**MANUAL FOR OPERATING SYSTEMS LAB**

**of the**

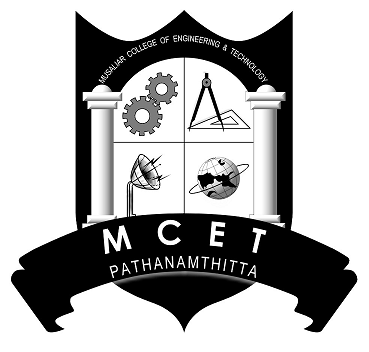
**B.Tech DEGREE COURSE**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**under**

**APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY**



**MUSALIAR COLLEGE OF ENGINEERING AND TECHNOLOGY, PATHANAMTHITTA**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course No | Course Name | L-T-P | Credits | Year of Introduction |
| CST 206 | OPERATING SYSTEMS LAB | 0-0-3 | 2 | 2019 |

## 0

## LIST OF EXPERIMENTS

|  |  |  |
| --- | --- | --- |
| Experiment no: | Description | Hours Allotted |
| 1 | BASIC COMMANDS IN LINUX | 2 |
| 2 | SHELL PROGRAMMINGEVEN OR ODDBIGGEST OF TWO NUMBERSBIGGEST OF THREE NUMBERSFACTORIAL OF NUMBERFIBONACCI SERIES | 3 |
| 3 | SYSTEM CALLS OF UNIX OPERATING SYSTEM | 3 |
| 4 | I/O SYSTEM CALLS OF UNIX OPERATING SYSTEM | 3 |
| 5 | C PROGRAMS TO SIMULATE UNIX COMMANDS LIKE LS, GREP. | 2 |
| 6 | FCFS (FIRST COME FIRST SERVE) CPU SCHEDULINGSJF (SHORTEST JOB FIRST) CPU SCHEDULING ALGORITHM | 2 |
| 7 | PRIORITY CPU SCHEDULING ALGORITHMROUND ROBIN CPU SCHEDULING ALGORITHM | 2 |
| 8 | DEVELOPING APPLICATION USING INTER PROCESS COMMUNICATION | 2 |
| 9 | INTERPROCESS COMMUNICATION USING MESSAGE PASSING. | 2 |
| 10 | MEMORY MANAGEMENT SCHEME-I | 2 |
| 11 | MEMORY MANAGEMENT SCHEME II | 2 |
| 12 | FILE ALLOCATION TECHNIQUE-CONTIGUOUS | 2 |
| 13 | IMPLEMENTATION BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE | 1 |

## SYLLABUS

## 

## INTRODUCTION TO SHELL PROGRAMMING

A **Shell** provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

Shell Prompt

The prompt, **$**, which is called the **command prompt**, is issued by the shell. While the prompt is displayed, you can type a command.

Shell reads your input after you press **Enter**. It determines the command you want executed by looking at the first word of your input. A word is an unbroken set of characters. Spaces and tabs separate words.

Following is a simple example of the **date** command, which displays the current date and time −

$date

Thu Jun 25 08:30:19 MST 2009

You can customize your command prompt using the environment variable PS1 explained in the Environment tutorial.

Shell Types

In Unix, there are two major types of shells −

* **Bourne shell** − If you are using a Bourne-type shell, the **$** character is the default prompt.
* **C shell** − If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories −

* Bourne shell (sh)
* Korn shell (ksh)
* Bourne Again shell (bash)
* POSIX shell (sh)

The different C-type shells follow −

* C shell (csh)
* TENEX/TOPS C shell (tcsh)

The original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as **/bin/sh** on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.

In this chapter, we are going to cover most of the Shell concepts that are based on the Borne Shell.

Shell Scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by **#** sign, describing the steps.

There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, and variables to read and store data, and the script may include functions

In Unix-like operating systems, the **chmod** command is used to change the access mode of a file. The name is an abbreviation of **change mode**.

**Syntax :**

chmod [reference][operator][mode] file...

The references are used to distinguish the users to whom the permissions apply i.e. they are list of letters that specifies whom to give permissions. The references are represented by one or more of the following letters:

|  |  |  |
| --- | --- | --- |
| Reference | Class | Description |
| u | owner | file's owner |
| g | group | users who are members ofthe file's group |
| o | others | users who are neither the file's owner nor members of the file's group |
| a | all | All three of the above, same as ugo |

The operator is used to specify how the modes of a file should be adjusted. The following operators are accepted:

Operator Description

+ Adds the specified modes to the specified classes

- Removes the specified modes from the specified classes

= The modes specified are to be made the exact modes for the specified classes

The modes indicate which permissions are to be granted or removed from the specified classes. There are three basic modes which correspond to the basic permissions:

r Permission to read the file.

w Permission to write (or delete) the file. x Permission to execute the file, or, in

the case of a directory, search it.

Types of permissions which we will be changing using chmod command :

In linux terminal, to see all the permissions to different files, type ls -l command which lists the files in the working directory in long format

## EX.NO:1 BASIC COMMANDS IN LINUX

AIM:

To Study the basic commands in Linux.

COMMANDS:

1. Task : To display system date and time. Syntax: Date

Explanation: This command displays the current date and time on the screen.

1. Task: To display current month. Syntax: Date +%m

Explanation: This command displays the current month on the screen.

1. Task : To d[isplay the name of the current month.](http://csetube.weebly.com/) Syntax: Da[te +%h](http://csetube.weebly.com/)

Explana[tion: This command displays name of the current month on the screen.](http://csetube.weebly.com/)

1. Task: To [display current system date.](http://csetube.weebly.com/) Syntax: Dat[e +%d](http://csetube.weebly.com/)

Explana[tion: This command displays the current system date on the screen.](http://csetube.weebly.com/)

1. Task: To [display current year.](http://csetube.weebly.com/) Syntax: Dat[e +%y](http://csetube.weebly.com/)

Explana[tion: This command displays the current year on the screen.](http://csetube.weebly.com/)

1. Task: To [display current system time.](http://csetube.weebly.com/) Syntax: Dat[e +%H](http://csetube.weebly.com/)

Explanation: This command displays the current system time on the screen.

1. Task: To display current system time in minutes. Syntax: Date +%m

Explanation: This command displays the current system time in minutes on the screen.

1. Task: To display current system time in seconds. Syntax: Date +%s

Explanation: This command displays the current system time in seconds on the screen.

1. Task: To display the calendar of current month. Syntax: cal

Explanation: This command displays the current month calendar on the screen.

1. Task: To display user defined message.

Syntax: echo ‘message’

Explanation: This command displays the message.

11. Task: To display the present working directory.

Syntax: pwd

Explanation: This command displays the path of the present working directory.

12. Task: To navigate through linux files and directories.

Syntax: cd file name

Explanation: **cd ..** (with two dots) to move one directory up

**cd** to go straight to the home folder

**cd-** (with a hyphen) to move to your previous directory

13.Task: To view the contents of a directory.

Syntax: ls path of folder

Explanation: **ls -R** will list all the files in the sub-directories as well

**ls -a** will show the hidden files

**ls -al** will list the files and directories with detailed information like the permissions, size, owner, etc.

14.Task: To list the contents of a file

Syntax: cat file.txt

Explanation: **cat > filename** creates a new file

**cat filename1 filename2>filename3** joins two files (1 and 2) and stores the output of them in a new file (3)

to convert a file to upper or lower case use, **cat filename | tr a-z A-Z >output.txt**

SHELL PROGRAMMING

Ex.No.2a EVEN OR ODD

## AIM:

To write a program to find whether a number is even or odd .

## ALGORITHM:

STEP 1: Read the input number.

STEP 2: Perform modular division on input number by 2. STEP 3: If remainder is 0 print the number is even.

STEP 4: Else print number is odd. STEP 5: Stop the program.

## PROGRAM

echo "enter the [number"](http://csetube.weebly.com/) read num

echo "enter the [number"](http://csetube.weebly.com/) read num

if [ `expr $num [% 2` -eq 0 ]](http://csetube.weebly.com/) then

echo "number [is even"](http://csetube.weebly.com/) else

echo "number [is odd"](http://csetube.weebly.com/) fi

**OUTPUT:**

enter the number: 5 the number is odd. **RESULT**:

Thus the program has been executed successfully.

Ex.No.2b **BIGGEST OF TWO NUMBERS**

## AIM :

To write a program to find biggest in two numbers.

## ALGORITHM :

STEP 1: Read The Two Numbers.

STEP 2: If Value Of A Is Greater Than B Is Big. STEP 3: Else Print B Is Big.

STEP [4: Stop The Program.](http://csetube.weebly.com/)

## PROGRAM:

echo "enter the [number"](http://csetube.weebly.com/) read a b

if [ $a -gt $b ] then

echo "A is big["](http://csetube.weebly.com/) else

echo "B is big["](http://csetube.weebly.com/) fi

## OUTPUT:

Enter The Two Number:

23 67

B is Big.

## RESULT:

Thus the program has been executed successfully.

Ex.No.2c **BIGGEST OF THREE NUMBERS**

## AIM:

To Write a Program to Find Biggest In Three Numbers.

## ALGORITHM:

STEP 1: Read The Three Numbers.

STEP 2: If A Is Greater Than B And A Is Greater Than C Then Print A Is Big. STEP 3: Else If B is greater Than C Then C Is Big.

STEP 4: Else Print C Is Big. STEP 5: Stop The Program.

## PROGRAM:

echo "enter thr[ee numbers"](http://csetube.weebly.com/) read a b c

if [ $a -gt $b ] [&& [ $a -gt $c ]](http://csetube.weebly.com/) then

echo "A is big["](http://csetube.weebly.com/) else if [ $b -gt [$c ]](http://csetube.weebly.com/) then

echo "B is big["](http://csetube.weebly.com/) else

echo "C is big["](http://csetube.weebly.com/) fi

fi

## OUTPUT:

ENTER THREE NUMBERS:

23 54 78

C IS BIG.

## RESULT:

Thus the program has been executed successfully.

Ex.No.2d FACTORIAL OF NUMBER

AIM:

To find a factorial of a number using shell script.

ALGORITHM:

Step 1: read a number. Step 2: Initialize fact as 1.

Step 3: Initialize I as 1.

Step 4: While I is lesser than or equal to no.

Step [5: Multiply the value of I and fact and assign to fact increment the valu](http://csetube.weebly.com/)e of I by 1. Step 6: [print the result.](http://csetube.weebly.com/)

Step 7: [Stop the program.](http://csetube.weebly.com/)

PROGRAM:

echo "enter the [number"](http://csetube.weebly.com/) read n

fact=1 i=1

while [ $i -le [$n ]](http://csetube.weebly.com/) do

fact=`expr [$i \* $fact`](http://csetube.weebly.com/) i=`expr $i + 1`

done

echo "the fca[torial number of $n is $fact](http://csetube.weebly.com/)

**OUTPUT:**

Enter the number :

4

The factorial of 4 is 24.

**RESULT:**

Thus the program has been executed successfully.

## FIBONACCI SERIES

Ex.No.2e

## AIM :

To write a program to display the Fibonacci series.

## ALGORITHM:

Step 1: Initialize n1& n2 as 0 & 1. Step 2: enter the limit for Fibonacci. Step 3: initialize variable as 0

Step 4: Print the Fibonacci series n1 and n2. Step 5: While the var number is lesser than lim-2 Step 6: Calculate n3=n1+n2.

Step 7: [Set n1=n2 and n2=n3](http://csetube.weebly.com/)

Step 8: [Increment var by 1 and print n2](http://csetube.weebly.com/) Step 9: [stop the program.](http://csetube.weebly.com/)

## PROGRAM:

echo " ENTER [THE LIMIT FOR FIBONNACI SERIES"](http://csetube.weebly.com/)

read lim n1=0 n2=1

var=0

echo "FIBONA[CCI SERIES IS "](http://csetube.weebly.com/)

echo "$n1" echo "$n2"

while [ $var -[lt `expr $lim - 2` ]](http://csetube.weebly.com/) do

n3=`expr $n1 [+ $n2 `](http://csetube.weebly.com/)

n1=`expr $n2 ` n2=`expr $n3 ` var=`expr $var + 1 ` echo "$n2"

done

## OUTPUT :

enter the limit for Fibonacci:

5

The Fibonacci series is:

0

1

1

2

3

## RESULT:

Thus the program has been executed successfully.

# INTRODUCTION TO OPERATING SYSTEMS

An Operating System is a program that manages the Computer hardware. It controls and coordinates the use of the hardware among the various application programs for the various users.

A Process is a program in execution. As a process executes, it changes state

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

Apart from the program code, it includes the current activity represented by

* + Program Counter,
  + Contents of Processor registers,
  + Process Stack which contains temporary data like function parameters, return addresses and local variables
  + Data section which contains global variables
  + Heap for dynamic memory allocation

A Multi-programmed system can have many processes running simultaneously with the CPU multiplexed among them. By switching the CPU between the processes, the OS can make the computer more productive. There is Process Scheduler which selects the process among many processes that are ready, for program execution on the CPU. Switching the CPU to another process requires performing a state save of the current process and a state restore of new process, this is Context Switch.

## Scheduling Algorithms

CPU Scheduler can select processes from ready queue based on various scheduling algorithms. Different scheduling algorithms have different properties, and the choice of a particular algorithm may favour one class of processes over another. The scheduling criteria include

* + CPU utilization:
  + Throughput: The number of processes that are completed per unit time.
  + Waiting time: The sum of periods spent waiting in ready queue.
  + Turnaround time: The interval between the time of submission of process to the time of completion.
  + Response time: The time from submission of a request until the first response is produced.

The different scheduling algorithms are

## FCFS: First Come First Serve Scheduling

* + It is the simplest algorithm to implement.
  + The process with the minimal arrival time will get the CPU first.
  + The lesser the arrival time, the sooner will the process gets the CPU.
  + It is the non-pre-emptive type of scheduling.
  + The Turnaround time and the waiting time are calculated by using the following formula.

***Turn Around Time = Completion Time - Arrival Time Waiting Time = Turnaround time - Burst Time***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process ID | Arrival Time | Burst Time | Completion Time | Turn  Around Time | Waiting Time |
| 0 | 0 | 2 | 2 | 2 | 0 |
| 1 | 1 | 6 | 8 | 7 | 1 |
| 2 | 2 | 4 | 12 | 8 | 4 |
| 3 | 3 | 9 | 21 | 18 | 9 |
| 4 | 4 | 12 | 33 | 29 | 17 |

Avg Waiting Time=31/5



## SJF: Shortest Job First Scheduling

* + The job with the shortest burst time will get the CPU first.
  + The lesser the burst time, the sooner will the process get the CPU.
  + It is the non-pre-emptive type of scheduling.
  + However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.
  + In the following example, there are five jobs named as P1, P2, P3, P4 and P5. Their arrival time and burst time are given in the table below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Process ID | Arrival Time | Burst Time | Completion Time | Turn  Around Time | Waiting Time |
| 1 | 1 | 7 | 8 | 7 | 0 |
| 2 | 3 | 3 | 13 | 10 | 7 |
| 3 | 6 | 2 | 10 | 4 | 2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 7 | 10 | 31 | 24 | 14 |
| 5 | 9 | 8 | 21 | 12 | 4 |

Since, No Process arrives at time 0 hence; there will be an empty slot in the **Gantt chart** from time 0 to 1 (the time at which the first process arrives)

.

* According to the algorithm, the OS schedules the process which is having the lowest burst time among the available processes in the ready queue.
* Till now, we have only one process in the ready queue hence the scheduler will schedule this to the processor no matter what is its burst time.
* This will be executed till 8 units of time.
* Till then we have three more processes arrived in the ready queue hence the scheduler will choose the process with the lowest burst time.
* Among the processes given in the table, P3 will be executed next since it is having the lowest burst time among all the available processes.



Avg Waiting Time = 27/5

## SRTF: Shortest Remaining Time First Scheduling

* + It is the pre-emptive form of SJF. In this algorithm, the OS schedules the Job according to the remaining time of the execution

## Priority Scheduling

* + In this algorithm, the priority will be assigned to each of the processes.
  + The higher the priority, the sooner will the process get the CPU.
  + If the priority of the two processes is same then they will be scheduled according to their

arrival time.

## Round Robin Scheduling

* + In the Round Robin scheduling algorithm, the OS defines a time quantum (slice).
  + All the processes will get executed in the cyclic way.
  + Each of the process will get the CPU for a small amount of time (called time quantum) and then get back to the ready queue to wait for its next turn. It is a pre-emptive type of sched- uling.

## Multilevel Queue Scheduling

* + A multi-level queue scheduling algorithm partitions the ready queue into several separate queues.
  + The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
  + Each queue has its own scheduling algorithm.

## Multilevel Feedback Queue Scheduling

* + Multilevel feedback queue scheduling, however, allows a process to move between queues.
  + The idea is to separate processes with different CPU-burst characteristics.
  + If a process uses too much CPU time, it will be moved to a lower-priority queue.
  + Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue.
  + This form of aging prevents starvation.

EX.NO.3 **SYSTEM CALLS OF UNIX OPERATING SYSTEM**

## Stat:

**AIM :**

To Execute a Unix Command in a ‘C’ program using stat() system call.

## ALGORITHM:

* 1. Start the program
  2. Declare the variables for the structure stat
  3. Allocate the size for the file by using malloc function
  4. Get the [input of the file whose statistics want to be founded](http://csetube.weebly.com/)
  5. Repeat [the above step until statistics of the files are listed](http://csetube.weebly.com/)
  6. Stop [the program.](http://csetube.weebly.com/)

7.

## PROGRAM:

[#include<stdio.h>](http://csetube.weebly.com/) #includ[e<unistd.h>](http://csetube.weebly.com/) #includ[e<sys/types.h>](http://csetube.weebly.com/) #includ[e<sys/stat.h>](http://csetube.weebly.com/) #includ[e<fcntl.h>](http://csetube.weebly.com/) #includ[e<stdlib.h>](http://csetube.weebly.com/)

int [main(void)](http://csetube.weebly.com/)

{

char \*p[ath,path1[10];](http://csetube.weebly.com/) struct s[tat \*nfile;](http://csetube.weebly.com/)

nfile=([struct stat \*) malloc (sizeof(struct stat));](http://csetube.weebly.com/) printf("enter name of file whose stsistics has to"); scanf("%s",path1);

stat(path1,nfile);

printf("user id %d\n",nfile->st\_uid); printf("block size :%d\n",nfile->st\_blksize); printf("last access time %d\n",nfile->st\_atime);

printf("time of last modification %d\n",nfile->st\_atime); printf("porduction mode %d \n",nfile->st\_mode); printf("size of file %d\n",nfile->st\_size); printf("nu,mber of links:%d\n",nfile->st\_nlink);

}

## OUTPUT:

enter name of file whose stsistics has to stat.c user id 621

block size :4096

last access time 1145148485

time of last modification 1145148485 porduction mode 33204

size of file 654 nu,mber of links:1

## Result:

Thus the program for stat system call has been executed successfully.

## Wait:

**AIM :**

To Execute a Unix Command in a ‘C’ program using wait() system call.

## AlLGORITHM :

* 1. Start the program
  2. Initialize the necessary variables
  3. Use wait() to return the parent id of the child else return -1 for an error
  4. Stop the program.

## PROGRAM:

#include<stdio.h> #include<unistd.h> int [main(void)](http://csetube.weebly.com/)

{

[int pid,status,exitch;](http://csetube.weebly.com/) if((pid=[fork())==-1)](http://csetube.weebly.com/)

{

perror(["error");](http://csetube.weebly.com/) exit (0)[;](http://csetube.weebly.com/)

}

if(pid=[=0)](http://csetube.weebly.com/)

{

sleep([1);](http://csetube.weebly.com/)

printf("[child process");](http://csetube.weebly.com/) exit (0)[;](http://csetube.weebly.com/)

}

else

{

printf("[parent process\n");](http://csetube.weebly.com/)

if((exitch=wait(&status))==-1)

{

perror("during wait()"); exit (0);

}

printf("parent existing\n"); exit (0);

}

}

## OUTPUT :

parent process

child processparent existing

## GETPID:

**AIM:**

To Execute a Unix Command in a ‘C’ program using getpid() system call.

## ALGORITHM:

* 1. Start the program
  2. Declare the necessary variables
  3. The getpid() system call returns the process ID of the parent of the
  4. calling process
  5. Stop the program.

## PROGRAM:

#incl[ude<stdio.h>](http://csetube.weebly.com/) int mai[n()](http://csetube.weebly.com/)

{

int pid; pid=ge[tpid();](http://csetube.weebly.com/)

printf("[process ID is %d\n",pid);](http://csetube.weebly.com/) pid=ge[tppid();](http://csetube.weebly.com/)

printf("[parent process ID id %d\n",pid);](http://csetube.weebly.com/)

}

## OUTPUT:

process [ID is 2848](http://csetube.weebly.com/)

parent [process ID id 2770](http://csetube.weebly.com/)

## RESULT:

Thus the program for getpid system call has been executed successfully.

## Fork:

**AIM:**

To create a process in the following hierarchy

Parent



Child1 Child2Child3

## ALGORITHM:

* 1. De[clare the necessary variables.](http://csetube.weebly.com/)
  2. Par[ent process is the process of the program which is running.](http://csetube.weebly.com/)
  3. Cre[ate the child1 process using fork() When parent is active.](http://csetube.weebly.com/)
  4. Cre[ate the child2 process using fork() when child1 is active.](http://csetube.weebly.com/)
  5. Cre[ate the child3 process using fork() when child2 is active.](http://csetube.weebly.com/)

## PROGRAM:

#incl[ude<stdio.h>](http://csetube.weebly.com/) int mai[n(void)](http://csetube.weebly.com/)

{

int for[k(void),value;](http://csetube.weebly.com/) value=[fork();](http://csetube.weebly.com/)

printf("[main:value =%d\n",value);](http://csetube.weebly.com/) return [0;](http://csetube.weebly.com/)

}

## Output:

main:value =0

main:value =2860

## Result:

Thus the program for fork system call has been executed successfully.

## Exec:

To Execute a Unix Command in a ‘C’ program using exec() system call.

## AIM:

**ALGORITHM:**

* 1. Start the program
  2. Declare the necessary variables
  3. Use the prototype execv (filename,argv) to transform an executable binary file into process
  4. Repeat this until all executed files are displayed
  5. Stop the program.

## PROGRAM:

#include<stdio.h> main()

{

[int pid;](http://csetube.weebly.com/)

ch[ar \*args[]={"/bin/ls","-l",0};](http://csetube.weebly.com/) pr[intf("\nParent Process");](http://csetube.weebly.com/) pi[d=fork();](http://csetube.weebly.com/)

if([pid==0)](http://csetube.weebly.com/)

{

}

## Output:

}

el[se](http://csetube.weebly.com/)

{

}

[execv("/bin/ls",args); printf("\nChild](http://csetube.weebly.com/) [process");](http://csetube.weebly.com/)

[wait();](http://csetube.weebly.com/)

[printf("\nParent process");exit(0);](http://csetube.weebly.com/)

total 440

-rwxrwxr-x 1 skec25 skec25 5210 Apr 16 06:25 a.out

-rw-rw-r-- 1 skec25 skec25 775 Apr 9 08:36 bestfit.c

-rw-rw-r-- 1 skec25 skec25 1669 Apr 10 09:19 correctpipe.c

-rw-rw-r-- 1 skec25 skec25 977 Apr 16 06:15 correctprio.c

-rw------- 1 skec25 skec25 13 Apr 10 08:14 datafile.dat

-rw------- 1 skec25 skec25 13 Apr 10 08:15 example.dat

-rw-rw-r-- 1 skec25 skec25 166 Apr 16 06:25 exec.c

-rw-rw-r-- 1 skec25 skec25 490 Apr 10 09:43 exit.c Parent Process

## Result:

Thus the program for exec system call has been executed successfully.

## Opendir,readdir:

**AIM:**

To write a C program to display the files in the given directory

## ALGORITHM:

* 1. Start the program
  2. Declare the variable to the structure dirent (defines the file system-independent directory) and also for DIR
  3. Specify the directory path to be displayed using the opendir system call
  4. Check for the existence of the directory and read the contents of the directory using readdir system call (returns a pointer to the next active directory entry)
  5. Repeat the above step until all the files in the directory are listed
  6. Stop the program

## PROGRAM:

[#include<stdio.h>](http://csetube.weebly.com/) #includ[e<dirent.h>](http://csetube.weebly.com/) struct d[irent \*dptr;](http://csetube.weebly.com/)

int [main(int argc,char \*argv[])](http://csetube.weebly.com/)

{

c[har buff[256];](http://csetube.weebly.com/) D[IR \*dirp;](http://csetube.weebly.com/)

pr[intf("\n\nEnter directory name");](http://csetube.weebly.com/) sc[anf("%s",buff);](http://csetube.weebly.com/) if([(dirp=opendir(buff))==NULL)](http://csetube.weebly.com/)

{

[printf("Error");](http://csetube.weebly.com/) [exit(1);](http://csetube.weebly.com/)

}

w[hile(dptr=readdir(dirp))](http://csetube.weebly.com/)

{

printf("%s\n",dptr->d\_name);

}

closedir(dirp);

}

## Output:

Enter directory name oslab

openreaddir.c a.out

..vidhya.c vidhya.

## (h)Open:

**AIM:**

To Execute a Unix Command in a ‘C’ program using open() system call.

## ALGORITHM:

**PROGRAM:**

1. Start the program
2. Declare the necessary variables
3. Open file1.dat to read or write access
4. Create file1.dat if it doesn’t exist
5. [Return error if file already exist](http://csetube.weebly.com/)
6. [Permit read or write access to the file](http://csetube.weebly.com/)
7. [Stop the program.](http://csetube.weebly.com/)

[#include<stdio.h>](http://csetube.weebly.com/) #includ[e<sys/types.h>](http://csetube.weebly.com/) #includ[e<sys/stat.h>](http://csetube.weebly.com/) int mai[n()](http://csetube.weebly.com/)

{

[int fd;](http://csetube.weebly.com/) fd[=creat("file1.dat",S\_IREAD|S\_IWRITE);](http://csetube.weebly.com/) if([fd==-1)](http://csetube.weebly.com/)

[printf("Error in opening file1.dat\n");](http://csetube.weebly.com/)

el[se](http://csetube.weebly.com/)

{ }

[\nfile1.dat opened for read/write access\n");](http://csetube.weebly.com/)printf("\nfile1.dat is currently empty");

[printf("](http://csetube.weebly.com/)

}

close(fd);

## OUTPUT:

file1.dat opened for read/write access.

## RESULT:

Thus the program for open system call has been executed successfully.

**Ex.No:4 I/O SYSTEM CALLS OF UNIX OPERATING SYSTEM**

**(OPEN, READ, WRITE, ETC)**

## OPEN,READ,WRITE:

**AIM:**

To implement UNIX I/O system calls open, read , write etc.

## ALGORITHM:

* 1. Create a new file using creat command (Not using FILE pointer).
  2. Open the source file and copy its content to new file using read and write command.
  3. Find size of the new file before and after closing the file using stat command.

## PROGRAM:

[#include<fcntl.h> //file control](http://csetube.weebly.com/) #includ[e<sys/types.h>](http://csetube.weebly.com/) #includ[e<sys/stat.h>](http://csetube.weebly.com/)

static c[har message[]="hai Hello world";](http://csetube.weebly.com/)

int [main()](http://csetube.weebly.com/)

{

[int fd;](http://csetube.weebly.com/)

c[har buffer[80];](http://csetube.weebly.com/) fd[=open("new2file.txt",O\_RDWR|O\_CREAT|O\_EXCL,S\_IREAD|S\_](http://csetube.weebly.com/)IWRITE); if([fd!=-1)](http://csetube.weebly.com/)

{

[printf("new2file.txt opened for read/write access\n");](http://csetube.weebly.com/) [write(fd,message,sizeof(message)); lseek(fd,0l,0);](http://csetube.weebly.com/) [if(read(fd,buffer,sizeof(message))==sizeof(message))](http://csetube.weebly.com/)

[printf("\"%s\" was written to new2file.txt\n",buffer);](http://csetube.weebly.com/)

else

printf("\*\*\*Error readind new2file.txt\*\*\*\n");

}

else

exit(0);

}

## OUTPUT:

close(fd);

printf("\*\*\*new2file.txt already exists\*\*\*\n");

new2file.txt opened for read/write access "hai Hello world" was written to new2file.txt

**Ex.No:5 C PROGRAMS TO SIMULATE UNIX COMMANDS LIKE LS, GREP.**

## LS:

**AIM:**

To implement ls command in c.

## ALGORITHM:

* 1. Include a dirent.h header file.
  2. Create a variable DIR as pointer
  3. Create a structure pointer of dirent
  4. Using opendir function,open the current directory
  5. Read the directory for files using readdir function
  6. Display it till the end of the file.

## PROGRAM:

[#include<stdio.h>](http://csetube.weebly.com/) #includ[e<dirent.h>](http://csetube.weebly.com/) #includ[e<errno.h>](http://csetube.weebly.com/) #includ[e<sys/stat.h>](http://csetube.weebly.com/)

int [main(int argc,char \*\* argv)](http://csetube.weebly.com/)

{

D[IR \*dir;](http://csetube.weebly.com/)

st[ruct dirent \*dirent; char \* where=NULL;](http://csetube.weebly.com/) if([argc==1)where=get\_current\_dir\_name();](http://csetube.weebly.com/) el[se](http://csetube.weebly.com/)

[where=argv[1];](http://csetube.weebly.com/) if([NULL==(dir=opendir(where))){](http://csetube.weebly.com/)

f[printf(stderr,"%d(%s)opendir %s failed\n",errno,strerror(errno),where](http://csetube.weebly.com/)); re[turn 2;](http://csetube.weebly.com/)

}

while(NULL!=(dirent=readdir(dir)))

{

printf("%s\n",dirent->d\_name);

}

closedir(dir); return 0;

}

## OUTPUT:

openreaddir.c file.txt openclose.c staffrr.c fifo.c example.dat newgetpid.c

## RESULT :

Thus the system call program has been executed successfully.

## GREP:

**AIM:**

To [implement the grep command in c](http://csetube.weebly.com/)

## ALGORITHM[:](http://csetube.weebly.com/)

* 1. Obtain [the required pattern to be searched and file name from the user](http://csetube.weebly.com/)
  2. Open [the file and read the constants used by word till the end of the file.](http://csetube.weebly.com/)
  3. Match [the given pattern with the read word and if it matches,display the lin](http://csetube.weebly.com/)e of occurance
  4. Do this [till the end of the file is reached.](http://csetube.weebly.com/)

## PROGRAM:

[#include<stdio.h>](http://csetube.weebly.com/) #includ[e<string.h>](http://csetube.weebly.com/)

int [main(int argv,char \* args[])](http://csetube.weebly.com/)

{

FILE \* f;

char str[100]; char c;

int i,flag,j,m,k;

char arg[]="HI"; char temp[30]; if(argv<3)

{

printf("usage grep<s> <val.txt>\n"); return;

}

f=fopen(args[2],"r"); while(!feof(f))

{

i=0;

while(1)

{

fscanf(f,"%c",&c); if(feof(f))

{

str[i++]='\0'; break;

}

if(c=='\n')

{

str[i++]='\0'; break;

}

str[i++]=c;

}

if([strlen(str)>=strlen(args[1]))](http://csetube.weebly.com/) fo[r(k=0;k<=strlen(str)-strlen(args[1]);k++)](http://csetube.weebly.com/)

{

f[or(m=0;m<strlen(args[1]);m++)](http://csetube.weebly.com/) [temp[m]=str[k+m];](http://csetube.weebly.com/) [temp[m]='\0';](http://csetube.weebly.com/) [if(strcmp(temp,args[1])==0)](http://csetube.weebly.com/)

[{](http://csetube.weebly.com/)

[printf("%s\n",str);](http://csetube.weebly.com/) [break;](http://csetube.weebly.com/)

[}](http://csetube.weebly.com/)

}

}

re[turn 0;](http://csetube.weebly.com/)

}

## OUTPUT:

[skec25@localhost ~]$ ./a.out print stat.c printf("enter name of file whose stsistics has to"); printf("user id %d\n",nfile->st\_uid);

printf("block size :%d\n",nfile->st\_blksize); printf("last access time %d\n",nfile->st\_atime); printf("time of last modification %d\n",nfile->st\_atime); printf("porduction mode %d \n",nfile->st\_mode); printf("size of file

%d\n",nfile->st\_size);

printf("nu,mber of links:%d\n",nfile->st\_nlink);

## RESULT:

Thus the program has been executed successfully.

**Ex.No:6a** FCFS (FIRST COME FIRST SERVE) CPU SCHEDULING

## AIM:

To write a program to implement the FCFS (First Come First Serve) CPU scheduling

Algorithm

## ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time of each processors
4. Calculation of Turn Around Time and Waiting Time
   1. tot\_TAT = tot\_TAT + pre\_TAT
   2. avg\_TAT = tot\_TAT/num\_of\_proc
   3. [tot\_WT = tot\_WT + pre\_WT + PRE\_BT](http://csetube.weebly.com/)
   4. [avg\_WT = tot\_WT/num\_of\_proc](http://csetube.weebly.com/)
5. Displa[y the result](http://csetube.weebly.com/)
6. STOP [the program](http://csetube.weebly.com/)

**PROGRAM:** [(FCFS Scheduling)](http://csetube.weebly.com/) #include<stdio[.h>](http://csetube.weebly.com/) #include<coni[o.h>](http://csetube.weebly.com/)

int p[30],bt[[30],tot\_tat=0,wt[30],n,tot\_wt=0,tat[30],FCFS\_wt=0,FCFS\_tat=0;](http://csetube.weebly.com/) float awt,avg\_[tat,avg\_wt;](http://csetube.weebly.com/)

void main()

{

int i; clrscr();

printf("\nEnt[er the no.of processes \n");](http://csetube.weebly.com/) scanf("%d",&[n);](http://csetube.weebly.com/)

printf("Enter [burst time for each process\n");](http://csetube.weebly.com/)

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

{

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

}

printf("\n FCFS Algorithm \n"); WT\_TAT(&FCFS\_tat,&FCFS\_wt); printf("\n\nTotal Turn around Time:%d",FCFS\_tat);

printf("\nAverage Turn around Time :%d ", FCFS\_tat/n); printf("\nTotal Waiting Time:%d",FCFS\_wt); printf("\nTotal avg. Waiting Time:%d",FCFS\_wt/n); getch();

}

int WT\_TAT(int \*a, int \*b)

{

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

{

if(i==0)

tat[i] = bt[i]; else

tat[i] = tat[i-1] + bt[i]; tot\_tat=tot\_tat+tat[i];

}

\*a = tot\_tat; wt[0]=0;

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

{

wt[i]=wt[i-1]+bt[i-1]; tot\_wt = tot[\_wt+wt[i];](http://csetube.weebly.com/)

}

\*b = tot\_wt;

printf("\nPRO[CESS\t\tBURST TIME\tTURN AROUND TIME\tWAITING TIM](http://csetube.weebly.com/)E");

for(i=0; i<[n; i++)](http://csetube.weebly.com/) printf("\nproc[ess[%d]\t\t%d\t\t%d\t\t%d",p[i],bt[i],tat[i],wt[i]);](http://csetube.weebly.com/) return 0;

}

**OUTPUT**:

(FC[FS Scheduling Algorithm)](http://csetube.weebly.com/)

**RESULT:**

Thus the program to implement the FCFS (First Come First Serve) CPU scheduling Algorithm was written, executed and the output was verified successfully.

**Ex.No:6b** SJF (SHORTEST JOB FIRST) CPU SCHEDULING ALGORITHM

## AIM:

To write a program to implement the SJF (Shortest Job First) CPU scheduling Algorithm

## ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time of each processors
4. Sort the processors based on the burst time
5. Calculation of Turn Around Time and Waiting Time
   1. tot\_TAT = tot\_TAT + pre\_TAT
   2. [avg\_TAT = tot\_TAT/num\_of\_proc](http://csetube.weebly.com/)
   3. [tot\_WT = tot\_WT + pre\_WT + PRE\_BT](http://csetube.weebly.com/)
   4. [avg\_WT = tot\_WT/num\_of\_proc](http://csetube.weebly.com/)
6. Displa[y the result](http://csetube.weebly.com/)
7. STOP [the program](http://csetube.weebly.com/)

**PROGRAM:** [(SJF Scheduling)](http://csetube.weebly.com/) #include<stdio[.h>](http://csetube.weebly.com/) #include<coni[o.h>](http://csetube.weebly.com/)

int p[30],bt[[30],tot\_tat=0,wt[30],n,tot\_wt=0,tat[30],SJF\_wt=0,SJF\_tat=0;](http://csetube.weebly.com/) float awt,avg\_[tat,avg\_wt;](http://csetube.weebly.com/)

void main()

{

int i; clrscr();

printf("\nEnter the no.of processes \n"); scanf("%d",&n);

printf("Enter burst time for each process\n"); for(i=0;i<n;i++)

{

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

}

sort(); WT\_TAT(&SJF\_tat,&SJF\_wt);

printf("\n\nTotal Turn around Time:%d",SJF\_tat); printf("\nAverage Turn around Time :%d ", SJF\_tat/n); printf("\nTotal Waiting Time:%d",SJF\_wt); printf("\nTotal avg. Waiting Time:%d",SJF\_wt/n);

getch();

}

int sort()

{

int t,i,j; for(i=0;i<n;i++)

{

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

{

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

{

swap(&bt[j],&bt[i]);

swap(&p[j],&p[i]);

}

}

}

return 0;

}

int swap(int \*a[, int \*b)](http://csetube.weebly.com/)

{

int t;

t = \*a;

\*a = \*b;

\*b = t; return 0;

}

int WT\_TAT([int \*a, int \*b)](http://csetube.weebly.com/)

{

int i; for(i=0;i<n;i[++)](http://csetube.weebly.com/)

{

if(i==0)

tat[i] = bt[i]; else

tat[i] = tat[i-1] + bt[i]; tot\_tat=tot\_tat+tat[i];

}

\*a = tot\_tat; wt[0]=0;

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

{

wt[i]=wt[i-1]+bt[i-1]; tot\_wt = tot\_wt+wt[i];

}

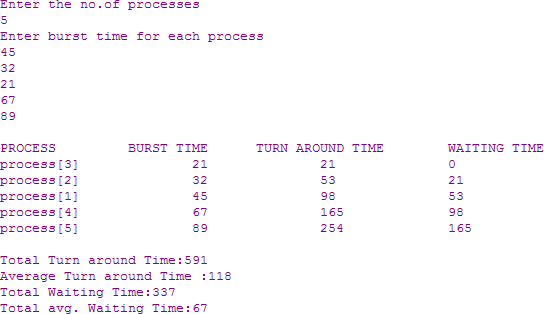
\*b = tot\_wt;

printf("\nPROCESS\t\tBURST TIME\tTURN AROUND TIME\tWAITING TIME");

for(i=0; i<n; i++) printf("\nprocess[%d]\t\t%d\t\t%d\t\t%d",p[i]+1,bt[i],tat[i],wt[i]); return 0;

}

**OUTPUT**: (SJF Scheduling Algorithm)



## RESULT:

Thus [the program to implement the SJF (Shortest Job First) CPU schedulin](http://csetube.weebly.com/)g Algorithm was written, ex[ecuted and the output was verified successfully.](http://csetube.weebly.com/)

**Ex.No:7a** PRIORITY CPU SCHEDULING ALGORITHM

## AIM:

To write a program to implement the Priority CPU scheduling Algorithm

## ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time of each processors
4. Get the priority of all the processors
5. Sort the processors based on the priority
6. Calculation of Turn Around Time and Waiting Time
   1. [tot\_TAT = tot\_TAT + pre\_TAT](http://csetube.weebly.com/)
   2. [avg\_TAT = tot\_TAT/num\_of\_proc](http://csetube.weebly.com/)
   3. [tot\_WT = tot\_WT + pre\_WT + PRE\_BT](http://csetube.weebly.com/)
   4. [avg\_WT = tot\_WT/num\_of\_proc](http://csetube.weebly.com/)
7. Displa[y the result](http://csetube.weebly.com/)
8. STOP [the program](http://csetube.weebly.com/)

**PROGRAM:** [(Priority Scheduling)](http://csetube.weebly.com/)

#include<stdi[o.h>](http://csetube.weebly.com/) #include<coni[o.h>](http://csetube.weebly.com/)

int p[30],bt[[30],tot\_tat=0,pr[30],wt[30],n,tot\_wt=0,tat[30],PR\_wt=0,PR\_tat=0;](http://csetube.weebly.com/) float awt,avg\_[tat,avg\_wt;](http://csetube.weebly.com/)

void main()

{

int i; clrscr();

printf("\nEnter the no.of processes \n"); scanf("%d",&n);

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

{

printf("Enter burst time and priority of process[%d]:",i+1); scanf("%d%d",&bt[i],&pr[i]);

p[i] = i;

}

sort(); WT\_TAT(&PR\_tat,&PR\_wt);

printf("\n\nTotal Turn around Time:%d",PR\_tat); printf("\nAverage Turn around Time :%d ", PR\_tat/n); printf("\nTotal Waiting Time:%d",PR\_wt); printf("\nTotal avg. Waiting Time:%d",PR\_wt/n);

getch();

}

int sort()

{

int t,i,j,t2,t1; for(i=0;i<n;i++)

{

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

{

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

{

swap(&bt[j],&bt[i]);

swap(&p[j],&p[i]);

swap(&pr[j],&pr[i]);

}

}

}

return 0;

}

int swap(int \*a[, int \*b)](http://csetube.weebly.com/)

{

int t;

t = \*a;

\*a = \*b;

\*b = t; return 0;

}

int WT\_TAT([int \*a, int \*b)](http://csetube.weebly.com/)

{

int i; for(i=0;i<n;i[++)](http://csetube.weebly.com/)

{

if(i==0)

tat[i] = bt[i]; else

tat[i] = tat[i-1] + bt[i]; tot\_tat=tot\_tat+tat[i];

}

\*a = tot\_tat; wt[0]=0;

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

{

wt[i]=wt[i-1]+bt[i-1]; tot\_wt = tot\_wt+wt[i];

}

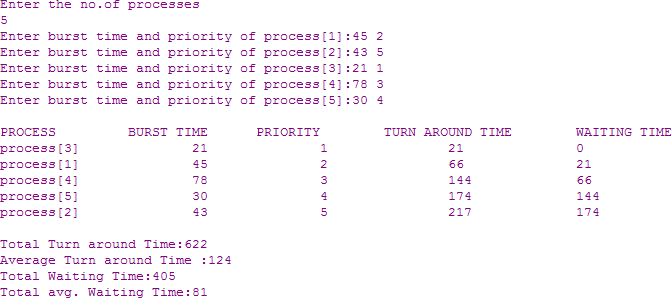
\*b = tot\_wt;

printf("\nPROCESS\t\tBURST TIME\tPRIORITY\tTURN AROUND TIME\tWAITING TIME");

for(i=0; i<n; i++) printf("\nprocess[%d]\t\t%d\t\t%d\t\t%d\t\t%d"

,p[i]+1,bt[i],pr[i],tat[i],wt[i]); return 0; }

**OUTPUT**: (Priority Scheduling Algorithm)



## RESULT:

Thus t[he program to implement the Priority CPU scheduling Algorithm was](http://csetube.weebly.com/) written, executed and t[he output was verified successfully.](http://csetube.weebly.com/)

**Ex.No:7b** ROUND ROBIN CPU SCHEDULING ALGORITHM

## AIM:

To write a program to implement the Round Robin CPU scheduling Algorithm

## ALGORITHM:

1. START the program
2. Get the number of processors
3. Get the Burst time(BT) of each processors
4. Get the Quantum time(QT)
5. Execute each processor until reach the QT or BT
6. Time of reaching processor’s BT is it’s Turn Around Time(TAT)
7. Time w[aits to start the execution, is the waiting time(WT) of each processo](http://csetube.weebly.com/)r
8. Calcula[tion of Turn Around Time and Waiting Time](http://csetube.weebly.com/)
   1. [tot\_TAT = tot\_TAT + cur\_TAT](http://csetube.weebly.com/)
   2. [avg\_TAT = tot\_TAT/num\_of\_proc](http://csetube.weebly.com/)
   3. [tot\_WT = tot\_WT + cur\_WT](http://csetube.weebly.com/)
   4. [avg\_WT = tot\_WT/num\_of\_proc](http://csetube.weebly.com/)
9. Displa[y the result](http://csetube.weebly.com/)
10. STOP [the program](http://csetube.weebly.com/) **PROGRAM:** [(Round Robin Algorithm)](http://csetube.weebly.com/) #include<stdio[.h>](http://csetube.weebly.com/)

#include<coni[o.h>](http://csetube.weebly.com/) int TRUE = 0;

int FALSE = -[1;](http://csetube.weebly.com/)

int tbt[30][,bt[30],tat[30],n=0,wt[30],qt=0,tqt=0,time=0,lmore,t\_tat=0,t\_wt=0;](http://csetube.weebly.com/) void main()

{

int i,j;

clrscr();

printf("\nEnter no. of processors:"); scanf("%d",&n);

printf("\nEnter Quantum Time:"); scanf("%d",&qt); for(i=0;i<n;i++)

{

printf("\nEnter Burst Time of Processor[%d]:",i+1); scanf("%d",&bt[i]);

tbt[i] = bt[i];

wt[i] = tat[i] = 0;

}

lmore = TRUE; while(lmore == TRUE)

{

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

{

if(bt[i] != 0)

wt[i] = wt[i] + (time - tat[i]); tqt = 1;

while(tqt <= qt && bt[i] !=0)

{

lmore = TRUE; bt[i] = bt[i] -1; tqt++;

time++; tat[i] = [time;](http://csetube.weebly.com/)

}

}

}

printf("\nPro[cessor ID\tBurstTime\tTurnAroundTime\tWaitingTime\n");](http://csetube.weebly.com/) for(i=0;i<n;i+[+)](http://csetube.weebly.com/)

{

printf("Proc[essor%d\t\t%d\t\t%d\t\t%d\n",i+1,tbt[i],tat[i],wt[i]);](http://csetube.weebly.com/) t\_tat = t\_tat [+ tat[i];](http://csetube.weebly.com/)

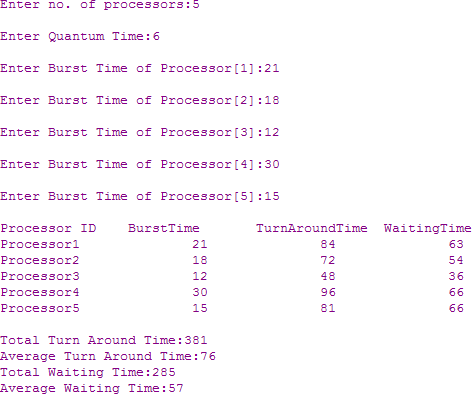
t\_wt = [t\_wt + wt[i];](http://csetube.weebly.com/)

}

printf("[\nTotal Turn Around Time:%d",t\_tat);](http://csetube.weebly.com/) printf("\nAve[rage Turn Around Time:%d",t\_tat/n);](http://csetube.weebly.com/) printf("\nTot[al Waiting Time:%d",t\_wt);](http://csetube.weebly.com/) printf("\nAve[rage Waiting Time:%d",t\_wt/n);](http://csetube.weebly.com/) getch();

}

**OUTPUT**: (Round Robin Scheduling Algorithm)



## RESULT:

Thus [the program to implement the Round Robin CPU scheduling Algorith](http://csetube.weebly.com/)m was written, execu[ted and the output was verified successfully.](http://csetube.weebly.com/)

**Ex.No:8 DEVELOPING APPLICATION USING INTER PROCESS COMMUNICATION (USING SHARED MEMORY, PIPES OR MESSAGE QUEUES)**

1. **Shared memory:**

**AIM:**

To implement the interprocess communication using shared memory.

**ALGORITHM:**

* 1. Start the program
  2. Declare the necessary variables
  3. shmat() and shmdt() are used to attach and detach shared memory segments. They are prototypes as follows:

[void \*shmat(int shmid, const void \*shmaddr, int shmflg);](http://csetube.weebly.com/) [int shmdt(const void \*shmaddr);](http://csetube.weebly.com/)

* 1. [shmat() returns a pointer, shmaddr, to the head of the shared segment associa](http://csetube.weebly.com/)ted with a valid [shmid. shmdt() detaches the shared memory segment located at the address indica](http://csetube.weebly.com/)ted by shmaddr
  2. [Shared1.c simply creates the string and shared memory portion.](http://csetube.weebly.com/)
  3. [Shared2.c attaches itself to the created shared memory portion and us](http://csetube.weebly.com/)es the string [(printf)](http://csetube.weebly.com/)
  4. [Stop the program.](http://csetube.weebly.com/)

**PROGRAM:**

**Shared1.c:**

#inc[lude <sys/types.h>](http://csetube.weebly.com/) #includ[e <sys/ipc.h>](http://csetube.weebly.com/) #includ[e <sys/shm.h>](http://csetube.weebly.com/) #includ[e <stdio.h>](http://csetube.weebly.com/)

#define [SHMSZ 27](http://csetube.weebly.com/) main()

{

char c;

int shmid; key\_t key; char \*shm, \*s; key = 5678;

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

exit(1);

}

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

exit(1);

}

**Shared2.c**

#include <sys/ipc.h> #include <sys/shm.h> #include <stdio.h> #define SHMSZ 27 main()

{

int shmid; key\_t key; char \*shm, \*s; key = 5678;

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

[exit(1);](http://csetube.weebly.com/)

}

if ((s[hm = shmat(shmid, NULL, 0)) == (char \*) -1) {](http://csetube.weebly.com/) pe[rror("shmat");](http://csetube.weebly.com/)

[exit(1);](http://csetube.weebly.com/)

}

for (s [= shm; \*s != NULL; s++)](http://csetube.weebly.com/) pu[tchar(\*s);](http://csetube.weebly.com/)

putch[ar('\n');](http://csetube.weebly.com/)

\*shm [= '\*';](http://csetube.weebly.com/)

exit([0);](http://csetube.weebly.com/)

}

**OUTPUT:**

Abcdef[ghijklmnopqrstuvwxyz](http://csetube.weebly.com/)

**RESULT:**

Thus t[he program has been executed successfully and the output is verified.](http://csetube.weebly.com/)

1. **PIPES:**

**AIM:**

To implement the interprocess communication using pipes.

**ALGORITHM:**

* 1. Start the program
  2. Create the child process using fork()
  3. Create the pipe structure using pipe()
  4. Now close the read end of the parent process using close()
  5. Write the data in the pip[e using write()
  6. Now close the write end of child process using close()
  7. Read the data in the pipe using read()
  8. Display the string
  9. Stop the program.

**PROGRAM:**

#include<stdio.h> int main()

{

int fd[2],child; char a[10];

printf("\n enter the string to enter into the pipe:"); scanf("%s",a);

pipe(fd); child=fork(); if(!child)

{ close(fd[10]);

write(fd[1],a,5); wait(0)[;](http://csetube.weebly.com/)

}

else

{

close([fd[1]);](http://csetube.weebly.com/)

read(fd[[0],a,5);](http://csetube.weebly.com/)

printf("[\n the string retrived from pipe is %s\n",a);](http://csetube.weebly.com/)

}

return [0;](http://csetube.weebly.com/)

}

**OUTPUT:**

enter t[he string to enter into the pipe:computer](http://csetube.weebly.com/) the str[ing retrived from pipe is computer](http://csetube.weebly.com/)

**RESULT:**

Thus th[e program has been executed and the output is verified successfully.](http://csetube.weebly.com/)

## message queue:

**AIM:**

To implement the interprocess communication using message passing.

## ALGORITHM:

* 1. Start the program
  2. Create two files msgsend and msgrecv.c
  3. Msgsend creates a message queue with a basic key and message flag msgflg = IPC\_CREAT | 0666 -- create queue and make it read and appendable by all, and sends one message to the queue.
  4. Msgrecv reads the message from the queue
  5. A message of type (sbuf.mtype) 1 is sent to the queue with the message ``Did you get this?''
  6. The Message queue is opened with msgget (message flag 0666) and the ***same*** key as message\_send.c
  7. A message of the ***same*** type 1 is received from the queue with the message ``Did you get this?'' stored in rbuf.mtext.
  8. Stop the program.

## PROGRAM:

**Msgsend:**

#include <sys/types.h> #include <sys/ipc.h> #include <sys/msg.h> [#include <stdio.h>](http://csetube.weebly.com/) #includ[e <string.h>](http://csetube.weebly.com/)

#define [MSGSZ 128](http://csetube.weebly.com/) typede[f struct msgbuf {](http://csetube.weebly.com/)

[long mtype;](http://csetube.weebly.com/)

c[har mtext[MSGSZ];](http://csetube.weebly.com/)

} [message\_buf;](http://csetube.weebly.com/)

main()

{

int [msqid;](http://csetube.weebly.com/)

int [msgflg = IPC\_CREAT | 0666;](http://csetube.weebly.com/) key\_[t key;](http://csetube.weebly.com/)

mess[age\_buf sbuf;](http://csetube.weebly.com/) size\_[t buf\_length;](http://csetube.weebly.com/)

key = [1234;](http://csetube.weebly.com/)

(void) fprintf(stderr, "\nmsgget: Calling msgget(%#lx,\

%#o)\n", key, msgflg);

if ((msqid = msgget(key, msgflg )) < 0) { perror("msgget");

exit(1);

}

else

(void) fprintf(stderr,"msgget: msgget succeeded: msqid = %d\n", msqid); sbuf.mtype = 1;

(void) fprintf(stderr,"msgget: msgget succeeded: msqid = %d\n", msqid); (void) strcpy(sbuf.mtext, "Did you get this?");

(void) fprintf(stderr,"msgget: msgget succeeded: msqid = %d\n", msqid);

buf\_length = strlen(sbuf.mtext) + 1 ;

if (msgsnd(msqid, &sbuf, buf\_length, IPC\_NOWAIT) < 0) {

printf ("%d, %d, %s, %d\n", msqid, sbuf.mtype, sbuf.mtext, buf\_length); perror("msgsnd");

exit(1);

}

else

printf("Message: \"%s\" Sent\n", sbuf.mtext);

exit([0);](http://csetube.weebly.com/)

}

## Msgrecv:

[#include <sys/types.h>](http://csetube.weebly.com/) #includ[e <sys/ipc.h>](http://csetube.weebly.com/) #includ[e <sys/msg.h>](http://csetube.weebly.com/) #includ[e <stdio.h>](http://csetube.weebly.com/)

#define [MSGSZ 128](http://csetube.weebly.com/) typede[f struct msgbuf {](http://csetube.weebly.com/)

long [mtype;](http://csetube.weebly.com/)

char [mtext[MSGSZ];](http://csetube.weebly.com/)

} mess[age\_buf;](http://csetube.weebly.com/)

main()

{

int msqid; key\_t key; message\_buf rbuf;

key = 1234;

if ((msqid = msgget(key, 0666)) < 0) { perror("msgget");

exit(1);

}

if (msgrcv(msqid, &rbuf, MSGSZ, 1, 0) < 0) {

perror("msgrcv"); exit(1);

}

printf("%s\n", rbuf.mtext); eixit(0);

}

## OUTPUT:

“Did you get this?”

## RESULT:

Thus the program has been executed successfully and the output is verified.

**Ex.No:9** INTERPROCESS COMMUNICATION USING MESSAGE PASSING.

## AIM:

To write a program to solve the Producer Consumer Problem using Semaphores

## ALGORITHM:

1. START the program
2. If the request is for Producer
   1. Get the count of number of items to be produced
   2. [Add the count with current Buffer size](http://csetube.weebly.com/)
   3. [If the new Buffer size is full](http://csetube.weebly.com/)

[Don’t produce anything; display an error message](http://csetube.weebly.com/)

[Producer has to wait till any consumer consume the existing ite](http://csetube.weebly.com/)m

* 1. [Else](http://csetube.weebly.com/)

[Produce the item](http://csetube.weebly.com/)

[Increment the Buffer by the number of item produced](http://csetube.weebly.com/)

1. If the r[equest is for Consumer](http://csetube.weebly.com/)
   1. [Get the count of number of items to be consumed](http://csetube.weebly.com/)
   2. [Subtract the count with current Buffer size](http://csetube.weebly.com/)
   3. [If the new Buffer size is lesser than 0](http://csetube.weebly.com/)

[Don’t allow the consumer to consume anything; display an error](http://csetube.weebly.com/) message [Consumer has to wait till the producer produce new items](http://csetube.weebly.com/)

* 1. [Else](http://csetube.weebly.com/)

[Consume the item](http://csetube.weebly.com/)

[Decrement the Buffer by the number of item consumed](http://csetube.weebly.com/)

1. STOP [the program](http://csetube.weebly.com/)

## PROGRAM:

#include<stdio.h>

int n=0,buffersize=0,currentsize=0; void producer()

{

printf("\nEnter number of elements to be produced: "); scanf("%d",&n);

if(0<=(buffersize-(currentsize+n)))

{

currentsize+=n;

printf("%d Elements produced by producer where buffersize is %d\n", currentsize, buffersize);

}

else

printf("\nBuffer is not sufficient\n");

}

void consumer()

{

int x;

printf("\nEnter no. of elements to be consumed: "); scanf("%d",&x);

if(currentsize>=x)

{

currentsize-=x;

printf("\nNumber of elements consumed: %d, Number of Elements left: %d", x, currentsize);

}

else

{

printf("\nN[umber of Elements consumed should not be greater than Number of](http://csetube.weebly.com/) Elements produced\n");

}

}

void main()

{

int c;

printf("\nEnt[er maximum size of buffer:");](http://csetube.weebly.com/) scanf("%d",[&buffersize);](http://csetube.weebly.com/)

do

{

printf("\n1[.Producer 2.Consumer 3.Exit");](http://csetube.weebly.com/) printf("\nE[nter Choice:");](http://csetube.weebly.com/) scanf("%d[",&c);](http://csetube.weebly.com/)

switch(c)

{

case 1:

if(currentsize >= buffersize) printf("\nBuffer is full. Cannot produce");

else

producer(); break; case 2:

if(currentsize <= 0)

printf("\nBuffer is Empty. Cannot consume"); else

consumer(); break; default:

exit(); break;

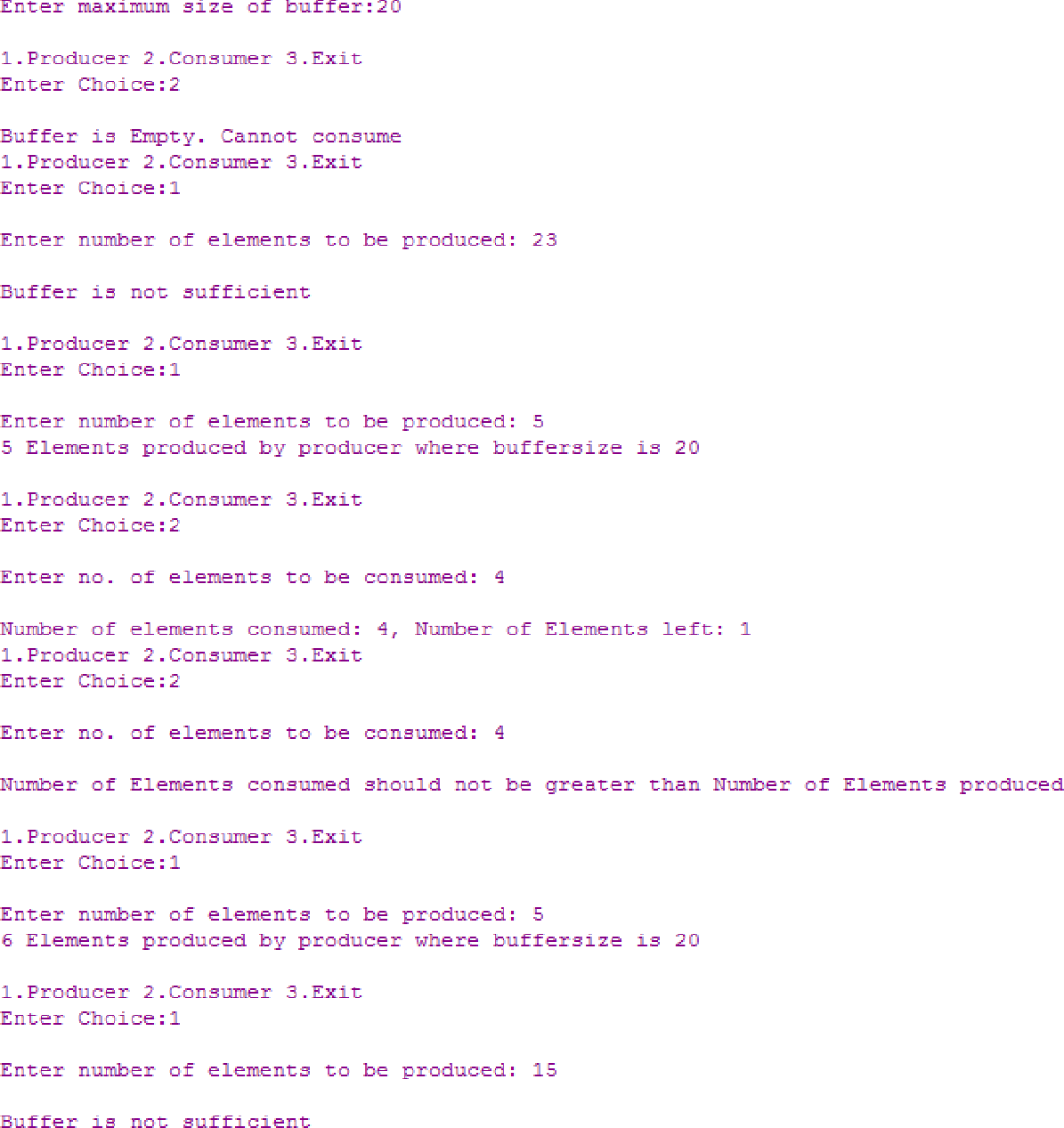
}

}

while(c!=3);

}

**OUTPUT:** (Producer Consumer problem)



**RESULT:**

Thus the program has been executed successfully and the output is verified.

## Ex.No:9 MEMORY MANAGEMENT SCHEME-I

1. **Firstfit:**

**AIM:**

To write a program to implement first fit algorithm for memory management.

## ALGORITHM:

1. Start the pro[cess](http://csetube.weebly.com/)
2. Declare the [size](http://csetube.weebly.com/)
3. Get the [number of processes to be inserted](http://csetube.weebly.com/)
4. Allocate the [first hole that is big enough searching](http://csetube.weebly.com/)
5. Start at the [beginning of the set of holes](http://csetube.weebly.com/)
6. If not start a[t the hole that is sharing the pervious first fit search end](http://csetube.weebly.com/)
7. Compare [the hole](http://csetube.weebly.com/)
8. if large e[nough then stop searching in the procedure](http://csetube.weebly.com/)
9. Display the [values](http://csetube.weebly.com/)
10. Stop the pr[ocess](http://csetube.weebly.com/)

## PROGRAM:

#incl[ude<stdio.h>](http://csetube.weebly.com/) struct p[rocess](http://csetube.weebly.com/)

{

int ps; int flag;

} p[50];

struct sizes

{

int size; int alloc;

} s[5];

int main()

{

int i=0,np=0,n=0,j=0; printf("\n first fit"); printf("\n");

printf("enter the number of blocks \t"); scanf("%d",&n);

printf("\t\t\n enter the size for %d blocks\n",n); for(i=0;i<n;i++)

{

printf("enter the size for %d block \t",i); scanf("%d",&s[i].size);

}

printf("\n\t\t enter the number of process\t",i); scanf("%d",&np);

printf("enter the size of %d processors !\t",np); printf("/n");

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

{

printf("enter the size of process %d\t",i); scanf("[\n%d",&p[i].ps);](http://csetube.weebly.com/)

}

printf("[\n\t\t Allocation of blocks using first fit is as follows\n");](http://csetube.weebly.com/) printf("[\n\t\t process \t process size\t blocks\n");](http://csetube.weebly.com/) for(i=0[;i<np;i++)](http://csetube.weebly.com/)

{

for(j=[0;j<n;j++)](http://csetube.weebly.com/)

{

if(p[i][.flag!=1)](http://csetube.weebly.com/)

{

if(p[i][.flag!=1)](http://csetube.weebly.com/)

{

if(p[i][.ps<=s[j].size)](http://csetube.weebly.com/)

{

if(!s[j].[alloc)](http://csetube.weebly.com/)

{

p[i].fla[g=1;](http://csetube.weebly.com/)

s[j].alloc=1;

printf("\n\t\t %d\t\t\t%d\t%d\t",i,p[i].ps,s[j].size);

}

}

}

}

}

}

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

{

if(p[i].flag!=1)

printf("sorry !!!!!!!process %d must wait as there is no sufficient memory");

}

}

\

## OUTPUT:

first fit

enter the number of blocks 4

|  |  |
| --- | --- |
| enter the size for 4 blocks |  |
| enter the size for 0 block | 3 |
| enter the size for 1 block | 2 |
| enter the size for 2 block | 5 |
| enter the size for 3 block | 10 |

enter the number of process 4 enter the size [of 4 processors !](http://csetube.weebly.com/)

|  |  |
| --- | --- |
| enter the size [of process 0](http://csetube.weebly.com/) | [2](http://csetube.weebly.com/) |
| enter the size [of process 1](http://csetube.weebly.com/) | [6](http://csetube.weebly.com/) |
| enter the size [of process 2](http://csetube.weebly.com/) | [7](http://csetube.weebly.com/) |
| enter the size [of process 3](http://csetube.weebly.com/) | [9](http://csetube.weebly.com/) |

Alloca[tion of blocks using first fit is as follows](http://csetube.weebly.com/) proc[ess process size blocks](http://csetube.weebly.com/)

[wait as there is no sufficient memory](http://csetube.weebly.com/)

|  |  |  |
| --- | --- | --- |
| so0rry [!!!!!!](http://csetube.weebly.com/) | [!proce2ss 1](http://csetube.weebly.com/) | [m3ust](http://csetube.weebly.com/) |
| 1 | [6](http://csetube.weebly.com/) | [10](http://csetube.weebly.com/) |

## RESULT:

The program for first fit was implemented and hence verified

## BESTFIT:

**AIM:**

To write a program to implement best fit algorithm for memory management.

## ALGORITHM:

1. Start the process
2. Declare the size
3. Get the number of processes to be inserted
4. Allocate the best hole that is small enough searching
5. Start at the best of the set of holes
6. If not start at the hole that is sharing the pervious best fit search end
7. Compare the hole
8. If small enough then stop searching in the procedure
9. Display the values
10. Stop the process

## PROGRAM:

#include<stdio.h> #define MAX 20 int main()

{

int bsize[MAX],fsize[MAX],nb,nf; int temp,low=10000;

[static int bflag[MAX],fflag[MAX];](http://csetube.weebly.com/) int i,j;

printf("[\n enter the number of blocks");](http://csetube.weebly.com/) scanf("[%d",&nb);](http://csetube.weebly.com/)

for(i=[1;i<=nb;i++)](http://csetube.weebly.com/)

{

printf("[Enter the size of memory block % d",i);](http://csetube.weebly.com/) scanf("[%d", &bsize[i]);](http://csetube.weebly.com/)

}

printf("[\n enter the number of files");](http://csetube.weebly.com/) scanf("[%d",&nf);](http://csetube.weebly.com/)

for(i=[1;i<=nf;i++)](http://csetube.weebly.com/)

{

printf("[\n enetr the size of file %d",i);](http://csetube.weebly.com/) scanf("[%d",&fsize[i]);](http://csetube.weebly.com/)

}

for(i=[1;i<=nf;i++)](http://csetube.weebly.com/)

{

for(j=1;j<=nb;j++)

{

if(bflag[j]!=1)

{

temp=bsize[j]-fsize[i]; if(temp>=0)

{

if(low>temp)

{

fflag[i]=j; low=temp;

}

}

}}

bflag[fflag[i]]=1; low=10000;

}

printf("\n file no \t file.size\t block no \t block size"); for(i=1;i<=nf;i++)

printf("\n \n %d \t\t%d\t\t%d\t\t%d",i,fsize[i],fflag[i],bsize[fflag[i]]);

}

## OUTPUT:

enter the number of blocks5

Enter the size of memory block 1 10 Enter the size of memory block 2 12 Enter the size of memory block 3 10 Enter t[he size of memory block 4 5](http://csetube.weebly.com/) Enter t[he size of memory block 5 7](http://csetube.weebly.com/)

enter [the number of files 5](http://csetube.weebly.com/) enetr t[he size of file 1 5](http://csetube.weebly.com/) enetr t[he size of file 2 6](http://csetube.weebly.com/)

file no [file.size block no block size](http://csetube.weebly.com/)

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | [5](http://csetube.weebly.com/) | [4 5](http://csetube.weebly.com/) | |
| 2 | [6](http://csetube.weebly.com/) | [5](http://csetube.weebly.com/) | [7](http://csetube.weebly.com/) |
| 3 | [7](http://csetube.weebly.com/) | [1](http://csetube.weebly.com/) | [10](http://csetube.weebly.com/) |
| 4 | 10 | 3 | 10 |
| 5 | 12 | 2 | 12 |

## RESULT:

The program for best fit was implemented and hence verified

## Ex.No 11 MEMORY MANAGEMENT SCHEME II

1. **FIFO:**

**AIM:**

To write a program to implement FIFO page replacement algorithm

## ALGORITHM:

1. Start the process
2. Declare the [size with respect to page length](http://csetube.weebly.com/)
3. Check the [need of replacement from the page to memory](http://csetube.weebly.com/)
4. Check the [need of replacement from old page to new page in memory](http://csetube.weebly.com/)
5. Forma que[ue to hold all pages](http://csetube.weebly.com/)
6. Insert the pa[ge require memory into the queue](http://csetube.weebly.com/)
7. Check for b[ad replacement and page fault](http://csetube.weebly.com/)
8. Get the [number of processes to be inserted](http://csetube.weebly.com/)
9. Display the [values](http://csetube.weebly.com/)
10. Stop the pr[ocess](http://csetube.weebly.com/)

## PROGRAM:

#incl[ude<stdio.h>](http://csetube.weebly.com/) int mai[n()](http://csetube.weebly.com/)

{

[int i,j,n,a[50],frame[10],no,k,avail,count=0;](http://csetube.weebly.com/) printf("[\n enter the number of pages:\n");](http://csetube.weebly.com/) scanf("[%d",&n);](http://csetube.weebly.com/)

printf("\n enter the page number:\n"); for(i=1;i<=n;i++) scanf("%d",&a[i]);

printf("\n enter the number of frames:\n"); scanf("%d",&no);

for(i=0;i<no;i++) frame[i]=-1;

j=0;

printf("\tref string\t page frmaes\n"); for(i=1;i<=n;i++)

{

printf("%d\t\t",a[i]); avail=0; for(k=0;k<no;k++)

if(frame[k]==a[i]) avail=1; if(avail==0)

{ frame[j]=a[i]; j=(j+1)%no; count++; for(k=0;k<no;k++)

printf("%d\t",frame[k]);

}

printf("\n");

}

printf("page fault is %d",count); getch();

return 0;

}

## OUTPUT:

enter [the number of pages:](http://csetube.weebly.com/) 4

enter t[he reference string:](http://csetube.weebly.com/) 7

2

1

0

enter [the number of frames:](http://csetube.weebly.com/) 3

re[f string page frmaes](http://csetube.weebly.com/) 7 [7 -1 -1](http://csetube.weebly.com/)

2 [7 2 -1](http://csetube.weebly.com/)

1 [7 2 1](http://csetube.weebly.com/)

0 [0 2 1](http://csetube.weebly.com/)

page fault is 4

## RESULT:

The program for FIFO page replacement was implemented and hence verified.

## LRU:

**AIM:**

To write a program a 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:

#include<stdlib.h> #include<stdio.h> #define [max 100](http://csetube.weebly.com/)

#define [min 10](http://csetube.weebly.com/)

int ref[m[ax],count,frame[min],n;](http://csetube.weebly.com/) void inp[ut()](http://csetube.weebly.com/)

{

int i,tem[p;](http://csetube.weebly.com/) count=0[;](http://csetube.weebly.com/)

printf("[\n\n\tEnter the number of page frames : ");](http://csetube.weebly.com/) scanf("[%d",&n);](http://csetube.weebly.com/)

printf("[\n\n\tEnter the reference string (-1 for end) : ");](http://csetube.weebly.com/) scanf("[%d",&temp);](http://csetube.weebly.com/)

while(te[mp != -1)](http://csetube.weebly.com/)

{

ref[cou[nt++]=temp;](http://csetube.weebly.com/) scanf("[%d",&temp);](http://csetube.weebly.com/)

}

}

void LRU()

{

int i,j,k,stack[min],top=0,fault=0; system("CLS"); for(i=0;i<count;i++)

{

if(top<n) stack[top++]=ref[i],fault++; else

{

for(j=0;j<n;j++) if(stack[j]==ref[i]) break;

if(j<n)

{

for(k=j;k<n-1;k++)

stack[k]=stack[k+1]; stack[k]=ref[i];

}

else

{

for(k=0;k<n-1;k++) stack[k]=stack[k+1]; stack[k]=ref[i]; fault++;

}

}

printf("\n\nAfter inserting %d the stack status is : ",ref[i]); for(j=0;j<top;j++)

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

}

printf("\n\n\tEnd to inserting the reference string."); printf("\n\n\[tTotal page fault is %d.",fault);](http://csetube.weebly.com/) printf("\n\n\[tPress any key to continue.");](http://csetube.weebly.com/)

}

void main()

{

int x;

//freopen("in[.cpp","r",stdin);](http://csetube.weebly.com/) while(1)

{

printf("\n\n\[t-----MENU ");](http://csetube.weebly.com/)

printf("\n\t1. [Input ");](http://csetube.weebly.com/)

printf("\n\t2. [LRU (Least Recently Used) Algorithm");](http://csetube.weebly.com/) printf("\n\t0. [Exit.");](http://csetube.weebly.com/)

printf("\n\n\[tEnter your choice.");](http://csetube.weebly.com/) scanf("%d",&[x);](http://csetube.weebly.com/)

switch(x)

{

case 1: input();

break; case 2:

LRU();

break; case 0:

exit(0);

}

}

}

## OUTPUT:

-MENU-

* 1. Input
  2. LRU (Least Recently Used) Algorithm

1. Exit.

Enter your choice.1

Enter the number of page frames : 3

Enter the reference string (-1 for end) : 2

0

1

1

-1

[MENU](http://csetube.weebly.com/)

1. [Input](http://csetube.weebly.com/)
2. [LRU (Least Recently Used) Algorithm](http://csetube.weebly.com/)
3. [Exit.](http://csetube.weebly.com/)

E[nter your choice.2](http://csetube.weebly.com/)

After i[nserting 2 the stack status is : 2](http://csetube.weebly.com/) After i[nserting 0 the stack status is : 2 0](http://csetube.weebly.com/) After i[nserting 1 the stack status is : 2 0 1](http://csetube.weebly.com/)

After i[nserting 1 the stack status is : 2 0 1](http://csetube.weebly.com/) E[nd to inserting the reference string.](http://csetube.weebly.com/) Total page fault is 3.

Press any key to continue.

## RESULT:

MENU

1. Input
2. LRU (Least Recently Used) Algorithm
3. Exit.

Enter your choice.0

The program for LRU page replacement was implanted and hence verified.

## Ex.No:12 FILE ALLOCATION TECHNIQUE-CONTIGUOUS AIM:

**To Write a C Program to implement file allocation technique.**

**ALGORITHM:**

1. Start the process
2. Declare the necessary variables
3. Get the number of files
4. Get the total no. of blocks that fit in to the file
5. Display the file name, start address and size of the file. 6.Stop the pro[gram.](http://csetube.weebly.com/)

## PROGRAM:

#include<stdi[o.h>](http://csetube.weebly.com/) void main()

{

int i,j,n,block[[20],start:](http://csetube.weebly.com/)

printf(“Enter [the no. of file:\n”);](http://csetube.weebly.com/) scanf(“%d”,&[n);](http://csetube.weebly.com/)

printf(“Enter [the number of blocks needed for each file:\n”);](http://csetube.weebly.com/) for(i=0,i<n;i++[)](http://csetube.weebly.com/)

scanf(“%d”[,&block[i]);](http://csetube.weebly.com/) start=0;

printf(“\t\tFile [name\tStart\tSize of file\t\t\n”);](http://csetube.weebly.com/) printf(“\n\t\tFi[le1\t\t%d\t\t\t%d\n”,start,block[0]);](http://csetube.weebly.com/) for(i=2;i<=n;i+[+)](http://csetube.weebly.com/)

{

Start=start+block[i-2]; Printf(“\t\tFile%d\t\t%d\t\tD\n”,i,start,block[i-1]);

}

}

Output

Enter the number of file:4

Enter the number of blocks needed for each file: 3

5

6

|  |  |  |
| --- | --- | --- |
| 1  Filename | start | size of file |
| File 1 | 0 | 3 |
| File 2 | 3 | 5 |
| File 3 | 8 | 6 |
| File 4 | 14 | 1 |

Result:

Thus the program for file allocation technique has been done successfully.

## EX.NO: 13 IMPLEMENTATION BANKERS ALGORITHM FOR DEAD LOCK AVOIDANCE

**AIM**

To write a C program to implement bankers algorithm for dead lock avoidance

## ALGORITHM:

Step 1: Start the Program

Step 2:Obtain the required data through char and int datatypes. Step 3:Enter the filename,index block.

Step 4: Print the file name index loop.

Step 5:File is allocated to the unused index blocks

Step 6: This is allocated to the unused linked allocation. Step 7: Stop the execution

## PROGRAM

#include<stdio.h> #include<conio.h> void main()

{

int work[5],avl[5],alloc[10][10],l;

int need[10][10],n,m,I,j,avail[10],max[10][10],k,count,fcount=0,pr[10];

char finish[10]={„f‟,‟f‟,„f‟,‟f‟,„f‟,‟f‟,„f‟,‟f‟,„f‟,‟f‟};

clrscr();

printf(“enter the no of process”); scanf(“%d”,&n);

printf(“enter the no of resources”); scanf(“%d”,&m);

printf(“enter the total no of resources”); for(i=1;i<=m;i++) scanf(“%d”,&avail[i]);

printf(“enter the max resources req by each pr alloc matrix”);

for(i=1;i<=n;i++) for(j=1;j<=m;j++) scanf(“%d”,&max[i][j]); printf(“process allocation matrix”); for(i=1;i<=n;i++) for(j=1;j<=m;j++) scanf(“%d”,&alloc[i][j]); for(i=1;i<=n;i++) for(j=1;j<=m;j++) need[i][j]=max[i][j]-alloc[i][j]; for(i=1;i<=n;i++)

{ k=0;

for(j=1;j<=m;j++)

{

k=k+alloc[i][j];

}

avl[i]=avl[i]-k;

work[i]=avl[i];

}

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

{

count=0; for(j=1;j<=m;j++)

{

if((finish[i]==‟f‟)&&(need[i][j]<=work[i])) of

count++;

}

if(count==m)

{

for(l=1;l<=m;l++) work[l]=work[l]+alloc[i][l]; finish[i]=‟t‟;

pr[k]=i;

JBIET

break;

}

}

for(i=1;i<=n;i++) if(finish[i]==‟t‟) fcount++; if(fcount==n)

{

printf(“the system is in safe state”);

Dept for(i=1;i<=n;i++) printf(“%d”,pr[i]);

}

else

printf(“the system is not in safe state”);

getch();

}

## RESULT:

Thus the program was executed successfully.