

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

LAB MANUAL

**CS23431 – OPERATING SYSTEMS**

**(REGULATION 2023)**

**RAJALAKSHMI ENGINEERING COLLEGE**

**Thandalam, Chennai-602015**

Name: AKSHAYAA S

Register No : 2116231801007

Year / Branch / Section: II / B.TECH AI&DS/FA

Semester: IV

Academic Year: 2024 - 2025

**INDEX**

|  |  |  |
| --- | --- | --- |
| **EXP.NO** | **Date** | **Title** |
| 1a | 23/01/2025 | Installation and Configuration of Linux |
| 2 | 29/01/2025 | Basic Linux Commands |
| 3 a) | 05/02/2025 | Shell script   1. Arithmetic Operation -using expr command 2. Check leap year using if-else |
| 3 b) | 18/02/2025 | 1. Reverse the number using while loop 2. Fibonacci series using for loop |
| 4 | 19/02/2025 | Text processing using Awk script   1. Employee average pay 2. Results of an examination |
| 5 | 19/02/2025 | System calls –fork(), exec(), getpid(),opendir(), readdir() |
| 6a | 25/02/2025 | FCFS |
| 6b | 26/02/2025 | SJF |
| 6c | 05/03/2025 | Priority |
| 6d | 25/03/2025 | Round Robin |
| 7. | 26/03/2025 | Inter-process Communication using Shared Memory |
| 8 | 01/04/2025 | Producer Consumer using Semaphores |
| 9 | 08/04/2025 | Bankers Deadlock Avoidance algorithms |

|  |  |  |
| --- | --- | --- |
| 10 a | 15/04/2025 | Best Fit |
| 10 b | 22/04/2025 | First Fit |
| 11a | 22/04/2025 | FIFO |
| 11b | 23/04/2025 | LRU |
| 11c | 29/04/2025 | Optimal |
| 12 | 29/04/2025 | File Organization Technique- single and Two level directory |

**Ex No: 1a)**

## INSTALLATION AND CONFIGURATION OF LINUX

### Aim:

To install and configure Linux operating system in a Virtual Machine.

### Installation/Configuration Steps:

1. Install the required packages for virtualization dnf install xen virt-manager qemu libvirt
2. Configure xend to start up on boot systemctl enable virt-manager.service
3. Reboot the machine Reboot
4. Create Virtual machine by first running virt-manager virt-manager &
5. Click on File and then click to connect to localhost
6. In the base menu, right click on the localhost(QEMU) to create a new VM 7. Select Linux ISO image
7. Choose puppy-linux.iso then kernel version
8. Select CPU and RAM limits
9. Create default disk image to 8 GB
10. Click finish for creating the new VM with PuppyLinux

### Result :

Thus, to implement installation and configuration of Linux has been executed successfully.

**Ex No: 2**

## BASIC LINUX COMMANDS

* 1. **GENERAL PURPOSE COMMANDS**

1. The ‘date’ command:

The date command displays the current date with day of week, month, day, time (24 hours clock) and the year.

SYNTAX: $ date

The date command can also be used with following format.

|  |  |  |
| --- | --- | --- |
| Format | Purpose | Example |
| + %m | To display only month | $ date + %m |
| + %h | To display month name | $ date + %h |
| + %d | To display day of month | $ date + %d |
| + %y | To display last two digits of the year | $ date + %y |
| + %H | To display Hours | $ date + %H |
| + %M | To display Minutes | $ date + %M |
| + %S | To display Seconds | $ date + %S |

1. The echo’command:

The echo command is used to print the message on the screen. SYNTAX: $ echo

EXAMPLE: $ echo “God is Great”

1. The ‘cal’ command:

The cal command displays the specified month or year calendar.

SYNTAX: $ cal [month] [year] EXAMPLE: $ cal Jan 2012

1. The ‘bc’ command:

Unix offers an online calculator and can be invoked by the command bc.

SYNTAX: $ bc EXAMPLE: bc –l 16/4

5/2

1. The ‘who’ command

The who command is used to display the data about all the users who are currently logged into the system.

SYNTAX: $ who

1. The ‘who am i’ command

The who am i command displays data about login details of the user.

SYNTAX: $ who am i

1. The ‘id’ command

The id command displays the numerical value corresponding to your login.

SYNTAX: $ id

1. The ‘tty’ command

The tty (teletype) command is used to know the terminal name that we are using.

SYNTAX: $ tty

1. The ‘clear’ command

The clear command is used to clear the screen of your terminal.

SYNTAX: $ clear

1. The ‘man’ command

The man command gives you complete access to the Unix commands.

SYNTAX: $ man [command]

1. The ‘ps’ command

The ps command is used to the process currently alive in the machine with the 'ps' (process status) command, which displays information about process that are alive when you run the command. 'ps;' produces a snapshot of machine activity.

SYNTAX: $ ps EXAMPLE: $ ps

$ ps –e

$ps -aux

1. The ‘uname’ command

The uname command is used to display relevant details about the operating system on the standard output.

-m -> Displays the machine id (i.e., name of the system hardware)

-n -> Displays the name of the network node. (host name)

-r -> Displays the release number of the operating system.

-s -> Displays the name of the operating system (i.e.. system name)

-v -> Displays the version of the operating system.

-a -> Displays the details of all the above five options.

SYNTAX: $ uname [option] EXAMPLE: $ uname -a

## DIRECTORY COMMANDS

1. The ‘pwd’ command:

The pwd (print working directory) command displays the current working directory.

SYNTAX: $ pwd

1. The ‘mkdir’ command:

The mkdir is used to create an empty directory in a disk.

SYNTAX: $ mkdir dirname EXAMPLE: $ mkdir receee

1. The ‘rmdir’ command:

The rmdir is used to remove a directory from the disk. Before removing a directory, the directory must be empty (no files and directories).

SYNTAX: $ rmdir dirname EXAMPLE: $ rmdir receee

1. The ‘cd’ command:

The cd command is used to move from one directory to another.

SYNTAX: $ cd dirname

EXAMPLE: $ cd receee

1. The ‘ls’ command:

The ls command displays the list of files in the current working directory.

SYNTAX: $ ls EXAMPLE: $ ls

$ ls –l

$ ls –a

## FILE HANDLING COMMANDS

1. The ‘cat’ command:

The cat command is used to create a file.

SYNTAX: $ cat > filename EXAMPLE: $ cat > rec

1. The ‘Display contents of a file’ command:

The cat command is also used to view the contents of a specified file.

SYNTAX: $ cat filename

1. The ‘cp’ command:

The cp command is used to copy the contents of one file to another and copies the file from one place to another.

SYNTAX: $ cp oldfile newfile EXAMPLE: $ cp cse ece

1. The ‘rm’ command:

The rm command is used to remove or erase an existing file SYNTAX: $ rm filename

EXAMPLE: $ rm rec

$ rm –f rec

Use option –fr to delete recursively the contents of the directory and its subdirectories.

1. The ‘mv’ command:

The mv command is used to move a file from one place to another. It removes a specified file from its original location and places it in specified location.

SYNTAX: $ mv oldfile newfile EXAMPLE: $ mv cse eee

1. The ‘file’ command:

The file command is used to determine the type of file.

SYNTAX: $ file filename EXAMPLE: $ file receee

1. The ‘wc’ command:

The wc command is used to count the number of words, lines and characters in a file.

SYNTAX: $ wc filename EXAMPLE: $ wc receee

1. The ‘Directing output to a file’ command:

The ls command lists the files on the terminal (screen). Using the redirection operator ‘>’ we can send the output to file instead of showing it on the screen.

SYNTAX: $ ls > filename EXAMPLE: $ ls > cseeee

1. The ‘pipes’ command:

The Unix allows us to connect two commands together using these pipes. A pipe ( | ) is an mechanism by which the output of one command can be channeled into the input of another command.

SYNTAX: $ command1 | command2 EXAMPLE: $ who | wc -l

1. The ‘tee’ command:

While using pipes, we have not seen any output from a command that gets piped into another command. To save the output, which is produced in the middle of a pipe, the tee command is very useful. SYNTAX: $ command | tee filename

EXAMPLE: $ who | tee sample | wc -l

1. The ‘Metacharacters of unix’ command:

Metacharacters are special characters that are at higher and abstract level compared to most of other characters in Unix. The shell understands and interprets these metacharacters in a special way.

\* - Specifies number of characters

?- Specifies a single character

[ ]- used to match a whole set of file names at a command line.

! – Used to Specify Not

EXAMPLE:

$ ls r\*\* - Displays all the files whose name begins with ‘r’

$ ls ?kkk - Displays the files which are having ‘kkk’, from the second characters irrespective of the first character.

$ ls [a-m] – Lists the files whose names begins alphabets from ‘a’ to ‘m’

$ ls [!a-m] – Lists all files other than files whose names begins alphabets from ‘a’ to ‘m’ 12.

The ‘File permissions’ command:

File permission is the way of controlling the accessibility of file for each of three users namely Users, Groups and Others.

There are three types of file permissions are available, they are

### r-read w-write

**x-execute**

The permissions for each file can be divided into three parts of three bits each.

|  |  |
| --- | --- |
| First three bits | Owner of the file |
| Next three bits | Group to which owner of the file belongs |
| Last three bits | Others |

EXAMPLE: $ ls college

-rwxr-xr-- 1 Lak std 1525 jan10 12:10 college Where,

-rwx The file is readable, writable and executable by the owner of the file.

Lak Specifies Owner of the file.

r-x Indicates the absence of the write permission by the Group owner of the file. Std Is the Group Owner of the file.

r-- Indicates read permissions for others.

1. The ‘chmod’ command:

The chmod command is used to set the read, write and execute permissions for all categories of users for file.

SYNTAX: $ chmod category operation permission file

|  |  |  |
| --- | --- | --- |
| Category | Operation | permission |
| u-users | + assign | r-read |
| g-group | -Remove | w-write |
| o-others | = assign absolutely | x-execute |
| a-all |  |  |

EXAMPLE:

$ chmod u –wx college

Removes write & execute permission for users for ‘college’ file.

$ chmod u +rw, g+rw college

Assigns read & write permission for users and groups for ‘college’ file.

$ chmod g=wx college

Assigns absolute permission for groups of all read, write and execute permissions for ‘college’ file.

1. The ‘Octal Notations’ command:

The file permissions can be changed using octal notations also. The octal notations for file permission are

|  |  |
| --- | --- |
| Read permission | 4 |
| Write permission | 2 |

EXAMPLE:

$ chmod 761 college

|  |  |
| --- | --- |
| Execute permission | 1 |

Assigns all permission to the owner, read and write permissions to the group and only executable permission to the others for ‘college’ file.

## GROUPING COMMANDS

1. The ‘semicolon’ command:

The semicolon(;) command is used to separate multiple commands at the command line.

SYNTAX: $ command1;command2;command3 ;commandn

EXAMPLE: $ who;date

1. The ‘&&’ operator:

The ‘&&’ operator signifies the logical AND operation in between two or more valid Unix commands.It means that only if the first command is successfully executed, then the next command will executed.

SYNTAX: $ command1 && command && command3 &&commandn

EXAMPLE: $ who && date

1. The ‘||’ operator:

The ‘||’ operator signifies the logical OR operation in between two or more valid Unix commands.It means, that only if the first command will happen to be un successfully,it will continue to execute next commands.

SYNTAX: $ command1 || command || command3 ||commandn

EXAMPLE: $ who || date

* 1. FILTERS

1. The head filter

It displays the first ten lines of a file.

SYNTAX: $ head filename

EXAMPLE: $ head college Display the top ten lines.

$ head -5 college Display the top five lines.

1. The tail filter

It displays ten lines of a file from the end of the file.

SYNTAX: $ tail filename

EXAMPLE: $ tail college Display the last ten lines.

$tail -5 college Display the last five lines.

1. The more filter:

The pg command shows the file page by page.

SYNTAX: $ ls –l | more

1. The ‘grep’ command:

This command is used to search for a particular pattern from a file or from the standard input and display those lines on the standard output. “Grep” stands for “global search for regular expression.”

SYNTAX: $ grep [pattern] [file\_name] EXAMPLE: $ cat> student

Arun cse Ram ece Kani cse

$ grep “cse” student Arun cse

Kani cse

1. The ‘sort’ command:

The sort command is used to sort the contents of a file. The sort command reports only to the

screen, the actual file remains unchanged. SYNTAX: $ sort filename

EXAMPLE: $ sort college OPTIONS:

|  |  |
| --- | --- |
| Command | Purpose |
| Sort –r college | Sorts and displays the file contents in reverse order |
| Sort –c college | Check if the file is sorted |
| Sort –n college | Sorts numerically |
| Sort –m college | Sorts numerically in reverse order |

|  |  |
| --- | --- |
| Sort –u college | Remove duplicate records |
| Sort –l college | Skip the column with +1 (one) option.Sorts according to second column |

1. The ‘nl’ command:

The nl filter adds lines numbers to a file and it displays the file and not provides access to edit but simply displays the contents on the screen.

SYNTAX: $ nl filename EXAMPLE: $ nl college

1. The ‘cut’ command:

We can select specified fields from a line of text using cut command.

SYNTAX: $ cut -c filename EXAMPLE: $ cut -c college OPTION:

-c – Option cut on the specified character position from each line.

## 1.5 OTHER ESSENTIAL COMMANDS

### free

Display amount of free and used physical and swapped memory system. synopsis- free [options]

example

[root@localhost ~]# free -t

total used free shared buff/cache available Mem: 4044380 605464 2045080 148820 1393836 3226708 Swap: 2621436 0 2621436

Total: 6665816 605464 4666516

### top

It provides a dynamic real-time view of processes in the system. synopsis- top [options]

example

[root@localhost ~]# top

top - 08:07:28 up 24 min, 2 users, load average: 0.01, 0.06, 0.23

Tasks: 211 total, 1 running, 210 sleeping, 0 stopped, 0 zombie

%Cpu(s): 0.8 us, 0.3 sy, 0.0 ni, 98.9 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st

KiB Mem : 4044380 total, 2052960 free, 600452 used, 1390968 buff/cache KiB Swap: 2621436 total, 2621436 free, 0 used. 3234820 avail Mem PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

1105 root 20 0 175008 75700 51264 S 1.7 1.9 0:20.46 Xorg 2529 root 20 0 80444

32640 24796 S 1.0 0.8 0:02.47 gnome-term 3. **ps**

It reports the snapshot of current processes synopsis- ps [options]

example

[root@localhost ~]# ps -e

PID TTY TIME CMD

1 ? 00:00:03 systemd

2 ? 00:00:00 kthreadd

3 ? 00:00:00 ksoftirqd/0

### vmstat

It reports virtual memory statistics synopsis- vmstat [options] example

[root@localhost ~]# vmstat

procs memory swap io -system cpu

-- r b swpd free buff cache si so bi bo in cs us sy id wa st 0 0 0 1879368 1604 1487116 0 0 64 7 72 140 1 0 97 1 0

### df

It displays the amount of disk space available in file-system.

**S**ynopsis- df [options] example [root@localhost ~]# df

Filesystem 1K-blocks Used Available Use% Mounted on

devtmpfs 2010800 0 2010800 0% /dev tmpfs 2022188 148 2022040 1% /dev/shm

tmpfs 2022188 1404 2020784 1% /run /dev/sda6 487652 168276 289680 37% /boot

### ping

It is used verify that a device can communicate with another on network. PING stands

for Packet Internet Groper. synopsis- ping [options] [root@localhost ~]# ping 172.16.4.1

PING 172.16.4.1 (172.16.4.1) 56(84) bytes of data.

64 bytes from 172.16.4.1: icmp\_seq=1 ttl=64 time=0.328 ms 64 bytes from 172.16.4.1: icmp\_seq=2 ttl=64 time=0.228 ms

64 bytes from 172.16.4.1: icmp\_seq=3 ttl=64 time=0.264 ms 64 bytes from 172.16.4.1: icmp\_seq=4 ttl=64 time=0.312 ms

^C

--- 172.16.4.1 ping statistics ---

4 packets transmitted, 4 received, 0% packet loss, time 3000ms rtt min/avg/max/mdev = 0.228/0.283/0.328/0.039 ms

### ifconfig

It is used configure network interface. synopsis- ifconfig [options]

example

[root@localhost ~]# ifconfig

enp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu

1500 inet 172.16.6.102 netmask 255.255.252.0 broadcast 172.16.7.255 inet6 fe80::4a0f:cfff:fe6d:6057 prefixlen 64 scopeid 0x20<link>

ether 48:0f:cf:6d:60:57 txqueuelen 1000 (Ethernet)

RX packets 23216 bytes 2483338 (2.3 MiB)

RX errors 0 dropped 5 overruns 0 frame 0

TX packets 1077 bytes 107740 (105.2 KiB)

TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0 **8. traceroute**

It tracks the route the packet takes to reach the destination. synopsis- traceroute [options]

example

[root@localhost ~]# traceroute [www.rajalakshmi.org](http://www.rajalakshmi.org/)

traceroute to [www.rajalakshmi.org](http://www.rajalakshmi.org/) (220.227.30.51), 30 hops max, 60 byte

packets 1 gateway (172.16.4.1) 0.299 ms 0.297 ms 0.327 ms

2 220.225.219.38 (220.225.219.38) 6.185 ms 6.203 ms 6.189 ms

### Result:

Thus, to implement basic Linux commands has been executed successfully.

**Ex. no: 3a)**

**Shell Script**

**Aim:**

To write a Shellscript to to display basic calculator.

### Program:

echo "Enter two no:" read a

read b

echo "Add $((a+b))"

echo "Sub $((a-b))"

echo "Mul $((a\*b))"

echo "Div $((a/b))"

echo "Mod $((a%b))"

### Sample Input and Output

**Run the program using the below command**

[REC@local host~]$ sh arith.sh

Enter two no 5

10

add 15

sub -5

mul 50

div 0 mod 5c"

### Result:

Thus, to implement shell scripting for arithmetic operations has been executed successfully.

**Ex. no: 3a)**

**Shell Script**

**Aim:**

To write a Shellscript to test given year is leap or not using conditional statement

### Program:

echo "enter number" read year

if [ $((year%4)) = 0 ]; then

if [ $((year%100)) = 0 ]; then

if [ $((year%400)) = 0 ]; then echo "Leap year"

else

echo "Not a leap year"

fi else

echo "Leap year"

fi else

echo "Not a leap year" fi

### Sample Input and Output

**Run the program using the below command**

[REC @ local host~]$ sh leap.sh enter number

12

leap year

### Result:

Thus, to implement shell scripting for leap year check has been executed successfully.

**Ex. No.: 3b)**

**Shell Script – Reverse of Digit**

**Aim:**

To write a Shell script to reverse a given digit using looping statement.

### Program:

echo "enter number" read n

rev=0

while [ $n -ne 0 ]; do a=$((n%10))

rev=$((rev\*10+a)) n=$((n/10))

done

echo "$rev"

### Sample Input and Output

**Run the program using the below command**

[REC@local host~]$sh indhu.sh

enter number 123

321

### Result:

Thus, to implement reverse number using shell scripting has been executed successfully.

**Ex. No.: 3b)**

**Shell Script – Fibbonacci Series**

**Aim:**

To write a Shell script to generate a Fibonacci series using for loop.

### Program:

#!/bin/bash echo "n" read n

a=0 b=1

for ((i=0;i<n;i++)) do echo "$a" c=$((a+b))

a=$b b=$c done

### Sample Input and Output

**Run the program using the below command**

[REC@local host~]$sh indhu.sh

enter number 21

fibonacci series 0

1

1

2

3

5

8

13

21

34

55

89

144

233

377

### Result:

Thus, to implement Fibonacci series using shell scripting has been executed successfully.

**Ex. No.: 4a)**

## EMPLOYEE AVERAGE PAY

### Aim:

To find out the average pay of all employees whose salary is more than 6000 and no. of days worked is more than 4.

### Algorithm:

1. Create a flat file emp.dat for employees with their name, salary per day and number of days worked and save it.
2. Create an awk script emp.awk
3. For each employee record do
   1. If Salary is greater than 6000 and number of days worked is more than 4, then print name and salary earned
   2. Compute total pay of employee
4. Print the total number of employees satisfying the criteria and their average pay.

### Program Code:

#!/usr/bin/awk -f BEGIN {

totalPay = 0

count = 0

}

{

name = $1 salaryPerDay = $2 daysWorked = $3

totalSalary = salaryPerDay \* daysWorked

if (salaryPerDay > 6000 && daysWorked > 4) { print name, totalSalary

totalPay += totalSalary count++

}

}

END {

if (count > 0) {

avgPay = totalPay / count

print "\nNumber of employees are=", count print "Total Pay:", totalPay

print "Average Pay:", avgPay

} else {

print "No employees meet the criteria."

}

}

### Sample Input:

//emp.dat – Col1 is name, Col2 is Salary Per Day and Col3 is //no. of days worked JOE 8000 5

RAM 6000 5

TIM 5000 6

BEN 7000 7

AMY 6500 6

### Output:

**Run the program using the below commands** [student@localhost ~]$ vi emp.dat [student@localhost ~]$ vi emp.awk [student@localhost ~]$ gawk -f emp.awk emp.dat.

EMPLOYEES DETAILS JOE 40000

BEN 49000

AMY 39000

no of employees are= 3 total pay= 128000

average pay= 42666.7 [student@localhost ~]$

### Result:

Thus, to implement text processing using Awk for employee pay has been executed successfully.

**Ex. No.: 4b)**

## RESULTS OF EXAMINATION

### Aim:

To print the pass/fail status of a student in a class.

### Algorithm:

1. Read the data from file
2. Get a data from each column
3. Compare the all subject marks column
   1. If marks less than 45 then print Fail
   2. else print Pass

### Program Code:

**//marks.awk**

#!/usr/bin/awk -f BEGIN {

print "NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS"

print " " print " "

}

{

name = $1 sub1 = $2 sub2 = $3 sub3 = $4 sub4 = $5 sub5 = $6 sub6 = $7

if (sub1 < 50 || sub2 < 50 || sub3 < 50 || sub4 < 50 || sub5 < 50 || sub6 < 50) { status = "FAIL"

} else {

status = "PASS"

}

printf "%-6s %-5d %-5d %-5d %-5d %-5d %-5d %-2s\n", name, sub1, sub2, sub3, sub4, sub5, sub6, status

}

END {

print " "

}

### Input:

**//marks.dat**

**//**Col1- name, Col 2 to Col7 – marks in various subjects BEN 40 55 66 77 55 77

TOM 60 67 84 92 90 60

RAM 90 95 84 87 56 70

JIM 60 70 65 78 90 87

### Output:

**Run the program using the below command**

[root@localhost student]# gawk -f marks.awk marks.dat

NAME SUB-1 SUB-2 SUB-3 SUB-4 SUB-5 SUB-6 STATUS

BEN 40 55 66 77 55 77 FAIL TOM 60 67 84 92 90 60 PASS RAM 90 95 84

87 56 70 PASS JIM 60 70 65 78 90 87 PASS

### Result:

Thus, to implement text processing using Awk for examination results has been executed successfully.

### Ex. No.: 5

**System Calls Programming**

**Aim:** To experiment system calls using fork(), execlp() and pid() functions.

### Algorithm:

1. **Start**
   * Include the required header files (stdio.h and stdlib.h).

### Variable Declaration

* + Declare an integer variable pid to hold the process ID.

### Create a Process

* + Call the fork() function to create a new process. Store the return value in the pid variable:
    - If fork() returns:
      * -1: Forking failed (child process not created).
      * 0: Process is the child process.
      * Positive integer: Process is the parent process.

### Print Statement Executed Twice

* + Print the statement:

scss

Copy code

THIS LINE EXECUTED TWICE

(This line is executed by both parent and child processes after fork()).

### Check for Process Creation Failure

* + If pid == -1:
    - Print:

Copy code

CHILD PROCESS NOT CREATED

* + - Exit the program using exit(0).

### Child Process Execution

* + If pid == 0 (child process):
    - Print:
      * Process ID of the child process using getpid().
      * Parent process ID of the child process using getppid().

### Parent Process Execution

* + If pid > 0 (parent process):
    - Print:
      * Process ID of the parent process using getpid().
      * Parent's parent process ID using getppid().

### Final Print Statement

* + Print the statement: objectivec

Copy code

IT CAN BE EXECUTED TWICE

(This line is executed by both parent and child processes).

### End

**Program:**

#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <sys/types.h> #include <sys/wait.h>

int main() { int pid;

pid = fork();

printf(" ");

if (pid == -1) {

printf("\n Child process not created\n"); exit(0);

}

if (pid == 0) {

printf("\n Child Process ID --> %d \n", getpid()); printf("\n Child's Parent Process ID --> %d \n", getppid());

} else {

printf("\n Parent Process ID --> %d \n", getpid()); printf("\n Parent's Parent Process ID --> %d \n", getppid()); wait(NULL);

}

printf(" ");

return 0;

}

### Output:

Child Process ID --> 4568

Child's Parent Process ID --> 4567 Parent Process ID --> 4567

Parent's Parent Process ID --> 1234

### Result:

Thus, to implement system calls using fork(), exec(), getpid(), opendir(), and readdir() has been executed successfully.

**Ex. No.: 6a)**

## FIRST COME FIRST SERVE

### Aim:

To implement First-come First- serve (FCFS) scheduling technique

### Algorithm:

1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process. 6. Display the total waiting time, average waiting time, turnaround time

### Program Code:

#include <stdio.h>

void calculate\_fcfs(int n, int burst\_times[], int waiting\_times[], int turnaround\_times[]) { waiting\_times[0] = 0;

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

waiting\_times[i] = burst\_times[i - 1] + waiting\_times[i - 1];

}

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

turnaround\_times[i] = burst\_times[i] + waiting\_times[i];

}

}

void display\_results(char process\_names[][20], int burst\_times[], int waiting\_times[], int turnaround\_times[], int n) {

int total\_waiting\_time = 0;

int total\_turnaround\_time = 0;

printf("\nProcess Name Burst Time Waiting Time Turnaround Time\n"); for (int i = 0; i < n; i++) {

printf("%-15s %-12d %-14d %-18d\n", process\_names[i], burst\_times[i], waiting\_times[i], turnaround\_times[i]);

total\_waiting\_time += waiting\_times[i]; total\_turnaround\_time += turnaround\_times[i];

}

float avg\_waiting\_time = (float)total\_waiting\_time / n;

float avg\_turnaround\_time = (float)total\_turnaround\_time / n;

printf("\nTotal Waiting Time: %d", total\_waiting\_time); printf("\nAverage Waiting Time: %.2f", avg\_waiting\_time); printf("\nTotal Turnaround Time: %d", total\_turnaround\_time); printf("\nAverage Turnaround Time: %.2f \n", avg\_turnaround\_time);

}

int main() { int n;

printf("Enter the number of processes: "); scanf("%d", &n);

char process\_names[n][20];

int burst\_times[n], waiting\_times[n], turnaround\_times[n];

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

printf("\nEnter the name of process %d: ", i + 1); scanf("%s", process\_names[i]);

printf("Enter the burst time for %s: ", process\_names[i]); scanf("%d", &burst\_times[i]);

}

calculate\_fcfs(n, burst\_times, waiting\_times, turnaround\_times); display\_results(process\_names, burst\_times, waiting\_times, turnaround\_times, n);

return 0;

}

### Sample Output:

Enter the number of process:

3

Enter the burst time of the processes:

24 3 3

|  |  |  |  |
| --- | --- | --- | --- |
| Process | Burst Time | Waiting Time | Turn Around Time |
| 0 | 24 | 0 | 24 |
| 1 | 3 | 24 | 27 |
| 2 | 3 | 27 | 30 |

Average waiting time is: 17.0 Average Turn around Time is: 19.0

### Result:

Thus, to implement First Come First Serve (FCFS) scheduling algorithm has been executed successfully.

**Ex. No.: 6b)**

## SHORTEST JOB FIRST

### Aim:

To implement the Shortest Job First (SJF) scheduling technique

### Algorithm:

1. Declare the structure and its elements.
2. Get number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero. 5. Sort based on burst time of all processes in ascending order 6. Calculate the waiting time and turnaround time for each process. 7. Calculate the average waiting time and average turnaround time. 8. Display the results.

### Program Code:

#include <stdio.h> #include <string.h>

struct Process {

char processName[10]; int arrivalTime;

int burstTime; int waitingTime;

int turnaroundTime;

};

void sortByBurstTime(struct Process processes[], int n) { struct Process temp;

for (int i = 0; i < n - 1; i++) { for (int j = i + 1; j < n; j++) {

if (processes[i].burstTime > processes[j].burstTime) { temp = processes[i];

processes[i] = processes[j]; processes[j] = temp;

}

}

}

}

void calculateWaitingTime(struct Process processes[], int n) { processes[0].waitingTime = 0;

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

processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;

}

}

void calculateTurnaroundTime(struct Process processes[], int n) { for (int i = 0; i < n; i++) {

processes[i].turnaroundTime = processes[i].waitingTime + processes[i].burstTime;

}

}

void calculateAverageTimes(struct Process processes[], int n) { float totalWaitingTime = 0, totalTurnaroundTime = 0;

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

totalWaitingTime += processes[i].waitingTime; totalTurnaroundTime += processes[i].turnaroundTime;

}

printf("\nAverage waiting time is: %.1f", totalWaitingTime / n); printf("\nAverage Turnaround Time is: %.1f", totalTurnaroundTime / n);

}

int main() { int n;

printf("Enter the number of processes: "); scanf("%d", &n);

struct Process processes[n]; for (int i = 0; i < n; i++) {

printf("Enter process name for process %d: ", i + 1);

scanf("%s", processes[i].processName); printf("Enter arrival time for process %d: ", i + 1); scanf("%d", &processes[i].arrivalTime); printf("Enter burst time for process %d: ", i + 1); scanf("%d", &processes[i].burstTime);

}

sortByBurstTime(processes, n); for (int i = 0; i < n; i++) {

processes[i].waitingTime = 0;

processes[i].turnaroundTime = 0;

}

calculateWaitingTime(processes, n); calculateTurnaroundTime(processes, n);

printf("\nProcess Name\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time"); for (int i = 0; i < n; i++) {

printf("\n%s\t\t%d\t\t%d\t\t%d\t\t%d", processes[i].processName, processes[i].arrivalTime, processes[i].burstTime, processes[i].waitingTime, processes[i].turnaroundTime);

}

calculateAverageTimes(processes, n);

return 0;

}

### Sample Output:

Enter the number of process:

4

Enter the burst time of the processes:

8 4 9 5

Process Burst Time Waiting Time Turn Around Time

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | 4 | 0 | 4 |
| 4 | 5 | 4 | 9 |
| 1 | 8 | 9 | 17 |
| 3 | 9 | 17 | 26 |

Average waiting time is: 7.5 Average Turn Around Time is: 13.0

### Result:

Thus, to implement Shortest Job First (SJF) scheduling algorithm has been executed successfully.

**Ex. No.: 6c)**

## PRIORITY SCHEDULING

### Aim:

To implement priority scheduling technique

### Algorithm:

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of process.
3. Sort based on burst time of all processes in ascending order based priority 4. Calculate the total waiting time and total turnaround time for each process 5. Display the process name & burst time for each process.

6. Display the total waiting time, average waiting time, turnaround time

### Program Code:

#include <stdio.h> #include <string.h>

struct Process {

char processName[10]; int burstTime;

int remainingTime; int priority;

int waitingTime;

int turnaroundTime; int arrivalTime;

};

void sortByArrivalTime(struct Process processes[], int n) { struct Process temp;

for (int i = 0; i < n - 1; i++) { for (int j = i + 1; j < n; j++) {

if (processes[i].arrivalTime > processes[j].arrivalTime) { temp = processes[i];

processes[i] = processes[j]; processes[j] = temp;

}

}

}

}

void calculateWaitingAndTurnaroundTime(struct Process processes[], int n) { int t = 0;

int completed = 0; int prevProcess = -1; int finished[n];

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

finished[i] = 0;

processes[i].remainingTime = processes[i].burstTime;

}

while (completed < n) { int idx = -1;

int highestPriority = -1;

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

if (processes[i].arrivalTime <= t && !finished[i] && processes[i].priority > highestPriority) { highestPriority = processes[i].priority;

idx = i;

}

}

if (idx != -1) { processes[idx].remainingTime--; t++;

if (processes[idx].remainingTime == 0) { finished[idx] = 1;

completed++;

processes[idx].waitingTime = t - processes[idx].burstTime - processes[idx].arrivalTime; processes[idx].turnaroundTime = processes[idx].waitingTime + processes[idx].burstTime;

}

} else { t++;

}

}

}

void calculateAverageTimes(struct Process processes[], int n) { float totalWaitingTime = 0, totalTurnaroundTime = 0;

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

totalWaitingTime += processes[i].waitingTime; totalTurnaroundTime += processes[i].turnaroundTime;

}

printf("\nAverage Waiting Time: %.2f", totalWaitingTime / n); printf("\nAverage Turnaround Time: %.2f\n", totalTurnaroundTime / n);

}

int main() { int n;

printf("Enter the number of processes: "); scanf("%d", &n);

struct Process processes[n]; for (int i = 0; i < n; i++) {

printf("Enter process name for process %d: ", i + 1); scanf("%s", processes[i].processName);

printf("Enter priority for process %d (higher number = higher priority): ", i + 1); scanf("%d", &processes[i].priority);

printf("Enter burst time for process %d: ", i + 1); scanf("%d", &processes[i].burstTime); printf("Enter arrival time for process %d: ", i + 1); scanf("%d", &processes[i].arrivalTime);

}

sortByArrivalTime(processes, n); calculateWaitingAndTurnaroundTime(processes, n);

printf("\nProcess Name\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time"); for (int i = 0; i < n; i++) {

printf("\n%s\t\t%d\t\t%d\t\t%d\t\t%d", processes[i].processName, processes[i].arrivalTime, processes[i].burstTime, processes[i].waitingTime, processes[i].turnaroundTime);

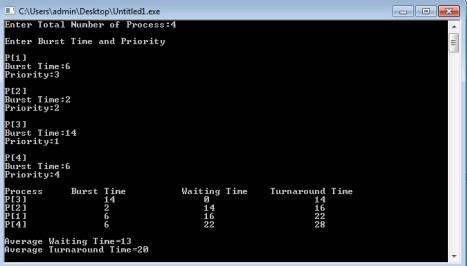
}

calculateAverageTimes(processes, n);

return 0;

}

### Sample Output:

****

**Result:**

Thus, to implement Priority Scheduling algorithm has been executed successfully.

**Ex. No.: 6d)**

## ROUND ROBIN SCHEDULING

### Aim:

To implement the Round Robin (RR) scheduling technique

### Algorithm:

1. Declare the structure and its elements.
2. Get number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array **rem\_bt[]** to keep track of remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0. 6. Initialize time : t = 0
6. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.
   1. If rem\_bt[i] > quantum
      1. t = t + quantum
      2. bt\_rem[i] -= quantum;
   2. Else // Last cycle for this process
      1. t = t + bt\_rem[i];
      2. wt[i] = t - bt[i]
      3. bt\_rem[i] = 0; // This process is over
7. Calculate the waiting time and turnaround time for each process.
8. Calculate the average waiting time and average turnaround time.
9. Display the results.

### Program Code:

#include <stdio.h>

void findWaitingTime(int processes[], int n, int at[], int bt[], int quantum, int wt[]) { int rem\_bt[n];

for (int i = 0; i < n; i++) rem\_bt[i] = bt[i];

int t = 0; while (1) {

int done = 1;

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

if (rem\_bt[i] > 0 && at[i] <= t) { done = 0;

if (rem\_bt[i] > quantum) { t += quantum; rem\_bt[i] -= quantum;

} else {

t += rem\_bt[i];

wt[i] = t - at[i] - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done) break;

}

}

void findTurnaroundTime(int processes[], int n, int bt[], int wt[], int tat[]) { for (int i = 0; i < n; i++)

tat[i] = bt[i] + wt[i];

}

void findAvgTime(int processes[], int n, int at[], int bt[], int quantum) { int wt[n], tat[n];

findWaitingTime(processes, n, at, bt, quantum, wt); findTurnaroundTime(processes, n, bt, wt, tat);

printf("\nProcess \tArrival Time \tBurst Time \tWaiting Time \tTurnaround Time\n"); float total\_wt = 0, total\_tat = 0;

for (int i = 0; i < n; i++) { total\_wt += wt[i]; total\_tat += tat[i];

printf("P%d \t\t%d \t\t%d \t\t%d \t\t%d\n", i + 1, at[i], bt[i], wt[i], tat[i]);

}

printf("\nAverage Waiting Time = %.2f", total\_wt / n); printf("\nAverage Turnaround Time = %.2f\n", total\_tat / n);

}

int main() {

int n, quantum;

printf("Enter the number of processes: "); scanf("%d", &n);

int processes[n], at[n], bt[n]; printf("Enter time quantum: "); scanf("%d", &quantum);

for (int i = 0; i < n; i++) { processes[i] = i + 1;

printf("Enter arrival time for process P%d: ", i + 1); scanf("%d", &at[i]);

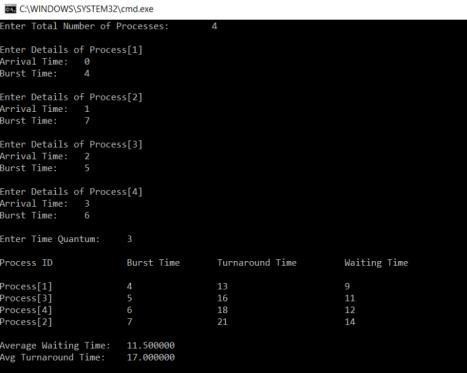
printf("Enter burst time for process P%d: ", i + 1); scanf("%d", &bt[i]);

}

findAvgTime(processes, n, at, bt, quantum); return 0;

}

### Sample Output:

****

**Result:**

Thus, to implement Round Robin scheduling algorithm has been executed successfully.

### Ex. No.: 7 IPC USING SHARED MEMORY

**Aim:**

To write a C program to do Inter Process Communication (IPC) using shared memory between sender process and receiver process.

### Algorithm: sender

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat
4. Write a string to the shared memory segment using sprintf
5. Set delay using sleep
6. Detach shared memory segment using shmdt

### receiver

1. Set the size of the shared memory segment
2. Allocate the shared memory segment using shmget
3. Attach the shared memory segment using shmat
4. Print the shared memory contents sent by the sender process.
5. Detach shared memory segment using shmdt

### Program Code: sender.c:

#include <sys/types.h>

#include <sys/ipc.h> #include <sys/shm.h> #include <stdio.h> #include <stdlib.h> #include <unistd.h>

#define SharedMemSize 50 void main() {

int shmid;

key\_t key = 5677; char \*shared\_memory;

if ((shmid = shmget(key, SharedMemSize, IPC\_CREAT | 0666)) < 0) { perror("shmget");

exit(1);

}

if ((shared\_memory = shmat(shmid, NULL, 0)) == (char \*) -1) { perror("shmat");

exit(1);

}

sprintf(shared\_memory, "Welcome to Shared Memory"); sleep(5);

exit(0);

}

### receiver.c:

#include <sys/types.h> #include <sys/ipc.h> #include <sys/shm.h> #include <stdio.h> #include <stdlib.h>

#define SharedMemSize 50 void main() {

int shmid;

key\_t key = 5677; char \*shared\_memory;

if ((shmid = shmget(key, SharedMemSize, 0666)) < 0) { perror("shmget");

exit(1);

}

if ((shared\_memory = shmat(shmid, NULL, 0)) == (char \*) -1) { perror("shmat");

exit(1);

}

printf("Message Received: %s\n", shared\_memory); exit(0);

}

### Sample Output Terminal 1

[root@localhost student]# gcc sender.c -o sender [root@localhost student]# ./sender

### Terminal 2

[root@localhost student]# gcc receiver.c -o receiver [root@localhost student]# ./receiver

Message Received: Welcome to Shared Memory [root@localhost student]#

### Result:

Thus, to implement inter-process communication using shared memory has been executed successfully.

**Ex. No.: 8**

## PRODUCER CONSUMER USING SEMAPHORES

**Aim:** To write a program to implement solution to producer consumer problem using semaphores.

### Algorithm:

1. Initialize semaphore empty, full and mutex.
2. Create two threads- producer thread and consumer thread.
3. Wait for target thread termination.
4. Call sem\_wait on empty semaphore followed by mutex semaphore before entry into critical section.
5. Produce/Consume the item in critical section.
6. Call sem\_post on mutex semaphore followed by full semaphore
7. before exiting critical section.
8. Allow the other thread to enter its critical section.
9. Terminate after looping ten times in producer and consumer Threads each.

### Program Code:

#include <stdio.h> #include <pthread.h> #include <semaphore.h> #include <unistd.h>

#define BUFFER\_SIZE 3

sem\_t empty, full, mutex; int buffer[BUFFER\_SIZE]; int count = 0;

void \*producer(void \*arg) { int item = 0;

while (1) { int choice;

printf("\n1. Producer\n2. Consumer\n3. Exit\nEnter your choice: "); scanf("%d", &choice);

if (choice == 1) {

if (count == BUFFER\_SIZE) { printf("Buffer is full!!\n"); continue;

}

sem\_wait(&empty); sem\_wait(&mutex);

buffer[count++] = ++item;

printf("Producer produces the item %d\n", item); sem\_post(&mutex);

sem\_post(&full);

} else if (choice == 2) { if (count == 0) {

printf("Buffer is empty!!\n"); continue;

}

sem\_wait(&full); sem\_wait(&mutex);

printf("Consumer consumes item %d\n", buffer[--count]); sem\_post(&mutex);

sem\_post(&empty);

} else if (choice == 3) { break;

} else {

printf("Invalid choice!\n");

}

}

return NULL;

}

int main() { pthread\_t prod;

sem\_init(&empty, 0, BUFFER\_SIZE);

sem\_init(&full, 0, 0);

sem\_init(&mutex, 0, 1);

pthread\_create(&prod, NULL, producer, NULL); pthread\_join(prod, NULL);

sem\_destroy(&empty); sem\_destroy(&full); sem\_destroy(&mutex);

return 0;

}

### Sample Output:

1. Producer
2. Consumer
3. Exit

Enter your choice:1 Producer produces the item 1 Enter your choice:2 Consumer consumes item

1 Enter your choice:2 Buffer is empty!!

Enter your choice:1 Producer produces the item 1 Enter your choice:1 Producer produces the item 2 Enter your choice:1 Producer produces the item 3 Enter your choice:1

Buffer is full!! Enter your choice:3

### Result:

Thus, to implement the producer-consumer problem using semaphores has been executed successfully.

**Ex. No.: 9**

## DEADLOCK AVOIDANCE

### Aim:

To find out a safe sequence using Banker’s algorithm for deadlock avoidance.

### Algorithm:

1. Initialize work=available and finish[i]=false for all values of i
2. Find an i such that both:

finish[i]=false and Needi<= work

1. If no such i exists go to step 6
2. Compute work=work+allocationi
3. Assign finish[i] to true and go to step 2
4. If finish[i]==true for all i, then print safe sequence
5. Else print there is no safe sequence

### Program Code:

#include <stdio.h> #include <stdbool.h>

#define P 5

#define R 3

void findSafeSequence(int processes[], int available[], int max[][R], int allocation[][R]) { int need[P][R];

bool finish[P] = {false}; int safeSequence[P];

int work[R];

for (int i = 0; i < P; i++) { for (int j = 0; j < R; j++) {

need[i][j] = max[i][j] - allocation[i][j];

}

}

for (int i = 0; i < R; i++) { work[i] = available[i];

}

int count = 0; while (count < P) {

bool found = false;

for (int i = 0; i < P; i++) { if (!finish[i]) {

bool canAllocate = true; for (int j = 0; j < R; j++) {

if (need[i][j] > work[j]) { canAllocate = false; break;

}

}

if (canAllocate) {

for (int j = 0; j < R; j++) { work[j] += allocation[i][j];

}

safeSequence[count++] = processes[i]; finish[i] = true;

found = true;

}

}

}

if (!found) {

printf("No safe sequence exists!\n"); return;

}

}

printf("The SAFE Sequence is: "); for (int i = 0; i < P; i++) {

printf("P%d", safeSequence[i]); if (i != P - 1) printf(" -> ");

}

printf("\n");

}

int main() {

int processes[P] = {0, 1, 2, 3, 4};

int available[R] = {3, 3, 2}; int max[P][R] = {

{7, 5, 3},

{3, 2, 2},

{9, 0, 2},

{2, 2, 2},

{4, 3, 3}

};

int allocation[P][R] = {

{0, 1, 0},

{2, 0, 0},

{3, 0, 2},

{2, 1, 1},

{0, 0, 2}

};

findSafeSequence(processes, available, max, allocation); return 0;

}

### Sample Output:

The SAFE Sequence is

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

### Result:

Thus, to implement Banker's algorithm for deadlock avoidance has been executed successfully.

**Ex. No.: 10a)**

## BEST FIT

### Aim:

To implement Best Fit memory allocation technique using Python.

### Algorithm:

1. Input memory blocks and processes with sizes
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process
4. If found then assign it to the current process.
5. If not found then leave that process and keep checking the further processes.

### Program Code:

#include <stdio.h> int main() {

int blockSize[10], processSize[10];

int blockCount, processCount; int allocation[10];

printf("Enter number of memory blocks: "); scanf("%d", &blockCount);

printf("Enter size of each memory block:\n"); for (int i = 0; i < blockCount; i++) {

printf("Block %d: ", i + 1);

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

}

printf("\nEnter number of processes: "); scanf("%d", &processCount); printf("Enter size of each process:\n"); for (int i = 0; i < processCount; i++) {

printf("Process %d: ", i + 1); scanf("%d", &processSize[i]); allocation[i] = -1;

}

for (int i = 0; i < processCount; i++) { int bestIdx = -1;

for (int j = 0; j < blockCount; j++) {

if (blockSize[j] >= processSize[i]) {

if (bestIdx == -1 || blockSize[j] < blockSize[bestIdx]) bestIdx = j;

}

}

if (bestIdx != -1) { allocation[i] = bestIdx;

blockSize[bestIdx] -= processSize[i];

}

}

printf("\n%-12s %-15s %-10s\n", "Process No.", "Process Size", "Block No."); for (int i = 0; i < processCount; i++) {

printf("%-12d %-15d ", i + 1, processSize[i]); if (allocation[i] != -1)

printf("%-10d\n", allocation[i] + 1); else

printf("%-10s\n", "Not Allocated");

}

return 0;

}

### Sample Output:

|  |  |  |
| --- | --- | --- |
| Process No. | Process Size | Block no. |
| 1 | 212 | 4 |
| 2 | 417 | 2 |
| 3 | 112 | 3 |
| 4 | 426 | 5 |

**Result:**

Thus, to implement Best Fit memory allocation technique has been executed successfully.

**Ex. No.: 10b)**

## FIRST FIT

### Aim:

To write a C program for implementation memory allocation methods for fixed partition

using first fit.

### Algorithm:

1. Define the max as 25.

2: Declare the variable frag[max],b[max],f[max],i,j,nb,nf,temp, highest=0, bf[max],ff[max]. 3: Get the number of blocks,files,size of the blocks using for loop.

4: In for loop check bf[j]!=1, if so temp=b[j]-f[i] 5: Check highest

### Program Code:

#include <stdio.h> #define MAX 25

int main() {

int nb, nf, i, j;

int frag[MAX], f[MAX], b[MAX], ff[MAX], bb[MAX], allocation[MAX];

printf("Number of Blocks: "); scanf("%d", &nb);

printf("Number of Files: "); scanf("%d", &nf);

printf("Blocks sizes:\n"); for (i = 0; i < nb; i++) {

printf("Block %d: ", i + 1);

scanf("%d", &b[i]); bb[i] = 0;

}

printf("Files sizes:\n"); for (i = 0; i < nf; i++) {

printf("File %d: ", i + 1);

scanf("%d", &f[i]); ff[i] = 0;

}

for (i = 0; i < nf; i++) { for (j = 0; j < nb; j++) {

if (bb[j] == 0 && b[j] >= f[i]) {

frag[i] = b[j] - f[i];

ff[i] = 1;

bb[j] = 1; allocation[i] = j; break;

}

}

}

printf("\n%-8s %-12s %-10s %-12s %-10s\n", "File No", "File Size", "Block No", "Block Size", "Fragment");

for (i = 0; i < nf; i++) { if (ff[i] == 1) {

int blockindex = allocation[i];

printf("%-8d %-12d %-10d %-12d %-10d\n",

i + 1, f[i], blockindex + 1, b[blockindex], frag[i]);

} else {

printf("%-8d %-12d %-10s\n", i + 1, f[i], "Not Allocated");

}

}

return 0;

}

**Sample Output:**

****

### Result:

Thus, to implement First Fit memory allocation technique has been executed successfully.

**Ex. No.: 11a)**

## FIFO PAGE REPLACEMENT

### Aim:

To find out the number of page faults that occur using First-in First-out (FIFO) page

replacement technique.

### Algorithm:

1. Declare the size with respect to page length
2. Check the need of replacement from the page to memory
3. Check the need of replacement from old page to new page in memory 4. Form a queue to hold all pages
4. Insert the page require memory into the queue
5. Check for bad replacement and page fault
6. Get the number of processes to be inserted
7. Display the values

### Program Code:

#include <stdio.h>

int isPageInFrame(int page, int frame[], int size) { for (int i = 0; i < size; i++) {

if (frame[i] == page) return 1;

}

return 0;

}

void printFrame(int frame[], int size) { for (int i = 0; i < size; i++) {

if (frame[i] != -1)

printf("%d ", frame[i]); else

printf("- ");

}

printf("\n");

}

int main() {

int referenceString[100], frame[10], n, frameSize; int pageFaults = 0, index = 0;

printf("Enter the size of reference string: "); scanf("%d", &n);

for (int i = 0; i < n; i++) { printf("Enter [%2d] : ", i + 1); scanf("%d", &referenceString[i]);

}

printf("Enter page frame size : ");

scanf("%d", &frameSize);

for (int i = 0; i < frameSize; i++)

frame[i] = -1; // Initialize frames with -1

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

printf("%d -> ", referenceString[i]);

if (!isPageInFrame(referenceString[i], frame, frameSize)) { frame[index] = referenceString[i];

index = (index + 1) % frameSize; pageFaults++;

printFrame(frame, frameSize);

} else {

printf("No Page Fault\n");

}

}

printf("\nTotal page faults: %d\n", pageFaults); return 0;

}

### Sample Output:

[root@localhost student]# python fifo.py

Enter the size of reference string: 20 Enter [ 1] : 7

Enter [ 2] : 0

Enter [ 3] : 1

Enter [ 4] : 2

Enter [ 5] : 0

Enter [ 6] : 3

Enter [ 7] : 0

Enter [ 8] : 4

Enter [ 9] : 2

Enter [10] : 3

Enter [11] : 0

Enter [12] : 3

Enter [13] : 2

Enter [14] : 1

Enter [15] : 2

Enter [16] : 0

Enter [17] : 1

Enter [18] : 7

Enter [19] : 0

Enter [20] : 1

Enter page frame size : 3

7 -> 7 - -

0 -> 7 0 -

1 -> 7 0 1

2 -> 2 0 1

0 -> No Page Fault

3 -> 2 3 1

0 -> 2 3 0

4 -> 4 3 0

2 -> 4 2 0

3 -> 4 2 3

0 -> 0 2 3

3 -> No Page Fault 2 -> No Page Fault

1 -> 0 1 3

2 -> 0 1 2

0 -> No Page Fault

1 -> No Page Fault 7 -> 7 1 2

0 -> 7 0 2

1 -> 7 0 1

Total page faults: 15. [root@localhost student]#

### Result :

Thus, to implement FIFO page replacement algorithm has been executed successfully.

### Ex. No.: 11b)

**Aim:**

**LRU**

To write a c program to implement LRU page replacement algorithm.

### Algorithm:

1: Start the process 2: Declare the size

3: Get the number of pages to be inserted 4: Get the value

5: Declare counter and stack

6: Select the least recently used page by counter value 7: Stack them according the selection.

8: Display the values 9: Stop the process

### Program Code:

#include <stdio.h> int main() {

int frames, pages, i, j, k, position, fault = 0, hit; int memory[10], time[10], ref[30], count = 0;

printf("Enter number of frames: "); scanf("%d", &frames); printf("Enter number of pages: "); scanf("%d", &pages); printf("Enter reference string: "); for (i = 0; i < pages; i++) {

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

}

for (i = 0; i < frames; i++) { memory[i] = -1;

}

for (i = 0; i < pages; i++) { hit = 0;

for (j = 0; j < frames; j++) { if (memory[j] == ref[i]) {

hit = 1;

time[j] = count++; break;

}

}

if (hit == 0) {

int empty = -1;

for (j = 0; j < frames; j++) { if (memory[j] == -1) {

empty = j;

break;

}

}

if (empty != -1) { memory[empty] = ref[i]; time[empty] = count++;

} else {

int lru = 0;

for (j = 1; j < frames; j++) { if (time[j] < time[lru]) {

lru = j;

}

}

memory[lru] = ref[i]; time[lru] = count++;

}

fault++;

}

for (k = 0; k < frames; k++) { if (memory[k] != -1)

printf("%d ", memory[k]); else

printf("- ");

}

printf("\n");

}

printf("Total Page Faults = %d\n", fault); return 0;

}

### Sample Output :

Enter number of frames: 3 Enter number of pages: 6

Enter reference string: 5 7 5 6 7 3

5 -1 -1

5 7 -1

5 7 -1

5 7 6

5 7 6

3 7 6

Total Page Faults = 4

### Result:

Thus, to implement LRU page replacement algorithm has been executed successfully.

**Ex. No.: 11c)**

**Optimal**

**Aim:**

To write a c program to implement Optimal page replacement algorithm.

## ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of pages to be inserted
4. Get the value
5. Declare counter and stack
6. Select the least frequently used page by counter value
7. Stack them according the selection.
8. Display the values
9. Stop the process

## PROGRAM:

#include <stdio.h>

int findOptimal(int pages[], int frames[], int n, int index, int frameCount) { int farthest = index;

int result = -1;

for (int i = 0; i < frameCount; i++) { int j;

for (j = index; j < n; j++) {

if (frames[i] == pages[j]) { if (j > farthest) {

farthest = j;

result = i;

}

break;

}

}

if (j == n) return i;

}

return (result == -1) ? 0 : result;

}

int main() {

int frameCount, nPages; printf("Enter number of frames: "); scanf("%d", &frameCount); printf("Enter number of pages: "); scanf("%d", &nPages);

int pages[nPages], frames[frameCount]; int count = 0, pageFaults = 0; printf("Enter the page reference string: "); for (int i = 0; i < nPages; i++) {

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

}

for (int i = 0; i < frameCount; i++) frames[i] = -1;

printf("\nPage Frames after each reference:\n"); for (int i = 0; i < nPages; i++) {

int flag = 0;

for (int j = 0; j < frameCount; j++) { if (frames[j] == pages[i]) {

flag = 1; break;

}

}

if (!flag) {

if (count < frameCount) { frames[count++] = pages[i];

} else {

int idx = findOptimal(pages, frames, nPages, i + 1, frameCount); frames[idx] = pages[i];

}

pageFaults++;

}

for (int j = 0; j < frameCount; j++) { if (frames[j] == -1)

printf("- "); else

printf("%d ", frames[j]);

}

printf("\n");

}

printf("\nTotal Page Faults = %d\n", pageFaults); return 0;

}

### Output:

Number of frames: 3 Number of pages: 12

Reference string: 7 0 1 2 0 3 0 4 2 3 0 3 Page Frames after each reference:

7 - -

7 0 -

7 0 1

2 0 1

2 0 1

2 3 1

2 3 0

4 3 0

4 2 0

4 2 3

0 2 3

0 2 3

Total Page Faults = 9

### Result:

Thus, to implement Optimal page replacement algorithm has been executed successfully.

**Ex. No.: 12**

**File Organization Technique- Single and Two level directory**

## AIM:

To implement File Organization Structures in C are

1. Single Level Directory
2. Two-Level Directory
3. Hierarchical Directory Structure
4. Directed Acyclic Graph Structure

### Single Level Directory ALGORITHM

* 1. Start
  2. Declare the number, names and size of the directories and file names.
  3. Get the values for the declared variables.
  4. Display the files that are available in the directories.
  5. Stop.

## PROGRAM:

#include <stdio.h> int main() {

int n;

char files[20][20];

printf("Enter the Number of files: "); scanf("%d", &n);

for (int i = 0; i < n; i++) { printf("Enter the file %d: ", i + 1); scanf("%s", files[i]);

printf("\n Root Directory\n");

printf(" |\n"); if (i == 0) {

printf(" [%s]\n", files[0]);

} else {

printf(" /");

for (int j = 1; j < i; j++) { printf(" ");

}

printf(" \\\n");

printf(" ");

for (int j = 0; j <= i; j++) { printf("[%s] ", files[j]);

}

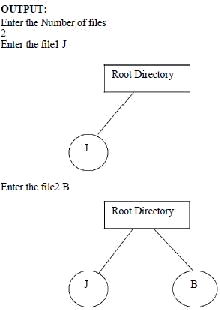
printf("\n");

}

printf("\n");

}

return 0;



1. **Two-level directory Structure**

## ALGORITHM:

* 1. Start
  2. Declare the number, names and size of the directories and subdirectories and file names.
  3. Get the values for the declared variables.
  4. Display the files that are available in the directories and subdirectories.
  5. Stop.

## PROGRAM:

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

struct Node { char name[20]; int isFile;

int childCount;

struct Node\* children[10];

};

struct Node\* createNode(char name[], int isFile) {

struct Node\* node = (struct Node\*)malloc(sizeof(struct Node)); strcpy(node->name, name);

node->isFile = isFile; node->childCount = 0; return node;

}

void createStructure(struct Node\* parent) { int count;

printf("Enter the name of dir/file (under %s): ", parent->name); scanf("%s", parent->name);

if (!parent->isFile) {

printf("How many %s (for %s): ", "children", parent->name); scanf("%d", &count);

parent->childCount = count;

for (int i = 0; i < count; i++) { char type[10];

printf("Is child %d a file or dir? (f/d): ", i + 1); scanf("%s", type);

int isFile = (type[0] == 'f');

parent->children[i] = createNode("", isFile); createStructure(parent->children[i]);

}

}

}

void displayStructure(struct Node\* node, int depth) { for (int i = 0; i < depth; i++) printf(" ");

printf("|-- %s\n", node->name);

for (int i = 0; i < node->childCount; i++) { displayStructure(node->children[i], depth + 1);

}

}

int main() {

struct Node\* root = createNode("NULL", 0); createStructure(root);

printf("\n--- HIERARCHICAL DIRECTORY STRUCTURE ---\n");

displayStructure(root,0); return 0;

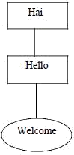
}

### Sample Output:

Enter the name of dir/file(under null): Hai How many users(for Hai):1

Enter name of dir/file(under Hai):Hello How many files(for Hello):1

Enter name of dir/file(under Hello):welcome



### Result:

Thus, to implement file organization techniques including single and two-level directory structures has been executed successfully.