Exp No: 1

# LINUX COMMANDS

1. pwd :-

Linux pwd (print working directory) command displays your location currently you are working on. It will give the whole path starting from the root ending to the directory.

2. cd Command :-

The 'cd' stands for 'change directory' and this command is used to change the current directory ie; the directory in which the user is currently working.

Syntax :- cd <dirname>

3. Linux ls command :-

The 'ls' is the list command in Linux. It will show the full list on content of your directory Just type ls and press the enter key. The whole content will be shown.

Example : ls

4. Linux mkdir | Linux Create Directory :-

Now lets learn how to create your own

directory with the help of command prompt.

The mkdir stands for 'make directory'. With the help of mkdir command, you can create a new directory whenever you want in our system. Just type "mkdir" <dir name>, in place of <dirname> type the name of new directory, you want to create and then press enter.

Syntax: mkdir <dirname>

5 Linux rmdir Command :-

This command is used to delete a directory. But it will not be able to delete a directory including a sub-directory. It means, a directory has to be empty to be deleted.

Syntax: rmdir <dir name>

6. Linux rm | Linux Delete File

The 'rm' means remove. This command is used to remove a file. The command line doesn't have a recycle bin or trash unlike other GUI's to recover the files. Hence be very much careful while using this command. Once you have deleted a file, it is removed permanently.

Syntax: rm <filename>

7. Linux cp | Linux Copy File

'cp' means copy. 'cp' command is used to copy a file or a directory

To copy a file into the same directory syntax will be,

cp<existing filename><new file name>

8. Linux mv | Linux Move File

Linux mv command is used to move existing file or diretory from one location to another. It is also used to rename a file or directory. If you want to rename a single directory or file, then 'mv' option will be better to use.

9. Linux Rename File and Directory

To rename a file, there are other commands also like 'mv'. But rename command is slightly different than others. This command will be rarely used and, it works differently on different distros of linux.

Basic syntax: rename 's/old-name/new-name/' files.

10. Linux Man Command

The "man' is the shortform for manual page. In unix-like operating systems such as linux

man is an interface to view the system's reference manual.
A user can request to display a man page by simply typing man followed by a space and then arguement. Here, its arguement can be a command utility or function. A manual page associated with each of these arguements is displayed.

Syntax of man:

man [option(s)] keyword(s)

For example,

man ls

This command will display all the information about 'ls' command as shown in the screen shot.

11. Fork :-

The fork() function is used to create a new process by duplicating existing process from which it is called. The existing process from which this function is called becomes the parent process and the newly created process becomes the child process.

12. exec :-

exec command in linux is used to execute a command from the bash itself. This command does not create a new process, it just replaces the

bash with the command to be executed.

Syntax:

exec [-cl] [-a name] [command [arguments]] [redirection...]

13. getpid :-

It returns the ~~calling~~ process ID of the calling process. (This is often used by routines that generate unique temporary filenames)

14. exit :-

1. exit: Exit without parameter. After pressing enter, the terminal will simply close.
2. exit [n]: Exit with Parameter
3. exit n : Using "sudo su" we are going to the root directory and then exit the root directory with a return status of 5.
4. exit-help: It displays help information.

15 wait :

wait is a built-in command of Linux that waits for completing any running process. wait command is used with a particular process ID or job id. When multiple processes are running in the shell, then only the process id of the last command will be known by the current shell.

16. close :

close() closes a file descriptor, so that it no longer refers to any file ~~or~~ and may be reused.

17. stat :-

The stat command prints information about given files and file systems.

18. opendir :-

The opendir() function opens a directory stream corresponding to the directory name and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

19. readdir :-

The readdir() function shall return a pointer to a structure representing the directory entry at the current position of the directory stream specified by the argument dirp and position the directory stream at the next entry. It shall return a null pointer upon reaching the end of the directory stream.

END OF EXP 1

EXP 2 WRITTEN

AS LEFT → RIGHT

OUTPUT

1) Question - 1

Hello World

2) Question - 2

Enter the number
5
The number is 5

3) Question - 3

Enter two numbers
5
10
Result =15

Exp No: 2

# SHELL PROGRAMS

AIM :-

To familiarize with :-

1) Write a shell program to display a message on the screen

→ echo " Hello World "

2) Write a shell program to accept a number and print it on the screen.

→ echo " Enter the number "
read a
echo " The number is $a "

3) Write a shell program to add two numbers

→ echo " Enter two numbers
read a
read b
x = 'expr $a + $b'
echo " Result = $x "

4) Question - 4

Enter two numbers
5
6
After swapping a = 5, b = 6

5) Question - 5

Enter the number
6
The square of 6 is 36

6) Question - 6

Enter the number
21
The number is odd

4) Write a shell program to swap 2 numbers without temporary variables

→ echo "Enter two numbers"
a = 'expr $a + $b'
b = 'expr $a - $b'
a = 'expr $a + $b'
echo "After swapping, a = $a, b = $b"

5) Write a program to find square of a number

→ echo "Enter the number"
read a
b = 'expr $a * $a'
echo "The square of $a is $b"

6) Write a program to check whether a number is even or odd.

→ echo "Enter a number"
read a
r = 'expr $a % 2'
if [ $r -eq 0 ]
then
echo "The number is even"
else
echo "The number is odd"
fi

7) Question-7

Enter two numbers

5

3

5 is greater than 3

8) Question - 8

Enter three numbers

8

5

1

8 is greater than 5 and 1.

7) Write a shell program to find the greatest of 2 numbers.

→ echo "Enter 2 numbers"
read a
read b
if [ $a -gt $b]
then
echo "$a is greater than $b"
else
echo "$b is greater them $a"
fi

8) Write a shell program to find the greatest of 3 numbers

→ echo "Enter three numbers"
read a
read b
read c
if [ $a -gr $b]
then
if [$a -gr $c]
then
echo "$a is greater than $b and $c"
else
echo "$c is greater than $a and $b"
fi
else

9) Question - 9

Home directory :-

/home/jai/Desktop/OS

Date and Time :-

Mon Sep 27 20: 22:25 IST 2021

Users :-

jai thj 7 2021-09-27 20:19 (:0.0)

jai pts/0 2021-09-27 20:20 (:0.0)

jai pts/1 2021-09-27 20:21 (:0.0)

10) Question - 10

Enter the two numbers

7

8

Enter the operation

+

7+8 = 15

```
if [$b -gr $c]
then
 echo " $b is greater than $a and $ c"
 else
 echo " $c is greater than $ a and $ b"
 fi
fi
```

9) Write a shell program to home directory date, time and find users.

```
→ echo "Home directory"
    pwd
    echo "Date and time"
    date
    echo "Users -"
    who
```

10) Write a shell program to implement simple calculator by accepting numbers from user

```
→ echo "Enter two numbers"
    read a
    read b
    echo "Enter the operator"
    read op.
    case $op in
    +) echo "$a + $b = 'expr $a + $b'";;
```

11) Question - 11

10 + 5 = 15

12) Question - 12

Enter a number

5

The factorial of 5 is 120

-) echo "$a - $b = 'expr $a - $b'" ;;
x) echo "$a x $b = 'expr $a * $b'" ;;
\) echo "$a / $b = 'expr $a / $b'" ;;
*) echo "Invalid entry" ;;
esac

11) Write a shell program to implement a simple calculator using command arguement passage

→ case $2 in
+) echo "$1 + $3 = 'expr $1 + $3'" ;;
-) echo "$1 - $3 = 'expr $1 - $3'" ;;
x) echo "$1 x $3 = 'expr $1 * $3'" ;;
/) echo "$1 / $3 = 'expr $1 / $3'" ;;
*) echo "Invalid entry" ;;
esac

12) Write a shell program to find the factorial of a given number.
→ echo "Enter a number"
read a
fact = 1
if [ $a -eq 0 ]
then
echo "The factorial of 0 is 1"
else
b = $a
while [ $b -gt 1 ]

13) Question - 13

Enter the number

5

Enter the range

10

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

6 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

```
do
fact = $((fact * b))
n = $((b-1))
done
fi
echo "factorial of $a is $fact"
```

13) Write a shell program to print the multiplication table of a given number

```
→ echo "Enter the number"
read a
echo "Enter the range"
read range
i=1
while [ $i -lt $range]
do
   temp = 'expr $a \* $i'
   echo "$a * $i = $temp"
   i = 'expr $i +1'
done
```

14 Write a shell program to reverse a given number and find the sum of digits

```
→ echo "Enter the number"
read a
s = 0
r = 0
```

4) Question - 14

Enter the number : 151

Reverse of 151 : 151

Sum : 7

5) Question - 15

Enter the Limit

5

Fibanacci series is

0

1

1

2

3

```
while [ $a -gt 0]
do
    k = $(( $a % 10))
    r = $ (($r * 10 + $k))
    a = $ (( $a / 10))
    s = $ (( $s + $k))
done
echo " Reverse of $a = $r "
echo " Sum = $s "
```

15 Write a shell program to print fibonacci series

→
```
echo " Enter the limit "
read limit
i = 2
first = 0
second = 1
echo " Fibonacci series is "
echo " $first "
echo " $second "
while [ $i -lt $limit ]
do
    i = 'expr $ i + 1 '
    third = 'expr $ first + $ second'
    echo " $third "
    first = $second
    second = $ third
done
```

/ /

RESULT

Familiarised with shell programs.

END OF EXP 2

EXP 3 scanned as

Left — Right

PROGRAM

```
#include <stdio.h>
#include <sys\stat.in>
#include <time h>
int main()
{
    struct stat buf;
    stat ("statfile.txt", & buf);
    printf ("File size : %d bytes \n", buf.st_size);
    printf ("File permission : %o n", buf.st_mod);
    printf ("Last modified time : %s ", ctime (& buf.st_mtime));
    return 0;
}
```

OUTPUT

File size : 60 bytes
File permission : 100666
Last modified time Thu Jun 24 22:10:30 2021

/ /

Exp No: 3

# SYSTEM CALLS OF UNIX SYSTEM

a) start()

## AIM

Write a C program to get properties of a file using start()

## ALGORITHM

1. Start
2. Define struct called buf of type stat
3. Call stat function with file path and pointer to buf as parameter
4. Print buf.st_size as integer
5. Print buf.st_mod as octal
6. Print value of buf.st_mtime converted to string using ctime function
7. STOP.

## RESULT

C program to get properties of file using stat() is executed successfully.

## PROGRAM

```
#include <stdio.h>
# include <unistd.h>
 int main()
{
 int childpid, x = 0;
 Printf (" Before it forks \n ");
 childpid = fork ();
 if (childpid == 0)
 {  printf (" Child process : x = %d \n", ++x );
 }
 else
 {  printf (" parent process : x = %d \n ", -- x);
 }
 return 0;
}
```

## OUTPUT

Before it forks

parent process : x = -1

child process : x = 1

/ /

b) fork ()

AIM

Write a C program to create a child process by duplicating the calling parent process using fork ()

ALGORITHM

1. Start
2. Declare childpid
3. Initialize $x = 0$
4. Call fork and assign its certain value to childpid
5. If childpid = 0 then
   5.1 set $x = x + 1$
   5.2 print 'child' x
6. else
   6.1 set $x = x - 1$
   6.2 print 'parent' x
7. Stop

RESULT

C program to execute a child process by duplicating the calling parent process using fork () is executed successfully.

PROGRAM

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int cat;
    printf ("Calling exec()....\n");
    cat = exec ("call_me.exe", "call_me", NULL);
    printf ("Failed exec() returned %d \n" cat);
    return 0;
}
```

OUTPUT

```
Calling exec()....
Hi, I am called by exec() from exec() family
```

/ /

2) exec ()

AIM

Write a C program to replace current process with a new process using exec() family functions.

ALGORITHM

1. Start
2. Print 'calling exec'
3. Call exec () to execute callme.exe
4. if exec() failed then
   print return value of exec ()
5. Stop

RESULT

C program to get process id of process with a new process using exec() is executed successfully.

PROGRAM

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>
int main()
{  int i;
   int pid = fork();
   for (i=1; i<6 ; i++)
   {  if (pid==0)
         printf ("child : %d \n ", i);
      else
       {  printf  ("Parent : %d \n", i);
          if (i==3)
          {
            printf ("Parent waiting ");
          }
          fflush (stdout);
          wait (NULL);
       }
} sleep (1);
   }
   if (pid == 0)
   {   printf (" Child : Exiting \n ");
   }
   return 0;
}
```

/ /

d) wait ()

AIM

Write a C program to implement wait() in C

ALGORITHM

1. Start
2. Declare pid
3. Call fork() and assign it return value to pid.
4. for i = 1 to 5
   if pid = 0 then
   print 'child', i
   else
   print 'parent', i
   if i = 3 then
   print 'parent waiting'
   call wait() to wait until child process exits
   end if
   end if
   sleep one second.
5. End for
6. if pid = 0, then print 'child exiting'
7. Stop.

# OUTPUT

Parent : 1

Child : 1

Parent : 2

Child : 2

Parent : 3

Parent : Waiting

Child : 3

Child : 4

Child : 5

Child : Exiting

Parent : 4

Parent : 5

RESULT

C program to implement wait() was executed successfully.

e) get pid()

AIM

Write a C program to get process id of process using getpid().

ALGORITHM

1. Start
2. Call fork() and assign its return value to pid
3. If pid = 0, then print getpid()
4. else
   print getpid()
   print pid
5. end if
6. Stop.

PROGRAM

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int pid = fork();
    if (pid==0)
    {  printf("Child: current pid is %d \n", getpid());
    }
    else
    {
       printf("Parent: current pid is %d \n" getpid());
       printf("Parent: child pid is %d \n", pid);
    }
  return 0;
}
```

OUTPUT

```
Parent : current pid is 64853
Parent : ~~current~~ Child pid is 69854
Child : current pid is 69854
```

/ /

## RESULT

C program to get process id of process using getpid() is executed successfully.

END OF EXP 3

EXP 4 scanned as

Left — Right

PROGRAM

```
#include<stdio.h>
void waitTime(int process[], int n, int b[], int wt[])
{
        int i;
        wt[0] = 0;
        for(i=1; i<n; i++)
        {
            wt[i] = wt[i-1] + bt[i-1];
        }
}
void turnAroundTime(int process[], int n, int bt[],
                                        int wt[], int tat[])
{
        int i;
        for(i=0; i<n; i++)
        {       tat[i]= bt[i] + wt[i];
        }
}
void avgTime(int process[], int n, int bt[])
{       int wt[n], tat[n], i, tot_wt=0, tot_tat=0;
        float avgWt, avg Tat;
        waitTime(process, n, bt, wt, tat);
        printf("\nt PROCESS \t BT \t WT \t TAT");
        for(i=0; i<n; i++)
        {   tot_wt += wt[i];
            tot_tat += tat[i];
            printf("\n\t %d \t %d \t %d \t %d", (i+1), bt[i],
                                        wt[i], tat[i]);
        }
```

Exp No: 4

# FCFS (FIRST COME FIRST SERVE) CPU SCHEDULING ALGORITHM

## AIM

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

## ALGORITHM

1. Start
2. Get the number of processes
3. Get the burst time of each processes
4. Calculation of Turn Around Time and Waiting Time
5. tot_TAT = tot_TAT + pre_TAT
6. avg_TAT = tot_TAT / num_of_proc
7. tot_WT = tot_WT + pre_WT + pre_BT
8. avg_WT = tot_WT / num_of_proc
9. Display the result
10. Stop the program.

```
    avgWt = (tot_wt/n);
    avgTat = (tot_tat/n);
    Printf("\n\n\t Average waiting time = %f", avgWt);
    Printf("\n\n\t Average turn around time = %f", avgTat);
}

Void main()
{
    Int process[10], n, Bt[10], i;
    Printf("Enter the number of processes : ");
    scanf("%d", &n);
    Printf("Enter the burst time of the processes:");
    for (i=0; i<n; i++)
    {  Printf("\n Process %d:", (i+1));
       scanf("%d", &Bt[i]);
    }
    avgTime(process, n, Bt);
}
```

OUTPUT

```
Enter the number of processes : 3
Enter the burst time of the processes :-
Process 1 : 24
Process 2 : 03
Process 3: 3
```

| PROCESS | BT | WT | TAT |
|---|---|---|---|
| 1 | 24 | 0 | 24 |
| 2 | 3 | 24 | 27 |
| 3 | 3 | 27 | 30 |

Average waiting time = 17.000000
Average Turn Around Time = 27.000000

/ /

## RESULT

The program to implement FCFS CPU scheduling was written, executed and the output was verified successfully.

END OF EXP 4

Exp 5 scanned as

Left — Right

PROGRAM

```
#include<stdio.h>
int process[10], bt[10], wt[10], tat[10];
int n, totWt=0, totTAat=0;
float avgWt, avgTat;
void waitTime (int process[], int n)
{
    int i;
    wt[0] =0;
    for (i=1; i<n; i++)
    {
        wt[i] = wt[i-1] + bt[i-1];
        totwt += wt[i];
    }
}
void turnAroundTime (int process[], int n)
{
    int i;
    for (i=0; i<n; i++)
    {
        tat[i] = wt[i] + bt[i];
        totTat += tat[i];
    }
}
void main()
{
    int i, j, temp;
    printf ("\n Enter the number of processes:");
    scanf ("%d", &n);
printf ("\n Enter the burst time of processes: ");
for (i=0; i<n; i++)
{
    printf ("\n Process %d : ", (i+1));
    scanf ("%d", &bt[i]);
}
```

Exp No: 5

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
2. Get the number of processes
3. Get the burst time of each process
4. Sort the processes based on the burst time
5. Calculation of Turn Around Time and Waiting Time
   a) tot_TAT = tot_TAT + pre_TAT
   b) avg_TAT = tot_TAT / num_of_proc
   c) tot_WT = tot_WT + pre_WT + pre_BT
   d) avg_WT = tot_WT / num_of_proc
6. Display the result
7. STOP

```
for(i=0; i<(n-1); i++)
{
    for(j=0; j<(n-i-1); j++)
    {
        if (bt[i] > bt[j+1])
        {
            temp = bt[j];
            bt[j] = bt[j+1];
            bt[j+1] = temp;

            temp = process[j];
            process[j] = process[j+1];
            process[j+1] = temp;
        }
    }
}
waitTime(process, n);
turnAroundTime(process, n);
avgWt = totWt/n;
avgTat = totTat/n;
printf("\n\t PROCESS \t BT \t WT \t TAT");
for (i=0; i<n; i++)
{
    totWt += wt[i];
    totTat += tat[i];
    printf("\n\t %d \t %d \t %d \t %d", (i+1), bt[i], wt[i],
                                          tat[i]);
}
printf("\n\n\t Average Waiting Time = %f", avgWt);
printf("\n\n\t Average Turn Around Time = %f", avgTat);

}
```

OUTPUT

Enter the number of processes : 4

Enter the burst time of processes :-

Process 1: 8

Process 2: 4

Process 3: 9

Process 4: 5

| PROCESS | BT | WT | TAT |
|---|---|---|---|
| 2 | 4 | 0 | 4 |
| 4 | 5 | 4 | 9 |
| 81 | 8 | 9 | 17 |
| 43 | 9 | 17 | 26 |

Average waiting time = 7.000000

Average Turn Around Time = 14.000000

RESULT

The program to implement the SJF CPU scheduling Algorithm was written, executed and output was verified successfully.

END OF EXP 5

EXP 6 scanned as

i) First fit → RIGHT

LEFT

ii) Best fit → RIGHT

LEFT

iii) Worst fit → RIGHT

LEFT

/ /

Exp No: 6

# MEMORY ALLOCATION METHODS FOR FIXED PARTITION

## (a) FIRST - FIT

### AIM

To create a C program to implement first fit memory allocation method.

### ALGORITHM

1. START
2. Declare the variables needed
3. Read the number of blocks and files
4. Read the size of each block and file
5. Arrange both the blocks and file size in order.
6. Check if the process ~~file~~ size is less than or equal to block size.
7. If yes, assign the corresponding block to the current process else print the current process is not allocated.
8. Display the process with blocks and allocate to respective process.
9. Stop.

/ /

## RESULT

The program to simulate ~~first-fit~~ memory allocation management method technique is successfully implemented.

# PROGRAM

a) First Fit

```
#include <stdio.h>
#include <conio.h>
#define max 25
void main()
{                                                    lowest = 1000
   int frag[max], b[max], f[max], i, j, nb, nf, temp;
   static int bf[max], ff[max];
   clrscr();
   printf("\n\t Memmemory Management Scheme
                                        - First Fit");
printf("\n Enter the number of blocks:");
scanf("%d", &nb);
printf("\n Enter the number of files:");
scanf("%d", &nf);
printf("\n Enter the size of the blocks:-\n");
for (i=i; i<=nb; i++)
{  printf("Block %d:" i);
   scanf("%d", &b[i]);
}
printf("Enter the size of the files :-\n");
for (i=1; i<=nf; i++)
{     printf("File %d", i);
      scanf("%d", &f[i]);
}
```

```
for(i=1; i<=nf; i++)
{   for(j=1; j<=nb; j++)
    {   if (bf[j] != 1)
        {   temp = b[j] - f[i];
            if (temp >= 0)
            {   if (lowest > temp)
                {   ff[i] = j;
                    lowest = temp;
                }
            }
        }
    }
    frag[i] = lowest;
    bf[ff[i]] = 1;
    lowest = 10000;
}
printf ("\n File No: \t File Size \t Block No: \t Block Size \t
                                                Fragment ");
for (i=1; i<=nf && ff[i] != 0; i++)
{   printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d ", i, f[i],
                        ff[i], b[ff[i]], frag[i]);
}   getch();
}
```

OUTPUT

```
Enter the number of blocks : 3
Enter the number of files : 2
Enter the size of the blocks :-
Block 1 : 5
Block 2 : 2
Block 3 : 7
Enter size of files :- File 1 : 1
                       File 2 : 4
```

| File No: | File Size | Block No: | Block Size | Fragment |
|---|---|---|---|---|
| 1 | 1 | 1 | 5 | 4 |
| 2 | 4 | 3 | 7 | 3 |

/ /

(b) Best Fit

AIM

To implement Best fit memory allocation method.

ALGORITHM

1. Start
2. Declare variables needed.
3. Read no: of blocks and files.
4. Read the size of each file and block.
5. Arrange both block and file size in order.
6. If process size is less than or equal to block size, assign corresponding block to current process else print current process is not allowed.
7. Display process with block and allocate to respective process
8. Stop

/ /

## RESULT

The program to simulate Best-fit memory allocation management technique is successfully implemented.

(b) Best Fit

```
#include<stdio.h>
#include<conio.h>
#define max 25
void main()
{  int frag[max], b[max], f[max], i, j, nb, nf, temp, lowest=10000;
static int bf[max], ff[max];
clrscr();
printf("\n Enter the no: of blocks : ");
scanf("%d ", &nb);
printf("\n Enter the no : of files :");
scanf("%d", &nf);
printf("\n Enter the size of the blocks :- \n");
for(i=1; i<=nb; i++)
{   printf("Block %d:", i);
    scanf("%d", &b[i]);
}
printf("Enter the size of files :- \n");
for(i=1; i<=nf; i++)
{   printf("File %d : ", i);
    scanf("%d", &f[i]);
}
for(i=1; i<=nf; i++)
{
    for(j=1; j<=nb; j++)
    {   if(bf[j]!=1)
        {   temp = b[j]-f[i];
            if(temp>=0)
            {   if(lowest>temp)
                {   ff[i]=j;
                    lowest=temp;
                }
            }
        }
    }
}
```

```
frag[i]= lowest;
bf[ff[i]] = 1;
lowest = 10000;
}
printf("\n File No: \t File Size \t Block No: \t Block Size
                                        \t Fragment ");
for (i=1; i<= nf && ff[i] !=0; i++)
{
   printf("\n %-d \t \t %-d \t \t %-d \t \t %-d \t \t %-d ",
                i, f[i], ff[i], b[ff[i]], frag[i]);
}
getch();
}
```

OUTPUT

```
Enter the number of blocks : 3
Enter the number of files : 2
Enter the size of the blocks:-
Block 1 : 5
Block 2 : 2
Block 3 : 7
Enter the size of the files:-
File 1 : 1
File 2 : 4
```

| File No | File Size | Block No | Block Size | Fragment |
|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 1 |
| 2 | 4 | 1 | 5 | 1 |

(c) Worst - Fit

AIM

Implement worst-fit memory allocation method

ALGORITHM

1. Start
2. Declare the variables needed
3. Input memory blocks and processes with size
4. Initialize all memory blocks as free.
5. Start by picking each process and the maximum block size that can be assigned to current
6. process i, find max / block size [i], block size [current], ... blocksize [n], process size [current] if found, then assign it to the current process.
6. If not, then leave the process and keep checking the ~~future~~ further processes.

RESULT

The program to simulate memory allocation arrangement using coonst fit algorithm is successfully implemented.

(c) Worst Fit

```
#include<stdio.h>
# include<conio.h>
#define max 25
 void main()
{  int frag[max], b[max], f[max], i, j, nb, nf, temp, highest
                                                    =0;
   static int bf[max], ff[max];
   clrscr();
   printf("\n\t Memory Management Scheme - Worst fit");
   printf("\n Enter the number of blocks:");
   scanf("%d", &nb);
   printf("\n Enter the number of files:");
   scanf("%d", &nf);
   printf("\n Enter the size of the blocks :\n");
   for (i=1; i<=nb; i++)
   {
      printf("Block %d:", i);
      scanf("%d", &b[i]);
   }
   printf("Enter the size of the files :- \n");
   for (i=1; i<=nf; i++)
   {
      printf("File %d:", i);
      scanf("%d", &f[i]);
   }
   for (i=1; i<=nf; i++)
   {  for (j=1; j<=nb; j++)
      {  if (bf[j] != 1)
         {
```

```
        temp = b[j] - f[i];
        if (temp >=0)
            if (highest < temp)
            {   ff[i] =j;
                highest = temp;
            }
    }
}
frag[i]= highest
bf[ff[i]] =1;
highest =0;
}
printf(" \n File no: \t File size: \t Block no: \t Bbck size :\t
                                                Fragment");
for (i=1; i<=nf ; i++)
{   printf("\n %d \t\t %d \t\t %d \t\t %d \t\t %d ", i, f[i],
            ff[i], b[ff[i]] , frag[i]);
}
getch();
}
```

OUTPUT

```
Enter the number of block : 3
Enter the number of files : 3
Enter the size of the block :-
 Block 1 : 5
 Block 2 : 2
 Block 3 : 7
Enter the size of the files :- File 1 : 1
 File 2 : 4
```

| File No. | File Size | Block No | Block Size | Fragment |
|---|---|---|---|---|
| 1 | 1 | 3 | 7 | 6 |
| 2 | 4 | 1 | 5 | 1 |

END OF EXP 6

EXP 7 scanned as

all ~~LEFTS~~ RIGHTS first

then LEFT

/ /

Exp No: 7

# PRODUCER CONSUMER PROBLEM USING SEMAPHORES

## AIM

To write a C program to simulate producer and consumer problem using sema-phores

## ALGORITHM

1. Start
2. Set mutex = 1, full = 0, empty = 3, x = 0
3. Declare the function
4. Print the menu
5. Take choice from user
6. If ch = 1,

   if mutex = 1 and empty != 0

   call producer,

   set mutex = wait (mutex)

   set full = signal (full)

   set empty = wait (empty)

   x++

   print producer produces item

   set mutex = signal (mutex)

7. If ch = 2,
   if mutex = 1 and full != 0,
      call consumer
         set mutex = wait (mutex)
         set full = wait (full)
         set empty = signal (empty)
         x--
         set mutex = signal (mutex)
8. If ch = 3,
         exit
9. End.

RESULT

The program to simulate producer and consumer problem using semaphone is successfully implemented.

# PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int mutex=1, full=0, empty=3, x=0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n 1. Producer \n 2. Consumer \n 3. Exit");
    while(1)
    {
        printf("\n Enter your choice : ");
        scanf("%d", &n);
        switch(n)
        {
            case 1: if((mutex==1) && (empty!=0))
                        producer();
                    else
                        printf("Buffer is full");
                    break;
            case 2: if((mutex==1) && (full!=0))
                        consumer();
                    else
                        printf("Buffer is empty");
                    break;
            case 3: exit(0);
                    break;
        }
    }
    return 0;
}
```

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

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

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

OUTPUT

1. Producer
2. Consumer
3. Exit

Enter your choice : 1

Producer produces item 1

Enter your choice : 2

Consumer consumes item 1

Enter your choice : 2

Buffer is empty

Enter your choice : 3