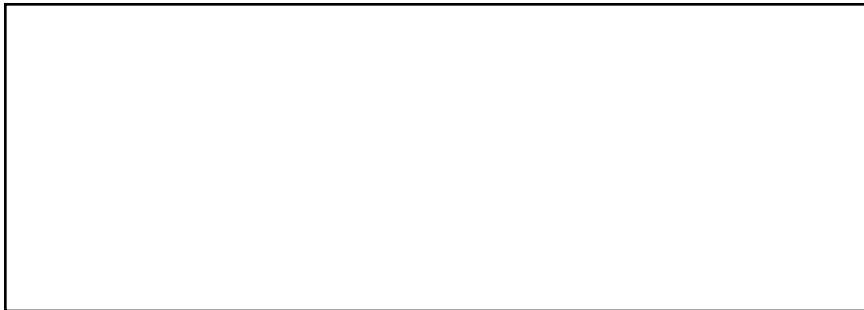
OUTPUT:

```
root@kali:~# ls -l file1.txt
-rw-r--r-- 1 root root 12 Feb  4 12:04 file1.txt
root@kali:~# chown master file1.txt
root@kali:~# ls -l file1.txt
-rw-r--r-- 1 master root 12 Feb  4 12:04 file1.txt
root@kali:~# █
```

7. Command - chgrp

Name	chgrp
Purpose	'chgrp' command changes the group ownership of each given File to Group (which can be either a group name or a numeric group id) or to match the same group as an existing reference file.
Syntax	chgrp [OPTION]... GROUP FILE...
Example	To Make oracleadmin the owner of the database directory \$ chgrp oracleadmin /usr/database

OUTPUT:



8. Command - ps

Name	ps
Purpose	displays information about a selection of the active processes.
Syntax	ps aux

Example	\$ ps aux
----------------	-----------

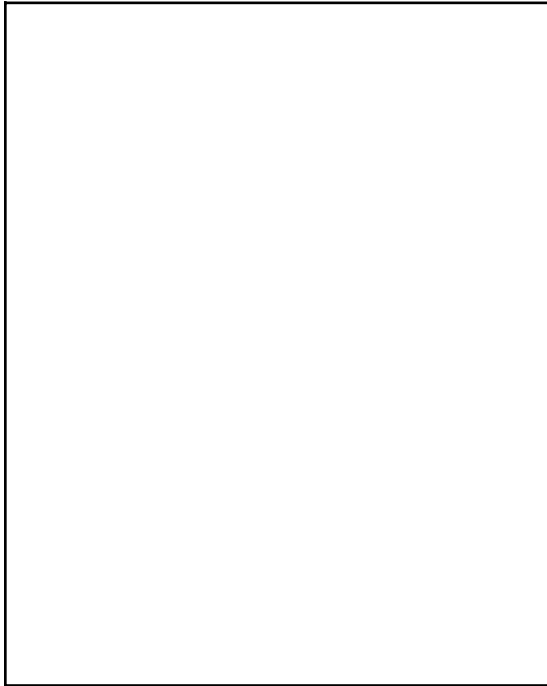
OUTPUT:

```
[ 0.652543] platform rtc_cmos: registered platform RTC device (
found)
[ 0.895026] io scheduler noop registered (default)
[ 1.112863] Non-volatile memory driver v1.3
[ 1.127542] Serial: 8250/16550 driver, 4 ports, IRQ sharing ena
[ 1.401601] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550
[ 1.544531] brd: module loaded
[ 1.623892] loop: module loaded
[ 1.623892] Uniform Multi-Platform E-IDE driver
[ 1.623892] ide-gd driver 1.18
[ 1.641308] ide-cd driver 5.00
[ 1.648462] serio: i8042 KBD port at 0x60,0x64 irq 1
[ 1.648462] serio: i8042 AUX port at 0x60,0x64 irq 12
[ 1.659082] mice: PS/2 mouse device common for all mice
[ 1.675730] rtc_cmos rtc_cmos: rtc core: registered rtc_cmos as
[ 1.675730] rtc0: alarms up to one day, 114 bytes nvram
[ 1.682875] input: AT Translated Set 2 keyboard as /devices/pla
rio0/input/input0
[ 1.703024] RAMDISK: ext2 filesystem found at block 0
[ 1.703024] RAMDISK: Loading 4719KIB [1 disk] into ram disk...
[ 3.323219] VFS: Mounted root (ext2 filesystem) on device 1:0.

/root# ps aux
PID  USER  COMMAND
/root#
```

9. Command – man

Name	man
Purpose	It is used to show the manual of the inputted command.
Syntax	\$ man <command_name>
Example	The inputting command will show the manual or all relevant information for the change directory command. \$ man cd

OUTPUT:**10. Command – rm**

Name	rm - remove file
Purpose	It is used to remove files from your Linux OS.
Syntax	rm filename.txt
Example	\$ rm testfile.txt

OUTPUT:

```
/root% rm data.txt
/root% ls
test.lua  tests
/root% _
```

CONCLUSION: - Hence we have studied and implemented internal commands of Linux successfully.

EXPERIMENT NO:-2

AIM: - To study and implement the shell scripts.

THEORY: -

SHELL SCRIPTS:

Shell scripts are short programs that are written in a shell programming language and interpreted by a shell process. They are extremely useful for automating tasks on Linux and other Unix-like operating systems.

A shell is a program that provides the traditional, text-only user interface for Unix like operating systems. Its primary function is to read commands (i.e., instructions) that are typed into a console (i.e., an all-text display mode) or terminal window (i.e., all-text mode window) and then execute (i.e., run) them. The default shell on Linux is the very commonly used and highly versatile bash.

Steps to write and execute a script:

- Open the terminal. Go to the directory where you want to create your script. • Create a file with **.sh** extension.
- Write the script in the file using an editor.
- Make the script executable with command **chmod +x <fileName.sh>**. •

Run the script using **./<fileName.sh>**.

Example: Print HELLO WORLD using shell script.

```
#!/bin/bash  
echo "HELLO WORLD"
```

SHELL SCRIPTS EXAMPLE:

1. Display top 10 processes in descending order

The following command will show the list of top processes ordered by RAM and CPU use in descendant form (remove the pipeline and head if you want to see the full list)

```
# ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head
```

The process list shows all the processes with various process specific details in separate columns. Some of the column names are pretty self explanatory.

PID - Process ID

USER - The system user account running the process.

%CPU - CPU usage by the process.

%MEM - Memory usage by the process

COMMAND - The command (executable file) of the process

OUTPUT:

```
/root% ps -eo pid,pid,cmd,%mem,%cpu --sort=-%mem | head
ps: unrecognized option '--sort=-%mem'
BusyBox v1.21.1 (2013-10-16 20:06:36 CEST) multi-call binary

Usage: ps [-o COL1,COL2=HEADER]

Show list of processes

-o COL1,COL2=HEADER      Select columns for display
```

2. Display processes with highest memory usage.

To find the process consuming the most CPU or memory, simply sort the list. Press M key (yes, in capital, not small) to sort the process list by memory usage. Processes using the most memory are shown first and rest in order.

Here are other options to sort by CPU usage, Process ID and Running Time -

Press 'P' – to sort the process list by cpu usage.

Press 'N' - to sort the list by process id

Press 'T' - to sort by the running time.

OUTPUT:

```
 1 ?      Ss      0:50 /sbin/init
 2 ?      S       0:01 [kthreadd]
 3 ?      S       0:00 [ksoftirqd/0]
 5 ?      S<      0:00 [kworker/0:0H]
 7 ?      S       0:13 [rcu_sched]
 8 ?      S       0:00 [rcu_bh]
 9 ?      S       0:00 [migration/0]
10 ?      S<      0:00 [lru-add-drain]
11 ?      S       0:00 [watchdog/0]
12 ?      S       0:00 [cpuhp/0]
13 ?      S       0:00 [cpuhp/1]
14 ?      S       0:00 [watchdog/1]
15 ?      S       0:00 [migration/1]
16 ?      S       0:00 [ksoftirqd/1]
18 ?      S<      0:00 [kworker/1:0H]
```

3. Display current logged in user and logname

```
echo "Hi, $USER! Let us be friends."  
echo "Hello, $LOGNAME! "
```

OUTPUT:

```
[ 1.047632] io scheduler noop registered (default)  
[ 1.317390] Non-volatile memory driver v1.3  
[ 1.331655] Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled  
[ 1.612055] serial8250: ttyS0 at I/O 0x3f8 (irq = 4) is a 16550A  
[ 1.820456] brd: module loaded  
[ 1.909239] loop: module loaded  
[ 1.909239] Uniform Multi-Platform E-IDE driver  
[ 1.917828] ide-gd driver 1.18  
[ 1.917828] ide-cd driver 5.00  
[ 1.923688] serio: i8042 KBD port at 0x60,0x64 irq 1  
[ 1.923688] serio: i8042 AUX port at 0x60,0x64 irq 12  
[ 1.942741] mice: PS/2 mouse device common for all mice  
[ 1.955425] rtc_cmos rtc_cmos: rtc core: registered rtc_cmos as rtc0  
[ 1.955425] rtc0: alarms up to one day, 114 bytes nvram  
[ 1.966304] input: AT Translated Set 2 keyboard as /devices/platform/i8042/ser  
io0/input/input0  
[ 2.008521] RAMDISK: ext2 filesystem found at block 0  
[ 2.008521] RAMDISK: Loading 4713KiB [1 disk] into ram disk... done.  
[ 3.684572] VFS: Mounted root (ext2 filesystem) on device 1:0.  
  
/root# echo "Hi,$USER! let us be friends."  
Hi,root! let us be friends.  
/root# echo "Hi,$LOGNAME!"  
Hi,!
```

4. Display OS Version , release number , kernel version

```
$ uname -a (Print all Information)  
$ uname -r (Print the kernel name)  
$ cat /proc/version  
$ cat /etc/issue  
$ cat /etc/redhat-release
```

OUTPUT

```
vivek@rhel7-nixcraft ~]$ cat /etc/os-release  
NAME="Red Hat Enterprise Linux Server"  
VERSION="7.5 (Maipo)"  
ID="rhel"  
ID_LIKE="fedora"  
VARIANT="Server"  
VARIANT_ID="server"  
VERSION_ID="7.5"  
PRETTY_NAME="Red Hat Enterprise Linux"  
ANSI_COLOR="0:31"  
CPE_NAME="cpe:/o:redhat:enterprise_linux:7.5:GA:server"  
BOME_URL="https://www.redhat.com/"  
BUG_REPORT_URL="https://bugzilla.redhat.com/"  
  
REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 7"  
REDHAT_BUGZILLA_PRODUCT_VERSION=7.5  
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"  
REDHAT_SUPPORT_PRODUCT_VERSION="7.5"  
vivek@rhel7-nixcraft ~]$  
vivek@rhel7-nixcraft ~]$ hostnamectl  
  Static hostname: rhel7-nixcraft  
        Icon name: computer-vm  
        Chassis: vm  
        Machine ID: bfce983b886a54432b8b992b24603144d  
        Boot ID: 85ed74bb8e4d4a19aeb2e8048a76fbc2  
        Virtualization: kvm  
        Operating System: Red Hat Enterprise Linux  
        CPE OS Name: cpe:/o:redhat:enterprise_linux:7.5:GA:server  
        Kernel: Linux 3.10.0-862.14.4.el7.x86_64  
        Architecture: x86_64  
vivek@rhel7-nixcraft ~]$
```

CONCLUSION:- Thus we have studied and implemented shell script programs successfully

EXPERIMENT NO:-3

AIM:- To study and implement the following system calls: open, read, write, close, getpid, setpid, getuid, getgid, getegid, geteuid.

THEORY: -

SYSTEM CALL:

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.

When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a context switch.

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

Generally, 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.

In a typical UNIX system, there are around 300 system calls.

1. Opening a File: open()

Description: “open()” allows you to open or create a file for reading and/or writing.

Syntax: `int open(char* fileName, int mode[, int permissions])`

Where

fileName : an absolute or relative pathname,

mode : a bitwise or'ing of a read/write flag together with zero or more miscellaneous

flags. permission : a number that encodes the value of the file's permission flags.

```
// C program to illustrate
// open system call
#include<stdio.h>
#include<fcntl.h>
#include<errno.h>
extern int errno;
int main()
{
    // if file does not have in directory
    // then file foo.txt is created.
```

```
int fd = open("foo.txt", O_RDONLY | O_CREAT);
```

DMCE COMPUTER DEPT OSL Page 12

```
printf("fd = %d\n", fd);  
if (fd == -1)  
{  
    // print which type of error have in a code  
    printf("Error Number % d\n", errno);  
    // print program detail "Success or failure"  
    perror("Program");  
}  
return 0;  
}
```

OUTPUT –

2. Reading From a File : read()

Description: To read bytes from a file, it uses the “read()” system call.

Syntax: ssize_t read(int fd, void* buf, size_t count)

Here “read()” copies count bytes from the file referenced by the file descriptor fd into the buffer buf.

```
// C program to illustrate
// read system Call
#include<stdio.h>
#include <fcntl.h>
#include<stdlib.h>
#include <unistd.h>
int main()
{
    int fd, sz;
    char *c = (char *) calloc(100, sizeof(char));
    fd = open("foo.txt", O_RDONLY);
    if (fd < 0) { perror("r1"); exit(1); }
    sz = read(fd, c, 10);
    printf("called read(% d, c, 10). returned that"
        " %d bytes were read.\n", fd, sz);
    c[sz] = '\0';
    printf("Those bytes are as follows: %s", c);
}
```

OUTPUT –

3. Writing to a File: write()

Description: To write bytes to a file, it uses the “write()” system call,

Syntax: ssize_t write(int fd, void* buf, size_t count)

Here “write()” copies count bytes from a buffer buf to the file referenced by the file descriptor fd.

```
// C program to illustrate
// write system Call
#include<stdio.h>
```



```

#include <fcntl.h>
#include<stdlib.h>
#include <unistd.h>
#include <string.h>
// I/O system Calls
int main (void)
{
    int fd[2];
    char buf1[12] = "hello world";
    char buf2[12];
    // assume foobar.txt is already created
    fd[0] = open("foobar.txt", O_RDWR);
    fd[1] = open("foobar.txt", O_RDWR);
    write(fd[0], buf1, strlen(buf1));
    write(1, buf2, read(fd[1], buf2, 12));
    close(fd[0]);
    close(fd[1]);
    return 0;
}

```

OUTPUT –

4. Closing a File: “close()”

Description: uses the “close()” system call to free the file descriptor of the input.

Syntax: int close(int fd)

Here “close()” frees the file descriptor fd.

- ✓ If fd is the last file descriptor associated with a particular open file, the kernel resources associated with the file are deallocated.
- ✓ If successful, “close()” returns a value of 0; otherwise, it returns a value of -1.

// C program to illustrate close system Call

```

#include<stdio.h>
#include<stdlib.h>
#include <fcntl.h>
#include <unistd.h>
int main()
{
    // assume that foo.txt is already created
    int fd1 = open("foo.txt", O_RDONLY, 0);
    close(fd1);
}

```

```
// assume that baz.txt is already created
int fd2 = open("baz.txt", O_RDONLY, 0);

printf("fd2 = % d\n", fd2);
exit(0);
}
```

OUTPUT –

5. Process management - getpid() & getppid()

Description: A process may obtain its own process ID and parent process ID numbers by using the “getpid()” and “getppid()” system calls, respectively.

Syntax: pid_t getpid(void)
pid_t getppid(void)

Here “getpid()” and “getppid()” return a process’ ID number and parent process’ ID number, respectively.

The parent process ID number of PID 1 (i.e., “init”) is 1.

OUTPUT -

6. Accessing User and Group IDs

Description: The system calls that allow you to read a process real and effective IDs

Syntax: uid_t getuid()
uid_t geteuid()
gid_t getgid()
gid_t getegid()

Here,

“getuid()” and “geteuid()” return the calling process’ real and effective user IDs, respectively.

“getgid()” and “getegid()” return the calling process’ real and effective group IDs, respectively.

The ID numbers correspond to the user and group IDs listed in “/etc/passwd” and “/etc/group” files.

These calls always succeed.

OUTPUT –

```
root@HOST:~/Desktop# gcc DTS.c -o DTS
root@HOST:~/Desktop# ./DTS
root@HOST:~/Desktop#
UserID:0
UserID:0
GroupID:0
GroupID:0█
```

CONCLUSION:- Thus we have studied and explored the commands of system calls

EXPERIMENT NO:-4

AIM: - To Implement basic commands of Linux like ls, cp, mv and others using kernel APIs.

THEORY:-

1. Command – stat

Name	stat
Purpose	To check the status of a file. This provides more detailed information about a file than 'ls -l' output.
Syntax	\$ stat usrcopy
Example	File: `usrcopy' Size: 491 Blocks: 8 IO Block: 4096 regular file Device: 808h/2056d Inode: 149452 Links: 1 Access: (0644/-rw-r--r--) Uid: (1000/ raghu) Gid: (1000/ raghu) Access: 2012-07-06 16:07:06.413522009 +0530 Modify: 2012-07-06 16:02:30.204152386 +0530 Change: 2012-07-06 16:17:18.992559654 +0530

OUTPUT:

--

2. Command – cal

Name	cal
Purpose	Displays the calendar of the current month.
Syntax	\$ cal

Example	<pre>\$ cal July 2012 Su Mo Tu We Th Fr Sa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31</pre>
----------------	--

OUTPUT:

3. Command – VI editor

Name	VI editor
Purpose	VI stands for Visual editor; another text editor in Linux. This is a standard editor in many Linux/Unix environments.
Syntax	<code>\$ vi filename</code>
Example	<code>\$ vi hello.txt</code>

OUTPUT:

4. Command – mv

Name	mv - move
-------------	-----------

DMCE COMPUTER DEPT OSL Page 18

Purpose	Move files or directories. The 'mv' command works like 'cp' command, except that the original file is removed. But, the mv command can be used to rename the files (or directories).
Syntax	\$ mv source destination
Example	mv myfile.txt myfiles Move the file myfile.txt into the directory myfiles . If myfiles is a file, it will be overwritten. If the file is marked as read-only, but you own the file, you will be prompted before overwriting it.

OUTPUT:

--

5. Command copy

Name	cp - copy
Purpose	Copy files and directories. If the source is a file, and the destination (file) name does not exist, then source is copied with new name i.e. with the name provided as the destination.
Syntax	\$ cp source destination

Example	<pre>\$ cp usrlisting listing_copy.txt \$ ls -l total 12 drwxr-xr-x 2 raghu raghu 4096 2012-07-06 14:09 example -rw-r--r-- 1 raghu raghu 0 2012-07-06 14:20 file1 -rw-r--r-- 1 raghu raghu 0 2012-07-06 14:20 file2 -rw-r--r-- 1 raghu raghu 0 2012-07-06 14:20 file3 -rw-r--r-- 1 raghu raghu 491 2012-07-06 16:02 listing_copy.txt -rw-r--r-- 1 raghu raghu 491 2012-07-06 14:23 usrlisting</pre>
----------------	---

OUTPUT:



6. Command – date

Name	cp - copy
Purpose	Displays current time and date. If you are interested only in time, you can use 'date +%T' (in hh:mm:ss):
Syntax	\$ date
Example	<pre>\$ date Fri Jul 6 01:07:09 IST 2012 \$ date +%T 01:13:14</pre>

OUTPUT:



7. Command – whoami

Name	whoami
Purpose	This command reveals the user who is currently logged in.
Syntax	\$ whoami
Example	\$ whoami raghu

OUTPUT:

--

8. Command – pwd

Name	pwd
Purpose	‘pwd’ command prints the absolute path to current working directory.
Syntax	\$ pwd
Example	\$ pwd /home/raghu

OUTPUT:

--

9. Command – touch

Name	touch
Purpose	For creating an empty file, use the touch command.
Syntax	\$ touch filename

Example	<pre>\$ touch file1 file2 file3 \$ ls -l total 4 drwxr-xr-x 2 raghu raghu 4096 2012-07-06 14:09 example -rw-r--r-- 1 raghu raghu 0 2012-07-06 14:20 file1 -rw-r--r-- 1 raghu raghu 0 2012-07-06 14:20 file2 -rw-r--r-- 1 raghu raghu 0 2012-07-06 14:20 file3</pre>
----------------	---

OUTPUT:

10. Command – wc

Name	Word count
Purpose	wc , or "word count," prints a count of <u>newlines</u> , words, and <u>bytes</u> for each input <u>file</u> .
Syntax	\$ wc filename
Example	<pre>wc myfile.txt 5 13 57 myfile.txt</pre> <p>Where 5 is the number of lines, 13 is the number of words, and 57 is the number of characters.</p>

OUTPUT:

CONCLUSION:- Thus we have studied and implemented basic commands of Linux like ls, cp, mv and others using kernel APIs.

EXPERIMENT NO:-5

AIM: - Write a program to implement CPU Scheduling algorithms like FCFS & SJF.

THEORY:-

1. FIRST-COME, FIRST-SERVE SCHEDULING (FCFS):

In this, which process enter the ready queue first is served first. The OS maintains DS that is ready queue. It is the simplest CPU scheduling algorithm. If a process request the CPU then it is loaded into the ready queue, which process is the head of the ready queue, connect the CPU to that process.

Algorithm for FCFS scheduling:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Set the waiting of the first process as '0' and its burst time as its turn around time

Step 5: for each process in the Ready Q calculate

(c) Waiting time for process(n)=waiting time of process (n-1) + Bursttime of process(n-1)

(d) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n) **Step 6:** Calculate

(e) Average waiting time = Total waiting Time / Number of process

(f) Average Turnaround time = Total Turnaround Time / Number of process

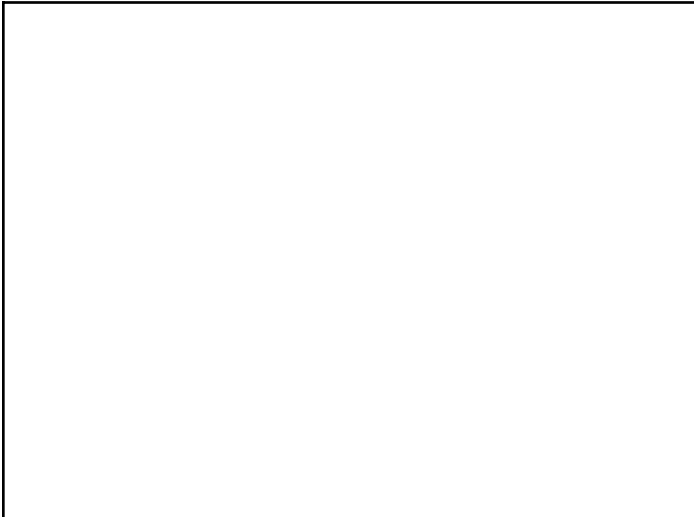
Step 7: Stop the process

/* Program to Simulate First Come First Serve CPU Scheduling Algorithm */

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
int i,j,n,bt[10],compt[10],at[10], wt[10],tat[10];
float sumwt=0.0,sumtat=0.0,avgwt,avgtat;
clrscr();
printf("Enter number of processes: ");
scanf("%d",&n);
printf("Enter the burst time of %d process\n", n);
for(i=0;i<n;i++)
{
scanf("%d",&bt[i]);
}
printf("Enter the arrival time of %d process\n", n);
for(i=0;i<n;i++)
{
scanf("%d",&at[i]);
}
}
```

```
compt[0]=bt[0]-at[0];
for(i=1;i<n;i++)
compt[i]=bt[i]+compt[i-1];
for(i=0;i<n;i++)
{
tat[i]=compt[i]-at[i];
wt[i]=tat[i]-bt[i];
sumtat+=tat[i];
sumwt+=wt[i];
}
avgwt=sumwt/n;
avgtat=sumtat/n;
printf(".....\n");
printf("PN\tBt\tCt\tTat\tWt\n");
printf(".....\n");
for(i=0;i<n;i++)
{
printf("%d\t%2d\t%2d\t%2d\t%2d\n",i,bt[i],compt[i],tat[i],wt[i]);
}
printf(".....\n");
printf(" Avgwt = %.2f\tAvgtat =%.2f\n",avgwt,avgtat);
printf(".....\n");
getch();
}
```

OUTPUT:



2. SHORTEST JOB FIRST:

The criteria of this algorithm are which process having the smallest CPU burst, CPU is assigned to that next process. If two process having the same CPU burst time FCFS is

used to break the tie.

DMCE COMPUTER DEPT OSL Page 24

Algorithm for SJF:

Step 1: Start the process

Step 2: Accept the number of processes in the ready Queue

Step 3: For each process in the ready Q, assign the process id and accept the CPU burst time

Step 4: Start the Ready Q according the shortest Burst time by sorting according to lowest to highest burst time.

Step 5: Set the waiting time of the first process as '0' and its turnaround time as its burst time.

Step 6: For each process in the ready queue, calculate

(a) Waiting time for process(n)=waiting time of process (n-1) + Bursttime of process(n-1)

(b) Turn around time for Process(n)= waiting time of Process(n)+ Burst time for process(n) **Step 7:** Calculate

(c) Average waiting time = Total waiting Time / Number of process

(d) Average Turnaround time = Total Turnaround Time / Number of process

Step 8: Stop the process

/* Program to Simulate Shortest Job First CPU Scheduling Algorithm

***/ #include<stdio.h>**

#include<conio.h>

#include<string.h>

void main()

{

int i,j,n,bt[10],compt[10], wt[10],tat[10],temp;

float sumwt=0.0,sumtat=0.0,avgwt,avgtat;

clrscr();

printf("Enter number of processes: ");

scanf("%d",&n);

printf("Enter the burst time of %d process\n", n);

for(i=0;i<n;i++)

{

scanf("%d",&bt[i]);

}

for(i=0;i<n;i++)

for(j=i+1;j<n;j++)

if(bt[i]>bt[j])

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

compt[0]=bt[0];

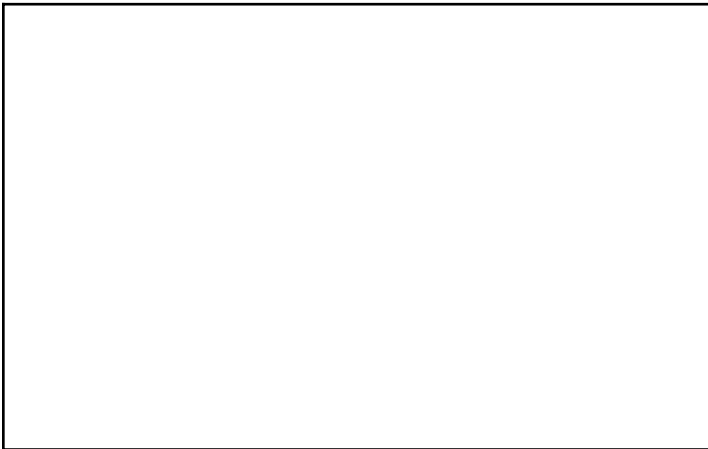
for(i=1;i<n;i++)

```
compt[i]=bt[i]+compt[i-1];
```

DMCE COMPUTER DEPT OSL Page 25

```
for(i=0;i<n;i++)
{
tat[i]=compt[i];
wt[i]=tat[i]-bt[i];
sumtat+=tat[i];
sumwt+=wt[i];
}
avgwt=sumwt/n;
avgtat=sumtat/n;
printf(".....\n");
printf("Bt\tCt\tTat\tWt\n");
printf(".....\n");
for(i=0;i<n;i++)
{
printf("%2d\t%2d\t%2d\t%2d\n",i,bt[i],compt[i],tat[i],wt[i]);
}
printf(".....\n");
printf(" Avgwt = %.2f\tAvgtat =%.2f\n",avgwt,avgtat);
printf(".....\n");
getch();
}
```

OUTPUT:



CONCLUSION:- Thus we have studied FCFS & SJF scheduling algorithm and its implementation.

EXPERIMENT NO:- 6

AIM:- Program to simulate producer and consumer problem using semaphores

THEORY:-

The producer consumer problem is a synchronization problem. We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of number of items in the buffer at any given time and “Empty” keeps track of number of unoccupied slots.

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

int mutex=1,full=0,empty=3,x=0;

int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                producer();
            else
                printf("Buffer is full!!");
                break;
            case 2: if((mutex==1)&&(full!=0))
                consumer();
            else
                printf("Buffer is empty!!"); break;
            case 3:
```

```
    exit(0);  
    break;  
}  
}
```

```
return 0;
```

DMCE COMPUTER DEPT OSL Page 27

```
}
```

```
int wait(int s)  
{  
    return (--s);  
}
```

```
int signal(int s)  
{  
    return(++s);  
}
```

```
void producer()  
{  
    mutex=wait(mutex);  
    full=signal(full);  
    empty=wait(empty);  
    x++;  
    printf("\nProducer produces the item %d",x);  
    mutex=signal(mutex);  
}
```

```
void consumer()  
{  
    mutex=wait(mutex);  
    full=wait(full);  
    empty=signal(empty);  
    printf("\nConsumer consumes item %d",x);  
    x--;  
    mutex=signal(mutex);  
}
```

Output



CONCLUSION:- Hence we have studied and implemented semaphore to simulate producer and consumable problem.

EXPERIMENT NO:-7

AIM:-

Write a program to demonstrate the concept of deadlock avoidance through Banker's Algorithm

THEORY:-

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Why Banker's algorithm is named so? 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. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money that bank has and if the remaining amount is greater than S then only the loan is sanctioned. It is done because if all the account holders comes to withdraw their money then the bank can easily do it.

In other words, 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.

```
// Banker's Algorithm
#include <stdio.h>
int main()
{
    // P0, P1, P2, P3, P4 are the Process names here

    int n, m, i, j, k;
    n = 5; // Number of processes
    m = 3; // Number of resources
    int alloc[5][3] = { { 0, 1, 0 }, // P0 // Allocation Matrix    { 2,
0, 0 }, // P1
                        { 3, 0, 2 }, // P2
                        { 2, 1, 1 }, // P3
                        { 0, 0, 2 } }; // P4

    int max[5][3] = { { 7, 5, 3 }, // P0 // MAX Matrix    { 3,
2, 2 }, // P1
                     { 9, 0, 2 }, // P2
                     { 2, 2, 2 }, // P3
                     { 4, 3, 3 } }; // P4

    int avail[3] = { 3, 3, 2 }; // Available Resources
```

```
for (k = 0; k < n; k++) {
```

```

f[k] = 0;
}
int need[n][m];
for (i = 0; i < n; i++) {
for (j = 0; j < m; j++)
need[i][j] = max[i][j] - alloc[i][j];    }
int y = 0;
for (k = 0; k < 5; k++) {
    for (i = 0; i < n; i++) {
if (f[i] == 0) {

int flag = 0;
for (j = 0; j < m; j++) {
if (need[i][j] > avail[j]){    flag = 1;
break;
}
}

if (flag == 0) {
ans[ind++] = i;
        for (y = 0; y < m; y++)
avail[y] += alloc[i][y];    f[i] = 1;
    }
}
}
}

printf("Following is the SAFE Sequence\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d", ans[n - 1]);

return (0);

}

```

Output:

Following is the SAFE Sequence

P1 -> P3 -> P4 -> P0 -> P2

EXPERIMENT NO:-8

AIM: - Write a program to implement dynamic partitioning placement algorithms i.e Best Fit, First –Fit and Worst –Fit.

THEORY:-

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple partition method, 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. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided.

When it is time to load or swap a process into main memory, and if there is more than one free block of memory of sufficient size, then the operating system must decide which free block to allocate. Best-fit strategy chooses the block that is closest in size to the request. First-fit chooses the first available block that is large enough. Worst fit chooses the largest available block.

/*Program to implement BEST-FIT dynamic partitioning placement algorithms*/

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;
static int bf[max],ff[max];
clrscr();
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
printf("Block %d:",i);scanf("%d",&b[i]);
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1)
```

DMCE COMPUTER DEPT OSL Page 32

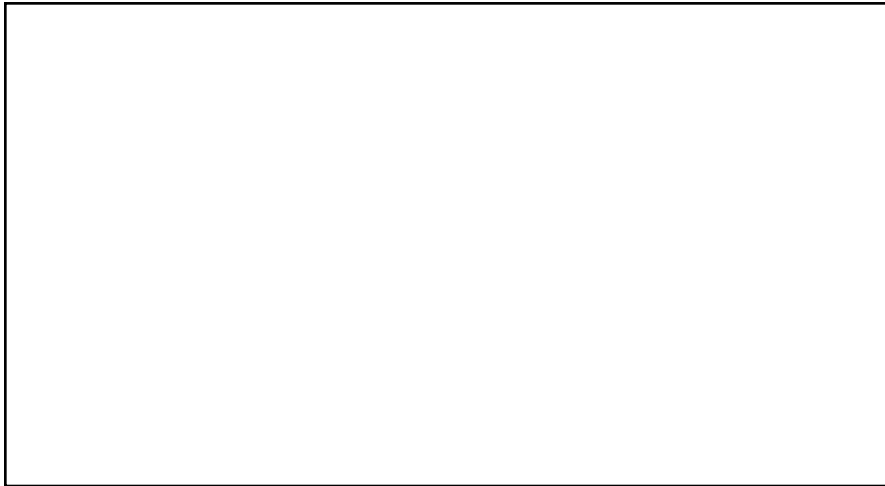
```
{
temp=b[j]-f[i];
```

```

if(temp>=0)
if(lowest>temp)
{
ff[i]=j;
lowest=temp;
}
}
}
frag[i]=lowest;
bf[ff[i]]=1;
lowest=10000;
}
printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");
for(i=1;i<=nf && ff[i]!=0;i++)
printf("\n%d\t%d\t%d\t%d\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}

```

OUTPUT:



```

/*Program to implement FIRST-FIT dynamic partitioning placement algorithm*/
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - Worst Fit");

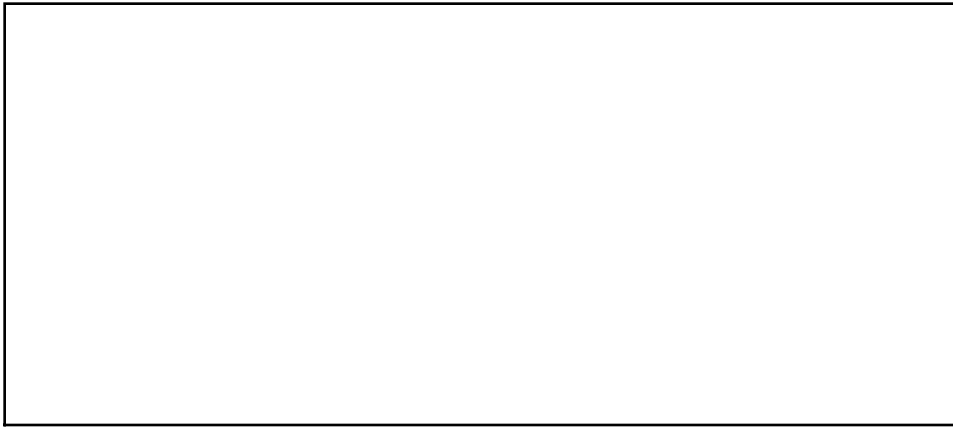
```

```

printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
if(bf[j]!=1) //if bf[j] is not allocated
{
temp=b[j]-f[i];
if(temp>=0)
if(highest<temp)
{
ff[i]=j;
highest=temp;
}
}
}
frag[i]=highest;
bf[ff[i]]=1;
highest=0;
}
printf("\nFile_no:\tFile_size
:\tBlock_no:\tBlock_size:\tFragement"); for(i=1;i<=nf;i++)
printf("\n%d\t%d\t%d\t%d\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}

```

OUTPUT:



/*Program to implement Worst-Fit dynamic partitioning placement algorithm*/

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{
int frag[max],b[max],f[max],i,j,nb,nf,temp;
static int bf[max],ff[max];
clrscr();
printf("\n\tMemory Management Scheme - First Fit");
printf("\nEnter the number of blocks:");
scanf("%d",&nb);
printf("Enter the number of files:");
scanf("%d",&nf);
printf("\nEnter the size of the blocks:-\n");
for(i=1;i<=nb;i++)
{
printf("Block %d:",i);
scanf("%d",&b[i]);
}
printf("Enter the size of the files :-\n");
for(i=1;i<=nf;i++)
{
printf("File %d:",i);
scanf("%d",&f[i]);
}
for(i=1;i<=nf;i++)
{
for(j=1;j<=nb;j++)
{
```



```
if(bf[j]!=1)
{
temp=b[j]-f[i];
if(temp>=0)
```

DMCE COMPUTER DEPT OSL Page 35

```
{
ff[i]=j;
break;
}
}
}
frag[i]=temp;
bf[ff[i]]=1;
}
printf("\nFile_no:\tFile_size
:\tBlock_no:\tBlock_size:\tFragement"); for(i=1;i<=nf;i++)
printf("\n%d\t%d\t%d\t%d\t%d",i,f[i],ff[i],b[ff[i]],frag[i]);
getch();
}
```

OUTPUT:



CONCLUSION:- Thus we have studied and implemented dynamic partitioning placement algorithms i.e Best Fit, First –Fit and Worst –Fit.

EXPERIMENT NO: 9

AIM:- Write a program to implement various page replacement policies.

THEORY:-

In a operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

Page Fault – A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Page Replacement Algorithms :

1. First In First Out (FIFO) –

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.

/*Program to implement FIFO page replacement algorithm*/

```
#include<stdio.h>
#include<conio.h>
main()
{
int i, j, k, f, pf=0, count=0, rs[25], m[10], n;
clrscr();
printf("\n Enter the length of reference string -- ");
scanf("%d",&n);
printf("\n Enter the reference string -- ");
for(i=0;i<n;i++)
```

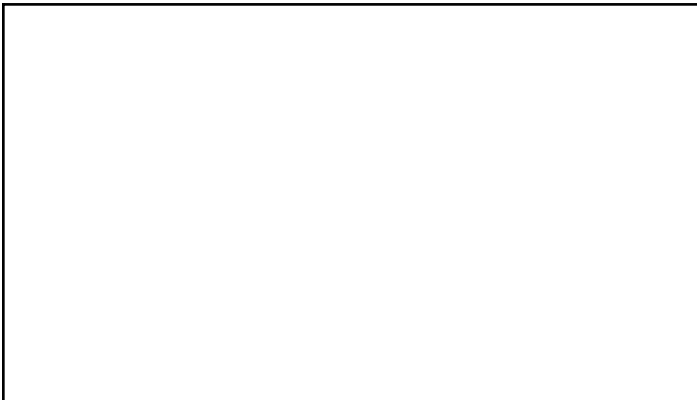
```

scanf("%d",&rs[i]);
printf("\n Enter no. of frames -- ");
scanf("%d",&f);
for(i=0;i<f;i++)
m[i]=-1;
printf("\n The Page Replacement Process is -- \n");
for(i=0;i<n;i++)
{
for(k=0;k<f;k++)
{
if(m[k]==rs[i])
break;

DMCE COMPUTER DEPT OSL Page 37
}
if(k==f)
{
m[count++]=rs[i];
pf++;
}
for(j=0;j<f;j++)
printf("\t%d",m[j]);
if(k==f)
printf("\tPF No. %d",pf);
printf("\n");
if(count==f)
count=0;
}
printf("\n The number of Page Faults using FIFO are %d",pf);
getch();
}

```

OUTPUT:



2. Least Recently Used (LRU)

In Least Recently Used (LRU) algorithm is a Greedy algorithm where the page to be replaced is least recently used. The idea is based on locality of reference, the least recently used page is not likely.

/*Program to implement LRU page replacement algorithm*/

```
#include<stdio.h>
#include<conio.h>
main()
{
int i, j , k, min, rs[25], m[10], count[10], flag[25], n, f, pf=0,
next=1; clrscr();
printf("Enter the length of reference string -- ");
scanf("%d",&n);
printf("Enter the reference string -- ");
```

DMCE COMPUTER DEPT OSL Page 38

```
for(i=0;i<n;i++)
{
scanf("%d",&rs[i]);
flag[i]=0;
}
printf("Enter the number of frames --
"); scanf("%d",&f);
for(i=0;i<f;i++)
{
count[i]=0;
m[i]=-1;
}
printf("\n\nThe Page Replacement process is --
\n"); for(i=0;i<n;i++)
{
for(j=0;j<f;j++)
{
if(m[j]==rs[i])
{
flag[i]=1;
count[j]=next;
next++;
}
}
if(flag[i]==0)
{
if(i<f)
{
m[i]=rs[i];
count[i]=next;
next++;
```

```

    }
    else
    {
        min=0;
        for(j=1;j<f;j++)
            if(count[min] > count[j])
                min=j;
        m[min]=rs[i];
        count[min]=next;
        next++;
    }
    pf++;
}
for(j=0;j<f;j++)
    printf("%d\t", m[j]);

```

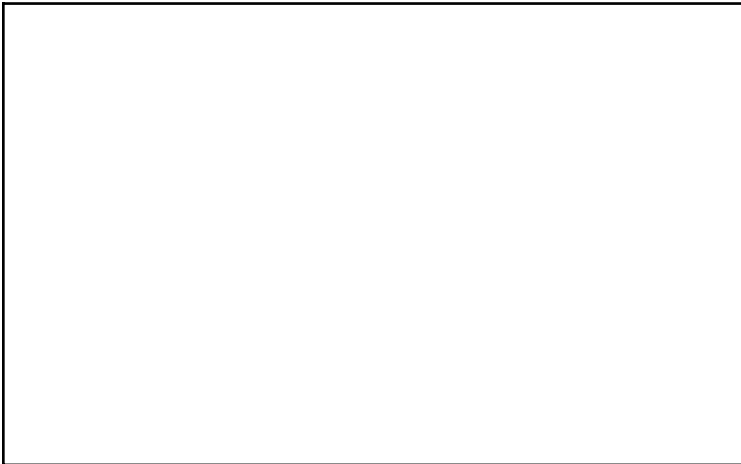
DMCE COMPUTER DEPT OSL Page 39

```

if(flag[i]==0)
    printf("PF No. -- %d" , pf);
printf("\n");
}
printf("\nThe number of page faults using LRU are %d",pf);
getch();
}

```

OUTPUT:



CONCLUSION: -Thus we have studied and implemented page replacement algorithms.

EXPERIMENT NO:-10

AIM: - Write a program to implement Disk Scheduling algorithms like FCFS, SCAN and C-SCAN.

THEORY:-

Disk scheduling is done by operating systems to schedule I/O requests arriving for the disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by the disk controller. Thus other I/O requests need to wait in the waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of the computer system and thus need to be accessed in an efficient manner.

Some of the important terms

- **Seek Time:** Seek time is the time taken to locate the disk arm to a specified track where the data is to be read or write. So the disk scheduling algorithm that gives minimum average seek time is better.
- **Rotational Latency:** Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads. So the disk scheduling algorithm that gives minimum rotational latency is better.
- **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
- **Disk Access Time:**

Disk Access Time is:

Disk Access Time = Seek Time + Rotational Latency + Transfer Time

Disk Scheduling Algorithms

FCFS: FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.

C program for FCFS disk scheduling:

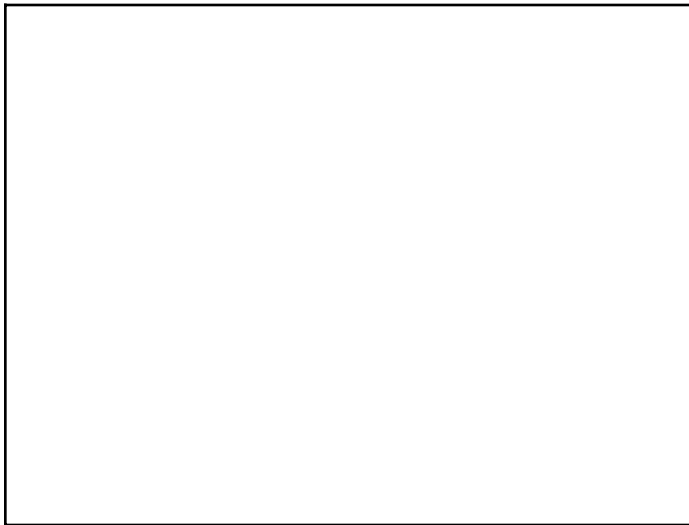
```
#include<stdio.h>
int main()
{
    int queue[20],n,head,i,j,k,seek=0,max,diff;
    float avg;
    printf("Enter the max range of disk\n");
    scanf("%d",&max);
    printf("Enter the size of queue request\n");
    scanf("%d",&n);
```

```

printf("Enter the queue of disk positions to be read\n");
for(i=1;i<=n;i++)
scanf("%d",&queue[i]);
printf("Enter the initial head position\n");
scanf("%d",&head);
queue[0]=head;
for(j=0;j<=n-1;j++)
{
diff=abs(queue[j+1]-queue[j]);
seek+=diff;
printf("Disk head moves from %d to %d with
seek %d\n",queue[j],queue[j+1],diff );
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}

```

OUTPUT :



SCAN disk scheduling :

DMCE COMPUTER DEPT OSL Page 42

In the SCAN algorithm, the disk arm starts at one end of the disk and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk. The SCAN algorithm is sometimes called the elevator algorithm, since the disk arm behaves just like an elevator in a building, first servicing all the requests going up and then reversing to service requests the other way.

C program for SCAN disk scheduling :

```
#include<stdio.h>
int main()
{
    int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],
    temp1=0,temp2=0;
    float avg;
    printf("Enter the max range of disk\n");
    scanf("%d",&max);
    printf("Enter the initial head position\n");
    scanf("%d",&head);
    printf("Enter the size of queue request\n");
    scanf("%d",&n);
    printf("Enter the queue of disk positions to be read\n");
    for(i=1;i<=n;i++)
    {
        scanf("%d",&temp);
```



```

if(temp>=head)
{
queue1[temp1]=temp;
temp1++;
}
else
{
queue2[temp2]=temp;
temp2++;
}
}
for(i=0;i<temp1-1;i++)
{
for(j=i+1;j<temp1;j++)
{
if(queue1[i]>queue1[j])
{
temp=queue1[i];
queue1[i]=queue1[j];
queue1[j]=temp;

```

```

}
}
}
for(i=0;i<temp2-1;i++)
{
for(j=i+1;j<temp2;j++)
{
if(queue2[i]<queue2[j])
{
temp=queue2[i];
queue2[i]=queue2[j];
queue2[j]=temp;
}
}
}
for(i=1,j=0;j<temp1;i++,j++)
queue[i]=queue1[j];
queue[i]=max;

```

```

for(i=temp1+2,j=0;j<temp2;i++,j++)
queue[i]=queue2[j];
queue[i]=0;
queue[0]=head;
for(j=0;j<=n+1;j++)

```

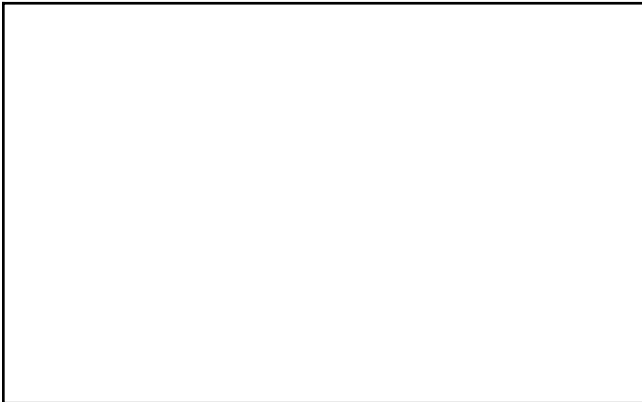


```

diff=abs(queue[j+1]-queue[j]);
seek+=diff;
printf("Disk head moves from %d to %d with
seek %d\n",queue[j],queue[j+1],diff );
}
printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}

```

OUTPUT :



C-SCAN disk scheduling :

Circular SCAN (C-SCAN) scheduling is a variant of SCAN designed to provide a more uniform wait time. Like SCAN, C-SCAN moves the head from one end of the disk to the other, servicing requests along the way. When the head reaches the other end, however, it immediately returns to the beginning of the disk without servicing any requests on the return trip (Figure 10.7). The C-SCAN scheduling algorithm essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

C program for C-SCAN disk scheduling :

```
#include<stdio.h>
int main()
{
    int queue[20],n,head,i,j,k,seek=0,max,diff,temp,queue1[20],queue2[20],
    temp1=0,temp2=0;
    float avg;
    printf("Enter the max range of disk\n");
    scanf("%d",&max);
    printf("Enter the initial head position\n");
    scanf("%d",&head);
    printf("Enter the size of queue request\n");
    scanf("%d",&n);
    printf("Enter the queue of disk positions to be read\n");
    for(i=1;i<=n;i++)
    {
        scanf("%d",&temp);
        if(temp>=head)
        {
            queue1[temp1]=temp;
            temp1++;
        }
        else
        {
            queue2[temp2]=temp;
            temp2++;
        }
    }
    for(i=0;i<temp1-1;i++)
    {
        for(j=i+1;j<temp1;j++)
        {
            if(queue1[i]>queue1[j])
            {
                temp=queue1[i];
                queue1[i]=queue1[j];
                queue1[j]=temp;
            }
        }
    }
    for(i=0;i<temp2-1;i++)
```

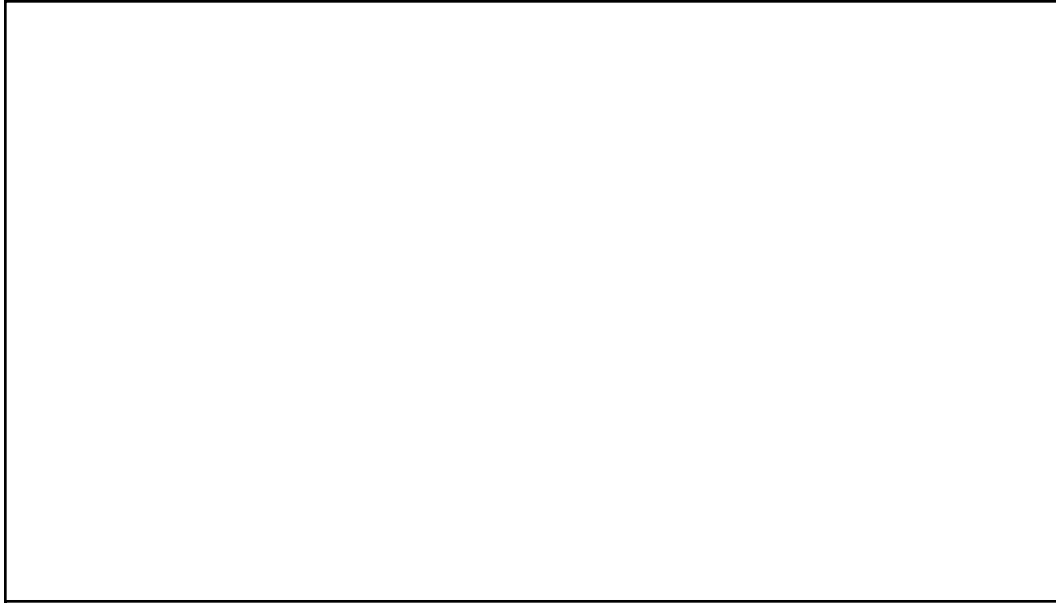
```

{
for(j=i+1;j<temp2;j++)
{
if(queue2[i]>queue2[j])
{
temp=queue2[i];
queue2[i]=queue2[j];
queue2[j]=temp;
}
}
}
for(i=1,j=0;j<temp1;i++,j++)
queue[i]=queue1[j];
queue[i]=max;
queue[i+1]=0;
for(i=temp1+3,j=0;j<temp2;i++,j++)
queue[i]=queue2[j];
queue[0]=head;
for(j=0;j<=n+1;j++)
{
diff=abs(queue[j+1]-queue[j]);
seek+=diff;
printf("Disk head moves from %d to %d with
seek %d\n",queue[j],queue[j+1],diff); }

printf("Total seek time is %d\n",seek);
avg=seek/(float)n;
printf("Average seek time is %f\n",avg);
return 0;
}

```

OUTPUT :



CONCLUSION: -Thus we have studied and implemented Disk Scheduling algorithms like FCFS, SCAN and C-SCAN.