|  |  |
| --- | --- |
| **EX.NO:1** | **BASICS OF UNIX COMMANDS** |

**AIM**

To study and excute the commands in unix.

**COMMAND**

**1. Date Command:**

This command is used to display the current data and time.

Syntax :

$date

$date +%ch

Options : -

a = Abbrevated weekday.

A = Full weekday.

b = Abbrevated month.

B = Full month.

c = Current day and time.

C = Display the century as a decimal number.

d = Day of the month.

D = Day in „mm/dd/yy‟ format

h = Abbrevated month day.

H = Display the hour.

L = Day of the year.

m = Month of the year.

M = Minute.

P = Display AM or PM

S = Seconds

T = HH:MM:SS format

u = Week of the year.

y = Display the year in 2 digit.

Y = Display the full year.

Z = Time zone .

To change the format :

Syntax :

$date “+%H-%M-%S‟

**2. Calender Command:**

This command is used to display the calendar of the year or the particular month of calendar year.

Syntax :

a.$cal <year>

b.$cal <month> <year>

Here the first syntax gives the entire calendar for given year & the second Syntax gives the calendar of reserved month of that year.

**3. Echo Command:**

This command is used to print the arguments on the screen.

Syntax : $echo <text>

Multi line echo command:

To have the output in the same line, the following commands can be used.

Syntax : $echo <text\>text

To have the output in different line, the following command can be used.

Syntax : $echo “text

>line2

>line3”

**4.’who’ Command :**

It is used to display who are the users connected to our computer currently.

Syntax : $who –option’s

Options : -

H–Display the output with headers.

b–Display the last booting date or time or when the system was lastely rebooted.

**5.’who am i’ Command :**

Display the details of the current working directory.

Syntax : $who am i

**6.’tty’ Command :**

It will display the terminal name.

Syntax : $tty

**7.’Binary’ Calculator Command:**

It will change the „$‟ mode and in the new mode, arithmetic operations such as +,-,\*,/,%,n,sqrt(),length(),=, etc can be performed . This command is used to go to the binary calculus mode.

Syntax:

$bc operations

^d

$

1 base –inputbase

0 base – outputbase are used for base conversions.

Base :

Decimal = 1 Binary = 2 Octal = 8 Hexa = 16

**8.’CLEAR’ Command:**

It is used to clear the screen.

Syntax : $clear

**9.’MAN’ Command:**

It helps us to know about the particular command and its options & working. It is like “help‟ command in windows.

Syntax : $man <command name>

**10. MANIPULATION Command:**

It is used to manipulate the screen.

Syntax : $tput <argument>

Arguments :

1. Clear – to clear the screen.

2. Longname – Display the complete name of the terminal.

3. SMSO – background become white and foreground become black color.

4. rmso – background become black and foreground becomes white color.

**11. LIST Command:**

It is used to list all the contents in the current working directory.

Syntax: $ ls –options <arguments>

If the command does not contain any argument means it is working in the Current directory.

Options:

a– used to list all the files including the hidden files.

c– list all the files columnwise.

d- list all the directories.

m- list the files separated by commas.

p- list files include “/” to all the directories.

r- list the files in reverse alphabetical order.

f- list the files based on the list modification date.

x-list in column wise sorted order.

**DIRECTORY RELATED COMMANDS:**

1. **Present Working Directory Command :**

To print the complete path of the current working directory.

Syntax : $pwd

1. **MKDIR Command :**

To create or make a new directory in a current directory.

Syntax : $mkdir <directory name>

1. **CD Command :**

To change or move the directory to the mentioned directory.

Syntax : $cd <directory name.

1. **RMDIR Command :**

To remove a directory in the current directory & not the current directory itself.

Syntax : $rmdir <directory name>

**FILE RELATED COMMANDS:**

**1. CREATE A FILE:**

To create a new file in the current directory we use CAT command.

Syntax :

$cat > filename

The > symbol is redirectory we use cat command.

**2. DISPLAY A FILE:**

To display the content of file mentioned we use CAT command without “>” operator.

Syntax :

$cat filename

**3. COPYING CONTENTS:**

To copy the content of one file with another. If file doesnot exist, a new file is created and if the file exists with some data then it is overwritten.

Syntax :

$ cat <filename source> >> <destination filename>

$ cat <source filename> >> <destination filename> it is avoid overwriting.

Options : -

-n content of file with numbers included with blank lines.

Syntax :

$cat –n <filename>

**4. SORTING A FILE:**

To sort the contents in alphabetical order in reverse order.

Syntax :

$sort <filename >

**5. COPYING CONTENTS FROM ONE FILE TO ANOTHER:**

To copy the contents from source to destination file. So that both contents are same.

Syntax :

$cp <source filename> <destination filename>

**6. MOVE Command:**

To completely move the contents from source file to destination file and to remove the source file.

Syntax :

$ mv <source filename> <destination filename>

**7. REMOVE Command:**

To permanently remove the file we use this command.

Syntax :

$rm <filename>

**8. WORD Command:**

To list the content count of no of lines, words, characters.

Syntax :

$wc <filename>

Options :

-c – to display no of characters.

-l – to display only the lines.

-w – to display the no of words.

**9. LINE PRINTER:**

To print the line through the printer, we use lp command.

Syntax :

$lp <filename>

**10. PAGE Command:**

This command is used to display the contents of the file page wise & next page can be viewed by pressing the enter key.

Syntax :

$pg <filename>

**FILTERS AND PIPES**

**HEAD:** It is used to display the top ten lines of file.

Syntax: $head<filename>

**TAIL:** This command is used to display the last ten lines of file.

Syntax: $tail<filename>

**PAGE:** This command shows the page by page a screen full of information is displayed after which the page command displays a prompt and passes for the user to strike the enter key to continue scrolling.

Syntax: $ls –a\p

**MORE:** It also displays the file page by page .To continue scrolling with more command, press the space bar key.

Syntax: $more<filename>

**GREP:** This command is used to search and print the specified patterns from the file.

Syntax: $grep [option] pattern <filename>

**SORT:** This command is used to sort the data in some order.

Syntax: $sort<filename>

**PIPE:** It is a mechanism by which the output of one command can be channeled into the input of another command.

Syntax: $who | wc -l

**TR:** The tr filter is used to translate one set of characters from the standard inputs to another.

Syntax: $tr “[a-z]” “[A-Z]”

**COMMUNICATION THROUGH UNIX COMMANDS**

**1. MESG**

Description: The message command is used to give permission to other users to send message to your terminal.

Syntax: $mesg y

**2. Command: WRITE**

Description: This command is used to communicate with other users, who are logged in at the same time.

Syntax: $write <user name>

**3. Command: WALL**

Description: This command sends message to all users those who are logged in using the unix server.

Syntax: $wall <message>

**4. Command: MAIL**

Description: It refers to textual information, which can be transferred from one user to another

Syntax: $mail <user name>

**5. Command: REPLY**

Description: It is used to send reply to specified user.

Syntax: $reply<user name>**RESULT:**

Thus the basic unix commands has been studied and executed successfully.

**EX.NO:2(A) FORK SYSTEM CALL**

**AIM:**

To create a new child process using fork system call.

**ALGORITHM:**

1.Declare a variable x to be shared by both child and parent.

2. Create a child process using fork system call.

3.If return value is -1 then

a. Print "Process creation unsuccessfull"

b. Terminate using exit system call.

4. If return value is 0 then

a. Print "Child process"

b. Print process id of the child using getpid system call

c. Print value of x

d. Print process id of the parent using getppid system call

5. Otherwise

a. Print "Parent process"

b. Print process id of the parent using getpid system call

c. Print value of x

d. Print process id of the shell using getppid system call.

6. Stop.

**PROGRAM:**

/\* Process creation - fork.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

main()

{

pid\_t pid;

int x = 5;

pid = fork();

x++;

if (pid < 0)

{

printf("Process creation error");

exit(-1);

}

else if (pid == 0)

{

printf("Child process:");

printf("\nProcess id is %d", getpid());

printf("\nValue of x is %d", x);

printf("\nProcess id of parent is %d\n", getppid());

}

else

{

printf("\nParent process:");

printf("\nProcess id is %d", getpid());

printf("\nValue of x is %d", x);

printf("\nProcess id of shell is %d\n", getppid());

}

}

**OUTPUT:**

$ gcc fork.c

$ ./a.out

Child process:

Process id is 19499

Value of x is 6

Process id of parent is 19498

Parent process:

Process id is 19498

Value of x is 6

Process id of shell is 3266

**RESULT:**

Thus a child process is created with copy of its parent's address space.

**EX:NO:2(B) EXEC WAIT, EXIT SYSTEM CALL**

**AIM**

To load an executable program in a child processes using exec, wait, exit system call.

**ALGORITHM**

1.If no. of command line arguments not equal to 3 then stop.

2.Create a child process using fork system call.

3. If return value is -1 then

a . Print "Process creation unsuccessfull".

b. Terminate using exit system call.

4. If return value is > 0 then

a. Suspend parent process until child completes using wait system call

b. Print "Child Terminated".

c. Terminate the parent process.

5. If return value is 0 then

a. Print "Child starts".

b. Load the program in the given path into child process using exec

system call.

c. If return value of exec is negative then print the exception and stop.

d. Terminate the child process.

6. Stop

**PROGRAM**

/\* Load a program in child process - exec.c \*/

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

main(int argc, char\*argv[])

{

pid\_t pid;

int i;

if (argc != 3)

{

printf("\nInsufficient arguments to load program");

printf("\nUsage: ./a.out <path> <cmd>\n");

exit(-1);

}

switch(pid = fork())

{

case -1:

printf("Fork failed");

exit(-1);

case 0:

printf("Child process\n");

i = exec(argv[1], argv[2], 0);

if (i < 0)

{

printf("%s program not loaded using exec system call\n", argv[2]);

exit(-1);

}

default:

wait(NULL);

printf("Child Terminated\n");

exit(0);

}

}

**OUTPUT**

$ gcc exec.c

$ ./a.out

Insufficient arguments to load program

Usage: ./a.out <path> <cmd>

$ ./a.out /bin/ls ls

Child process

a.out cmdpipe.c consumer.c

dirlist.c ex6a.c ex6b.c

ex6c.c ex6d.c exec.c

fappend.c fcfs.c fcreate.c

fork.c fread.c hello

list list.c pri.c

producer.c rr.c simls.c

sjf.c stat.c wait.c

Child Terminated

$ ./a.out /bin/who who

Child process

who program not loaded using exec system call

Child Terminated

$ ./a.out /usr/bin/who who

Child process

vijai pts/0 2013-04-24 15:48 (192.168.144.1)

Child Terminated

**RESULT**

Thus the child process loads a binary executable file into its address space.

**EX:NO:2(C) GETPID AND EXIT SYSTEM CALL**

**AIM**

To load an executable program in a child processes using getpid, exit system call.

**ALGORITHM**

1.Declare a variable x to be shared by both child and parent.

2. Create a child process using fork system call.

3.If return value is -1 then

a. Print "Process creation unsuccessfull".

b. Terminate using exit system call.

4. If return value is 0 then

a. Print "Child process".

b. Print process id of the child using getpid system call.

c. Print value of x.

d. Print process id of the parent using getppid system call.

5. Otherwise

a. Print "Parent process".

b. Print process id of the parent using getpid system call.

c. Print value of x.

d. Print process id of the shell using getppid system call.

6. Stop.

**PROGRAM**

#include<stdio.h>

#include<unistd.h>

main()

{

int pid,pid1,pid2;

pid=fork();

if(pid==-1)

{

printf(“ERROR IN PROCESS CREATION \n”);

exit(1);

}

if(pid!=0)

{

pid1=getpid();

printf(“\n the parent process ID is %d\n”, pid1);

}

else

{

pid2=getpid();

printf(“\n the child process ID is %d\n”, pid2);

}

}

**OUTPUT**

Cc fork.c

./a.out

THE CHILD PROCESS ID IS 8640

THE PARENT PROCESS ID IS 8644

**RESULT**

Thus the child process loads a binary executable file into its address space.

**EX:NO: 2D OPEN SYSTEM CALL**

**AIM**

To create a file and to write contents.

**ALGORITHM**

1. Declare a character buffer buf to store 100 bytes.
2. Get the new filename as command line argument.
3. Create a file with the given name using open system call with O\_CREAT and O\_TRUNC options.
4. Check the file descriptor.
5. If file creation is unsuccessful, then stop.
6. Write length of buf onto file using write system call.
7. Get input from the console until user types Ctrl+D.
8. Read 100 bytes (max.) from console and store onto buf using read system call
9. Close the file.
10. Stop

**PROGRAM**

/\* File creation - fcreate.c \*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <fcntl.h>

main(int argc, char \*argv[])

{

int fd, n, len;

char buf[100];

if (argc != 2)

{

printf("Usage: ./a.out <filename>\n");

exit(-1);

}

fd = open(argv[1], O\_WRONLY|O\_CREAT|O\_TRUNC, 0644);

if(fd < 0)

{

printf("File creation problem\n");

exit(-1);

}

printf("Press Ctrl+D at end in a new line:\n");

while((n = read(0, buf, sizeof(buf))) > 0)

{

len = strlen(buf);

write(fd, buf, len);

}

close(fd);

}

**OUTPUT**

$ gcc fcreate.c

$ ./a.out hello

File I/O

Open system call is used to either open or create a file.

creat system call is used to create a file. It is seldom used.

**RESULT**

Thus a file has been created with input from the user. The process can be verified by using cat command.

**EX:NO: 2E STAT SYSTEM CALL**

**AIM**

To display file status using stat system call.

**ALGORITHM**

1. Get filename as command line argument.
2. If filename does not exist then stop.
3. Call stat system call on the filename that returns a structure.
4. Display members st\_uid, st\_gid, st\_blksize, st\_block, st\_size, st\_nlink , etc.,
5. Convert time members such as st\_atime, st\_mtime into time using ctime function.
6. Compare st\_mode with mode constants such as S\_IRUSR, S\_IWGRP, S\_IXOTH and display file permissions.
7. Stop.

**PROGRAM**

/\* File status - stat.c \*/

#include <stdio.h>

#include <sys/stat.h>

#include <stdlib.h>

#include <time.h>

int main(int argc, char\*argv[])

{

struct stat file;

int n;

if (argc != 2)

{

printf("Usage: ./a.out <filename>\n");

exit(-1);

}

if ((n = stat(argv[1], &file)) == -1)

{

perror(argv[1]);

exit(-1);

}

printf("User id : %d\n", file.st\_uid);

printf("Group id : %d\n", file.st\_gid);

printf("Block size : %d\n", file.st\_blksize);

printf("Blocks allocated : %d\n", file.st\_blocks);

printf("Inode no. : %d\n", file.st\_ino);

printf("Last accessed : %s", ctime(&(file.st\_atime)));

printf("Last modified : %s", ctime(&(file.st\_mtime)));

printf("File size : %d bytes\n", file.st\_size);

printf("No. of links : %d\n", file.st\_nlink);

printf("Permissions : ");

printf( (S\_ISDIR(file.st\_mode)) ? "d" : "-");

printf( (file.st\_mode & S\_IRUSR) ? "r" : "-");

printf( (file.st\_mode & S\_IWUSR) ? "w" : "-");

printf( (file.st\_mode & S\_IXUSR) ? "x" : "-");

printf( (file.st\_mode & S\_IRGRP) ? "r" : "-");

printf( (file.st\_mode & S\_IWGRP) ? "w" : "-");

printf( (file.st\_mode & S\_IXGRP) ? "x" : "-");

printf( (file.st\_mode & S\_IROTH) ? "r" : "-");

printf( (file.st\_mode & S\_IWOTH) ? "w" : "-");

printf( (file.st\_mode & S\_IXOTH) ? "x" : "-");

printf("\n");

if(file.st\_mode & S\_IFREG)

printf("File type : Regular\n");

if(file.st\_mode & S\_IFDIR)

printf("File type : Directory\n");

}

**OUTPUT**

$ gcc stat.c

$ ./a.out fork.c

User id : 0

Group id : 0

Block size : 4096

Blocks allocated : 8

Inode no. : 16627

Last accessed : Fri Feb 22 21:57:09 2013

Last modified : Fri Feb 22 21:56:13 2013

File size : 591 bytes

No. of links : 1

Permissions : -rw-r--r--

File type : Regular

**RESULT**

Thus the attributes of a file is displayed using stat system call has been executed successfully .

**EX:NO:3(A) PROGRAM FOR SIMULATION OF LS UNIX COMMANDS**

**AIM**

To implement a program for simulation of ls unix commands.

**ALGORITHM**

1. Start the program
2. Use the available directory, if not create a directory.
3. Open the directory and list the files.
4. Stop the program

**PROGRAM**

#include<stdio.h>

#include<dirent.h>

main(int argc, char \*\*argv)

{

DIR \*dp;

struct dirent \*link;

dp=opendir(argv[1]);

printf(“\n contents of the directory %s are \n”, argv[1]);

while((link=readdir(dp))!=0)

printf(“%s”,link->d\_name);

closedir(dp);

}

**OUTPUT**

Cc list.c

./a.out os

CONTENTS OF THE DIRECTORY OS ARE

Priority.c

Robin.c

copy

**RESULT**

Thus the program to implement the simulation of ls unix commands has been executed successfully .

**EX:NO:3(B) PROGRAM FOR SIMULATION OF GREP UNIX COMMANDS**

**AIM**

To implement the program for simulation of grep unix commands.

**ALGORITHM**

1. Start the program.
2. Enter the regular expression.
3. The grep command searches for matching a regular expression against text in a file, multiple files or a stream of input.
4. It searches for the pattern of text that you specify on the command line and prints output
5. Stop the program.

**PROGRAM**

#include<stdio.h>

#include<string.h>

#define max 1024

void usage()

{

printf(“usage:\t. /a.out filename word \n “);

}

int main(int argc, char \*argv[])

{

FILE \*fp;

char fline[max];

char \*newline;

int count=0;

int occurrences=0;

if(argc!=3)

{

usage();

exit(1);

}

if(!(fp=fopen(argv[1],”r”)))

{

printf(“grep: couldnot open file : %s \n”,argv[1]);

exit(1);

}

while(fgets(fline,max,fp)!=NULL)

{

count++;

if(newline=strchr(fline, ‘\n’))

\*newline=’\0’;

if(strstr(fline,argv[2])!=NULL)

{

printf(“%s: %d %s \n”, argv[1],count, fline);

occurrences++;

}

}

}

**OUTPUT**

CAT>SAMP

ONE

ONE TWO

THREE FOUR

Cc grep.c

./a.out samp one

Samp:1 one

Samp:2 one two

**RESULT**

Thus the program to implement the simulation of grep unix commands has been executed successfully .

|  |  |
| --- | --- |
| **EX.No:4(A)** | **SHELL PROGRAMMING**  **CONCATENATION OF TWO STRINGS** |

**AIM**

To write a shell program to concatenate two strings.

**ALGORITHM**

Step1: Enter into the vi editor and go to the insert mode for entering the code

Step2: Read the first string.

Step3: Read the second string

Step4: Concatenate the two strings

Step5: Enter into the escape mode for the execution of the result and verify the output

**PROGRAM**

echo “enter the first string”

read str1

echo “enter the second string”

read str2

echo “the concatenated string is” $str1 $str2

**OUTPUT**

Enter first string: Hello

Enter first string: World

The concatenated string is Hello World

**RESULT**

Thus the shell program to concatenate two strings is executed and output is verified successfully.

|  |  |
| --- | --- |
| **EX.No:4(B)** | **COMPARISON OF TWO STRINGS** |

**AIM**

To write a shell program to compare the two strings.

**ALGORITHM**

Step1: Enter into the vi editor and go to the insert mode for entering the code

Step2: Read the first string.

Step3: Read the second string

Step4: Compare the two strings using the if loop

Step5: If the condition satisfies then print that two strings are equal else print two strings are not equal.

Step6: Enter into the escape mode for the execution of the result and verify the output

**PROGRAM**

echo “enter the first string”

read str1

echo “enter the second string”

read str2

if [ $str1 = $str2 ]

then

echo “strings are equal”

else

echo “strings are unequal”

fi

**OUTPUT 1:**

Enter first string: hai

Enter second string: hai

The two strings are equal

**OUTPUT 2:**

Enter first string: hai

Enter second string: cse

The two strings are not equal

**RESULT:**

Thus the shell program to compare the two strings is executed and output is verified successfully.

|  |  |
| --- | --- |
| **EX.NO:4(C)** | **MAXIMUM OF THREE NUMBERS** |

**AIM**

To write a shell program to find greatest of three numbers.

**ALGORITHM**

Step1: Declare the three variables.

Step2: Check if A is greater than B and C.

Step3: If so print A is greater.

Step4: Else check if B is greater than C.

Step5: If so print B is greater.

Step6: Else print C is greater.

**PROGRAM**

echo "enter A"

read a

echo "enter B"

read b

echo "enter C"

read c

if [ $a -gt $b -a $a -gt $c ]

then

echo "A is greater"

elif [ $b -gt $a -a $b -gt $c ]

then

echo "B is greater"

else

echo "C is greater"

fi

**OUTPUT**

Enter A:23

Enter B:45

Enter C:67

C is greater

**RESULT:**

Thus the shell program to find the maximum of three numbers is executed and output is verified successfully.

|  |  |
| --- | --- |
| **EX.No:4(D)** | **FIBONACCI SERIES** |

**AIM**

To write a shell program to generate Fibonacci series.

**ALGORITHM**

Step 1 : Initialise a to 0 and b to 1.

Step 2 : Print the values of 'a' and 'b'.

Step 3 : Add the values of 'a' and 'b'. Store the added value in variable 'c'.

Step 4 : Print the value of 'c'.

Step 5 : Initialise 'a' to 'b' and 'b' to 'c'.

Step 6 : Repeat the steps 3,4,5 till the value of 'a' is less than 10.

**PROGRAM**

echo enter the number

read n

a=-1

b=1

i=0

while [ $i –le $n ]

do

t=`expr $a + $b`

echo $t

a=$b

b=$t

i=`expr $i + 1

done

**OUTPUT**

Enter the no: 5

0

1

1

2

3

5

**RESULT:**

Thus the shell program to find the fibonacci series is executed and output is verified successfully.

|  |  |
| --- | --- |
| **EX.No:4(E)** | **ARITHMETIC OPERATIONS USING CASE** |

**AIM**

To write a shell program to perform the arithmetic operations using case.

**ALGORITHM**

Step 1 : Read the input variables and assign the value

Step 2 : Print the various arithmetic operations which we are going to perform

Step 3 : Using the case operator assign the various functions for the arithmetic operators.

Step 4 : Check the values for all the corresponding operations.

Step 5 : Print the result and stop the execution.

**PROGRAM**

echo 1.Addition

echo 2.Subraction

echo 3.Multiplication

echo 4.Division

echo enter your choice

read a

echo enter the value of b

read b

echo enter the value of c

read c

echo b is $b c is $c

case $a in

1)d=`expr $b + $c`

echo the sum is $d

;;

2)d=`expr $b - $c`

echo the difference is $d

;;

3)d=`expr $b \\* $c`

echo the product is $d

;;

4)d=`expr $b / $c`

echo the quotient is $d

;;

esac

**OUTPUT**

1.Addition

2.Subraction

3.Multiplication

4.Division

Enter your choice:1

Enter the value of b:3

Enter the value of c:4

b is 3 c is 4

the sum is 7

b is 3 c is 4

the sum is 7

**RESULT**

Thus the shell program to perform arithmetic operations using case is executed and output is verified successfully.

**IMPLEMENTATION OF FCFS SCHEDULING**

|  |  |
| --- | --- |
| Ex.No:5(A) |  |

**AIM**

To write a program for first come first serve (FCFS) scheduling algorithm.

**ALGORITHM**

1. Start the program the program.
2. Declare and Initialize the variables.
3. Get the number of process, its burst time and arrival time.
4. Calculate the average turnaround time and waiting time of each process.
   * + Twt=Twt+(Wt[i]-A[i]);
     + Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
     + Att=(float)Ttt/n;
     + Awt=(float)Twt/n;
5. Display the result.
6. Stop the program.

**PROGRAM**

#include<stdio.h>

#include<string.h>

int n,Bu[20],Twt,Ttt,A[10],Wt[10],w;

float Awt,Att;

char pname[20][20],c[20][20];

void Getdata();

void Gantt\_chart();

void Calculate();

void fcfs();

void Getdata()

{

int i;

printf("\n Enter the number of processes: ");

scanf("%d",&n);

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

{

fflush(stdin);

printf("\n\n Enter the process name: ");

scanf("%s",&pname[i]);

printf("\n Enter The BurstTime for Process %s = ",pname[i]);

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

printf("\n Enter the Arrival Time for Process %s = ",pname[i]);

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

}

}

void Gantt\_chart()

{

int i;

printf("\n\n\t\t\tGANTT CHART\n");

printf("\n---------------------------------------------------------\n");

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

printf("|\t%s\t",pname[i]);

printf("|\t\n");

printf("\n---------------------------------------------------------\n");

printf("\n");

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

printf("%d\t\t",Wt[i]);

printf("%d",Wt[n]+Bu[n]);

printf("\n---------------------------------------------------------\n");

printf("\n");

}

void Calculate()

{

int i;

Wt[1]=0;

for(i=2;i<=n;i++)

{

Wt[i]=Bu[i-1]+Wt[i-1];

}

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

{

Twt=Twt+(Wt[i]-A[i]);

Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);

}

Att=(float)Ttt/n;

Awt=(float)Twt/n;

printf("\n\n Average Turn around time=%3.2f ms ",Att);

printf("\n\n AverageWaiting Time=%3.2f ms",Awt);

}

void fcfs()

{

int i,j,temp, temp1;

Twt=0;

Ttt=0;

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

{

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

{

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

{

temp=Bu[i];

temp1=A[i];

Bu[i]=Bu[j];

A[i]=A[j];

Bu[j]=temp;

A[j]=temp1;

strcpy(c[i],pname[i]);

strcpy(pname[i],pname[j]);

strcpy(pname[j],c[i]);

}

}

}

Calculate();

Gantt\_chart();

}

void main()

{

int ch;

Getdata();

fcfs();

}

**OUTPUT**

[2cse@localhost ~]$ cc fcfs.c

[2cse@localhost ~]$ ./a.out

Enter the number of processes: 3

Enter the process name: p1

Enter The BurstTime for Process p1 = 4

Enter the Arrival Time for Process p1 = 0

Enter the process name: p2

Enter The BurstTime for Process p2 = 6

Enter the Arrival Time for Process p2 = 1

Enter the process name: p3

Enter The BurstTime for Process p3 = 8

Enter the Arrival Time for Process p3 = 2

Average Turn around time=9.67 ms

AverageWaiting Time=3.67 ms

GANTT CHART

------------------------------------------

| p1 | p2 | p3 |

------------------------------------------

0 4 10 18

------------------------------------------

**RESULT**

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

|  |  |
| --- | --- |
| **EX.NO:5(B)** | **IMPLEMENTATION OF SJF SCHEDULING** |

**AIM**

To write a program for shortest job first (SJF) scheduling algorithm.

**ALGORITHM**

1. Start the program.
2. Declare and Initialize the variables.
3. Get the number of process and its burst time.
4. Re-arrange the burst times using “BUBBLE SORT” in ascending order.
5. Calculate the average turnaround time and waiting time of each process.
   * Twt=Twt+(Wt[i]-A[i]);
   * Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);
   * Att=(float)Ttt/n;
   * Awt=(float)Twt/n;
6. Display the results.
7. Stop the program.

**PROGRAM**

#include<stdio.h>

#include<string.h>

int Twt,Ttt,A[20],Wt[20],n,Bu[20],B[10];

float Att,Awt;

char pname[20][20];

void Getdata();

void Gantt\_chart();

void Sjf();

void Getdata()

{

int i;

printf("\n Enter the number of processes: ");

scanf("%d",&n);

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

{

fflush(stdin);

printf("\n\n Enter the process name: ");

scanf("%s",&pname[i]);

printf("\n Enter The BurstTime for Process %s = ",pname[i]);

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

printf("\n Enter the Arrival Time for Process %s = ",pname[i]);

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

}

}

void Gantt\_chart()

{

int i;

printf("\n\nGANTT CHART");

printf("\n----------------------------------------------------------\n");

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

printf("|\t%s\t",pname[i]);

printf("|\t\n");

printf("\n-----------------------------------------------------------\n");

printf("\n");

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

printf("%d\t\t",Wt[i]);

printf("%d",Wt[n]+B[n]);

printf("\n------------------------------------------------------------\n");

printf("\n");

}

void Sjf()

{

int w,t,i,Tt=0,temp,j;

char S[10],c[20][20];

int temp1;

printf("\n\n SHORTEST JOB FIRST SCHEDULING ALGORITHM \n\n");

Twt=Ttt=0;

w=0;

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

{

B[i]=Bu[i];

S[i]='T';

Tt=Tt+B[i];

}

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

{

for(j=3;j<=n;j++)

{

if(B[j-1]>B[j])

{

temp=B[j-1];

temp1=A[j-1];

B[j-1]=B[j];

A[j-1]=A[j];

B[j]=temp;

A[j]=temp1;

strcpy(c[j-1],pname[j-1]);

strcpy(pname[j-1],pname[j]);

strcpy(pname[j],c[j-1]);

}

}

}

Wt[1]=0;

w=w+B[1];

t=w;

S[1]='F';

while(w<Tt)

{

i=2;

while(i<=n)

{

if(S[i]=='T'&&A[i]<=t)

{

Wt[i]=w;

S[i]='F';

w=w+B[i];

t=w;

i=2;

}

else

i++;

}

}

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

{

Twt=Twt+(Wt[i]-A[i]);

Ttt=Ttt+((Wt[i]+Bu[i])-A[i]);

}

Att=(float)Ttt/n;

Awt=(float)Twt/n;

printf("\n\n Average Turn around time=%3.2f ms ",Att);

printf("\n\n AverageWaiting Time=%3.2f ms",Awt);

Gantt\_chart();

}

void main()

{

Getdata();

Sjf();

}

**OUTPUT**

[2cse@localhost ~]$ cc sjf.c

[2cse@localhost ~]$ ./a.out

Enter the number of processes: 3

Enter the process name: p1

Enter The BurstTime for Process p1 = 5

Enter the Arrival Time for Process p1 = 0

Enter the process name: p2

Enter The BurstTime for Process p2 = 8

Enter the Arrival Time for Process p2 = 1

Enter the process name: p3

Enter The BurstTime for Process p3 = 3

Enter the Arrival Time for Process p3 = 2

SHORTEST JOB FIRST SCHEDULING ALGORITHM

Average Turn around time=8.67 ms

AverageWaiting Time=3.33 ms

GANTT CHART

-------------------------------------------

| p1 | p3 | p2 |

--------------------------------------------

0 5 8 16

---------------------------------------------

**RESULT**

Thus the program to implement the SJF (Shortest Job First) scheduling Algorithm was written, executed and the output was verified successfully.

|  |  |
| --- | --- |
| **EX.NO:5(C)** | **IMPLEMENTATION OF PRIORITY SCHEDULING** |

**AIM**

To write the program to perform priority scheduling.

**ALGORITHM**

1. Start the program.
2. Get the number of processes, their burst time and priority.
3. Initialize the waiting time for process 1 is 0.
4. Based upon the priority processes are arranged.
5. The waiting time and turnaround time for other processes are calculated as

* twait=twait+wait[i];
* totl=totl+tta;

1. The waiting time and turnaround time for all the processes are summed and then the average waiting time and turnaround time are calculated.

* wavg=twait/n;
* tavg=totl/n;

1. The average waiting time and turnaround time are displayed.
2. Stop the program.

**PROGRAM**

#include<stdio.h>

main()

{

int prs[10],prty[10];

int n,i,j,twait=0,tta=0,tot=0,totl=0,temp,temp1;

int wait[10],bst[10],p[10];

float wavg,tavg;

printf("\n\t\tPRIORITY SCHEDULING");

printf("\n enter the no of process:");

scanf("%d",&n);

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

{

printf("\n enter the process name: P");

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

printf("\n enter the burstTime");

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

printf("\n enter the priority");

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

}

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

{

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

{

if(prty[i]>=prty[j])

{

temp=bst[i];

bst[i]=bst[j];

bst[j]=temp;

temp1=prs[i];

prs[i]=prs[j];

prs[j]=temp1;

}

}

}

printf ("\n\n\n sorted process with priority");

printf("\n-------------------------------------");

printf("\n Processname \t burst Time \n");

printf("\n-------------------------------------\n");

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

{

printf("\tp%d \t %d \n\n",prs[i],bst[i]);

}

printf("\n\n");

printf("\n-------------------------------------------------------------");

printf("\n Processor\tBursttime\tTurnaroundtime\tWaitingtime\n");

printf("\n-------------------------------------------------------------");

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

{

tta=tta+bst[i];

wait[i]=tta-bst[i];

printf("\n\tp%d\t%d\t\t%d\t\t%d",prs[i],bst[i],tta,wait[i]);

twait=twait+wait[i];

totl=totl+tta;

}

wavg=twait/n;

tavg=totl/n;

printf("\n\n\n\t\t\*\*\*GRANTT CHART\*\*\*\n");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

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

printf("|\tp%d\t",prs[i]);

printf("|\t\n");

printf("\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n");

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

printf("%d\t\t",wait[i]);

printf("%d",wait[n]+bst[n]);

printf("\n");

printf("\n\n Total burst time is :%d",tta);

printf("\n\n Total turnaround time :%d",totl);

printf("\n\n The average turnaround time:%f",tavg);

printf("\n\n Total waiting time :%d",twait);

printf("\n\n The avg waiting time :%f",wavg);

}

**OUTPUT**

[2cse@localhost ~]$ cc pri.c

[2cse@localhost ~]$ ./a.out

PRIORITY SCHEDULING

enter the no of process:5

enter the process name: P1

enter the burstTime5

enter the priority2

enter the process name: P2

enter the burstTime6

enter the priority4

enter the process name: P3

enter the burstTime2

enter the priority3

enter the process name: P4

enter the burstTime8

enter the priority1

enter the process name: P5

enter the burstTime7

enter the priority5

sorted process with priority

-------------------------------------

Processname burst Time

------------------------------------

p4 8

p1 5

p3 2

p2 6

p5 7

-------------------------------------------------------------

Processor Bursttime Turnaroundtime Waitingtime

-------------------------------------------------------------

p4 8 8 0

p1 5 13 8

p3 2 15 13

p2 6 21 15

p5 7 28 21

\*\*\*GRANTT CHART\*\*\*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| p4 | p1 | p3 | p2 | p5|

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0 8 13 15 21 28

Total burst time is :28

Total turnaround time :85

The average turnaround time:17.000000

Total waiting time :57

The avg waiting time :11.000000

**RESULT**

Thus the program for priority scheduling was executed successfully.

|  |  |
| --- | --- |
| **EX.NO:5(D)** | IMPLEMENTATION OF ROUND ROBIN SCHEDULING |

**AIM**

To write a program to implement the Round Robin (RR) CPU scheduling.

**ALGORITHM**

1. Start the process
2. Get the number of Processes
3. Get the value for burst time for individual processes
4. Get the value for time quantum
5. Make the CPU scheduler go around the ready queue allocating CPU to each process for the time interval specified
6. Make the CPU scheduler pick the first process and set time to interrupt after quantum. And after it's expiry dispatch the process
7. If the process has burst time less than the time quantum then the process is released by the CPU
8. If the process has burst time greater than time quantum then it is interrupted by the OS and the process is put to the tail of ready queue and the schedule selects next process from head of the queue
9. Calculate the total and average waiting time and turnaround time and display the results
10. Stop the process

**PROGRAM**

#include<stdio.h>

struct process

{

int at,bt,wt,tat,st,ft,flag,id,tbt;

}p[10],temp;

int n,t,save\_et[10],save\_id[10],turn,btsum;

float awt,atat;

void read();

void print();

void rndrbn();

void fifoq();

main()

{

int ch;

read();

fifoq();

rndrbn();

print();

}

void read()

{

int i;

printf("Enter no of processes:");

scanf("%d",&n);

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

{

printf("Enter arriving time ,burst time of process p%d : ",(i+1));

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

p[i].id=i+1;

p[i].wt=p[i].flag=0;

p[i].tbt=p[i].bt;

btsum+=p[i].bt;

}

printf("Enter time quantum : ");

scanf("%d",&t);

}

void fifoq()

{

int i,j;

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

{

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

{

if(p[j].at>p[j+1].at)

{

temp=p[j];

p[j]=p[j+1];

p[j+1]=temp;

}

}

}

}

void rndrbn()

{

int cnt=n;

int i=0;

int et=0;

int sum=0;

float twt=0;

float ttat=0;

while(cnt!=0)

{

if((p[i].bt)>t)

{

et=t;

p[i].bt-=t;

}

else

{

et=p[i].bt;

p[i].bt=0;

}

p[i].st=sum;

if((p[i].flag)==0)

{

p[i].wt=p[i].st-p[i].at;

p[i].flag++;

}

else

p[i].wt=p[i].wt+(p[i].st-p[i].ft);

sum=sum+et;

p[i].ft=sum;

save\_et[turn]=et;

save\_id[turn++]=p[i].id;

if((p[i].bt)==0)

{

cnt--;

}

do

{

i=(i+1)%n;

}while((p[i].bt)==0 && cnt!=0);

}

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

{

p[i].tat=p[i].wt+p[i].tbt;

twt+=p[i].wt;

ttat+=p[i].tat;

}

awt=twt/n;

atat=ttat/n;

}

void print()

{

int i,sum=0;

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

printf("---");

printf("\n");

printf("| ");

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

printf(" %p\*d |",-(save\_et[i]-1),save\_id[i]);

printf(" ");

printf("\n");

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

printf("---");

printf("\n");

printf(" ");

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

{

printf("%p\*d ",-(save\_et[i]),sum);

sum+=save\_et[i];

}

printf("%d\n",sum);

printf("\nPid\tWT\t TT");

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

{

printf("\n%d\t %d \t%d\n",p[i].id,p[i].wt,p[i].tat);

}

printf("AWT=%f\t\t ATT=%.2f\n",awt,atat);

btsum=0;

}

**OUTPUT**

[2cse@localhost ~]$ ./a.out

Enter no of processes:3

Enter arriving time ,burst time of process p1 : 0 5

Enter arriving time ,burst time of process p2 : 2 6

Enter arriving time ,burst time of process p3 : 3 8

Enter time quantum : 2

-----------------------------------------------------------------

| 1 | 2 | 3 | 1 | 2 | 3 | 1 | 2 | 3 | 3 |

-----------------------------------------------------------------

0 2 4 6 8 10 12 13 15 17 19

Pid WT TT

1 8 13

2 7 13

3 8 16

AWT=7.666667 ATT=14.00

**RESULT**

Thus the program for Round Robin Scheduling was executed and verified successfully.

|  |  |
| --- | --- |
| **EX.No:6** | **PRODUCER - CONSUMER PROBLEM USING SEMAPHORE** |

**AIM**

To implement the producer – consumer problem using semaphores

**ALGORITHM:**

1. Start the program.
2. Declare the variables in the type of pthread\_t as tid\_produce tid\_consume.
3. Declare a structure for semaphore variables.
4. During run time read the number of items to be produced and consumed.
5. Declare and define semaphore function for creation and destroy.
6. Define producer function.
7. Define consumer function.
8. Call producer and consumer function.
9. Stop the execution

**PROGRAM**

#include <stdio.h>

#include <semaphore.h>

#include <pthread.h>

#define NBUFF 10

int nitems;

struct {

int buff[NBUFF];

sem\_t mutex, nempty, nstored;

} shared;

void \*produce (void \*);

void \*consume (void \*);

int main(int argc, char \*\*argv)

{

pthread\_t tid\_produce, tid\_consume;

if(argc !=2)

{

printf("Useage: filename <nitems> ");

return 0;

}

printf ("\n\n Producer - Consumer Problem using Semaphore\n");

printf (" -------------------------------------------\n\n");

nitems=atoi(argv[1]);

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

sem\_init(&shared.nempty,0,NBUFF);

sem\_init(&shared.nstored,0,0);

pthread\_setconcurrency(2);

pthread\_create(&tid\_produce,NULL,produce,NULL);

pthread\_create(&tid\_consume,NULL,consume,NULL);

pthread\_join(tid\_produce,NULL);

pthread\_join(tid\_consume,NULL);

sem\_destroy(&shared.mutex);

sem\_destroy(&shared.nempty);

sem\_destroy(&shared.nstored);

}

void \*produce (void \*arg)

{

int i;

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

{

sem\_wait(&shared.nempty);

sem\_wait(&shared.mutex);

shared.buff[i % NBUFF]=i;

printf ("\tProducer.....");

printf ("buff[%d] = %d\n\n",i,shared.buff[i % NBUFF]);

sem\_post(&shared.mutex);

sem\_post(&shared.nstored);

sleep(3);

}

return NULL;

}

void \*consume (void \*arg)

{

int i;

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

{

sem\_wait(&shared.nstored);

sem\_wait(&shared.mutex);

printf("\tConsumer.....");

printf("buff[%d] = %d\n\n\n",i,shared.buff[i%NBUFF]);

sem\_post(&shared.mutex);

sem\_post(&shared.nempty);

sleep(3);

}

return NULL;

}

**OUTPUT**

[cse2@localhost ~]$ cc -pthread prod\_cons.c

[cse2@localhost ~]$ ./a.out 4

Producer - Consumer Problem using Semaphore

-------------------------------------------

Producer.....buff[0] = 0

Consumer.....buff[0] = 0

Producer.....buff[1] = 1

Consumer.....buff[1] = 1

Producer.....buff[2] = 2

Consumer.....buff[2] = 2

Producer.....buff[3] = 3

Consumer.....buff[3] = 3

**RESULT**

Thus the program to implement producer - consumer problem using semaphores was executed successfully.

|  |  |
| --- | --- |
| **EX.NO:7** | **IMPLEMENTATION OF SHARED MEMORY AND IPC** |

**AIM**

To write a program for interprocess communication using shared memory.

**ALGORITHM**

1. Start the program.
2. Create the child process using fork().
3. Create the shared memory for parent process using shmget() system call.
4. Now allow the parent process to write inn shared memory using shmpet pointer which is return type of shmat().
5. Now across and attach the same shared memory to the child process.
6. The data in the shared memory is read by the child process using the shnot pointer.
7. Now, detach and rebase the shared memory.
8. Stop the program.

**PROGRAM**

#include<stdio.h>

#include<sys/types.h>

#include<sys/shm.h>

#include<sys/ipc.h>

int main()

{

int child,shmid,i;

char \* shmptr;

child=fork();

if(!child)

{

shmid=shmget(2041,32,IPC\_CREAT|0666);

shmptr=shmat(shmid,0,0);

printf(“\n Parent writing\n”);

for(i=0;i<10;i++)

{

shmptr[i]=’a’+i;

putchar(shmptr[i]);

}

printf(“\n\n %s”, shmptr);

wait(NULL);

}

else

{

shmid=shmget(2041,32,0666);

shmptr=shmat(shmid,0,0);

printf(“\n Child is reading\n”);

for(i=0;i<10;i++)

putchar(shmptr[i]);

shmdt(NULL);

shmctl(shmid,IPC\_RMID,NULL);

}

return 0;

}

**OUTPUT**

[cse2@localhost ~]$ cc share.c

[cse2@localhost ~]$ ./a.out

Parent writing

abcdefghij

Child is reading

abcdefghij

**RESULT**

Thus the interprocess communication using shared memory was executed successfully.

|  |  |
| --- | --- |
| **BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE**  **EX.NO:8** |  |

**AIM**

To implement deadlock avoidance by using Banker’s Algorithm.

**ALGORITHM**

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
6. If it is possible then the system is in safe state.
7. Else system is not in safety state.
8. If the new request comes then check that the system is in safety or not if we allow the request.
9. Stop the program.

**PROGRAM**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int Max[10][10], need[10][10], alloc[10][10], avail[10], completed[10], safeSequence[10];

int p, r, i, j, process, count;

count = 0;

printf("Enter the no of processes : ");

scanf("%d", &p);

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

completed[i] = 0;

printf("\n\nEnter the no of resources : ");

scanf("%d", &r);

printf("\n\nEnter the Max Matrix for each process : ");

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

{

printf("\nFor process %d : ", i + 1);

for(j = 0; j < r; j++)

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

}

printf("\n\nEnter the allocation for each process : ");

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

{

printf("\nFor process %d : ",i + 1);

for(j = 0; j < r; j++)

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

}

printf("\n\nEnter the Available Resources : ");

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

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

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

for(j = 0; j < r; j++)

need[i][j] = Max[i][j] - alloc[i][j];

do

{

printf("\n Max matrix:\tAllocation matrix:\n");

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

{

for( j = 0; j < r; j++)

printf("%d ", Max[i][j]);

printf("\t\t");

for( j = 0; j < r; j++)

printf("%d ", alloc[i][j]);

printf("\n");

}

process = -1;

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

{

if(completed[i] == 0)//if not completed

{

process = i ;

for(j = 0; j < r; j++)

{

if(avail[j] < need[i][j])

{

process = -1;

break;

}

}

}

if(process != -1)

break;

}

if(process != -1)

{

printf("\nProcess %d runs to completion!", process + 1);

safeSequence[count] = process + 1;

count++;

for(j = 0; j < r; j++)

{

avail[j] += alloc[process][j];

alloc[process][j] = 0;

Max[process][j] = 0;

completed[process] = 1;

}

}

}

while(count != p && process != -1);

if(count == p)

{

printf("\nThe system is in a safe state!!\n");

printf("Safe Sequence : < ");

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

printf("%d ", safeSequence[i]);

printf(">\n");

}

else

printf("\nThe system is in an unsafe state!!");

}

**OUTPUT**

Enter the no of processes : 5

Enter the no of resources : 3

Enter the Max Matrix for each process :

For process 1 : 7

5

3

For process 2 : 3

2

2

For process 3 : 7

0

2

For process 4 : 2

2

2

For process 5 : 4

3

3

Enter the allocation for each process :

For process 1 : 0

1

0

For process 2 : 2

0

0

For process 3 : 3

0

2

For process 4 : 2

1

1

For process 5 : 0

0

2

Enter the Available Resources : 3

3

2

Max matrix: Allocation matrix:

7 5 3 0 1 0

3 2 2 2 0 0

7 0 2 3 0 2

2 2 2 2 1 1

4 3 3 0 0 2

Process 2 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

7 0 2 3 0 2

2 2 2 2 1 1

4 3 3 0 0 2

Process 3 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

0 0 0 0 0 0

2 2 2 2 1 1

4 3 3 0 0 2

Process 4 runs to completion!

Max matrix: Allocation matrix:

7 5 3 0 1 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 3 3 0 0 2

Process 1 runs to completion!

Max matrix: Allocation matrix:

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

0 0 0 0 0 0

4 3 3 0 0 2

Process 5 runs to completion!

The system is in a safe state!!

Safe Sequence : < 2 3 4 1 5 >

**RESULT:**

Thus the program to implement bankers algorithm for deadlock avoidance was executed and verified successfully**.**

**EX.NO:9 IMPLEMENTATION OF DEADLOCK DETECTION ALGORITHM**

**AIM**

To write a program to implement deadlock detection algorithm.

**ALGORITHM**

1. Start the program.
2. Get the values of resources and processes.
3. Get the avail value.
4. After allocation find the need value.
5. Check whether its possible to allocate.
   1. If it is possible then the system is in safe state.
   2. Else system is not in safety state.
6. Stop the program.

**PROGRAM**

#include <stdio.h>;  
#include <conio.h>;  
void main()  
{  
 int found,flag,l,p[4][5],tp,tr,c[4][5],i,j,k=1,m[5],r[5],a[5],temp[5],sum=0;  
 clrscr();  
 printf("Enter total no of processes");  
 scanf("%d",&tp);  
 printf("Enter total no of resources");  
 scanf("%d",&tr);  
 printf("Enter claim (Max. Need) matrix\n");  
 for(i=1;i<=tp;i++)  
 {  
  printf("process %d:\n",i);  
  for(j=1;j<=tr;j++)  
  scanf("%d",&c[i][j]);  
 }  
 printf("Enter allocation matrix\n");  
 for(i=1;i<=tp;i++)  
 {  
  printf("process %d:\n",i);  
  for(j=1;j<=tr;j++)  
  scanf("%d",&p[i][j]);  
 }  
 printf("Enter resource vector (Total resources):\n");  
 for(i=1;i<=tr;i++)  
 {  
  scanf("%d",&r[i]);  
 }  
 printf("Enter availability vector (available resources):\n");  
 for(i=1;i<=tr;i++)  
 {  
  scanf("%d",&a[i]);  
  temp[i]=a[i];  
 }  
  
 for(i=1;i<=tp;i++)  
 {  
  sum=0;  
  for(j=1;j<=tr;j++)  
  {  
   sum+=p[i][j];  
  }  
  if(sum==0)  
  {  
   m[k]=i;  
   k++;  
  }  
 }  
 for(i=1;i<=tp;i++)  
 {  
  for(l=1;l<k;l++)  
  if(i!=m[l])  
  {  
   flag=1;  
   for(j=1;j<=tr;j++)  
   if(c[i][j]<temp[j])  
   {  
    flag=0;  
    break;  
   }  
  }  
  if(flag==1)  
  {  
   m[k]=i;  
   k++;  
   for(j=1;j<=tr;j++)  
   temp[j]+=p[i][j];  
  }  
 }  
 printf("deadlock causing processes are:");  
 for(j=1;j<=tp;j++)  
 {  
  found=0;  
  for(i=1;i<k;i++)  
  {  
   if(j==m[i])  
   found=1;  
  }  
  if(found==0)  
  printf("%d\t",j);  
 }  
 getch();  
}

**OUTPUT**  
Enter total no. of processes : 4  
Enter total no. of resources : 5  
Enter claim (Max. Need) matrix :  
0 1 0 0 1  
0 0 1 0 1  
0 0 0 0 1  
1 0 1 0 1  
Enter allocation matrix :  
1 0 1 1 0  
1 1 0 0 0  
0 0 0 1 0  
0 0 0 0 0  
Enter resource vector (Total resources) :  
2 1 1 2 1  
Enter availability vector (available resources) :  
0 0 0 0 1  
deadlock causing processes are : 2 3

**RESULT:**

Thus the program for implementing algorithm for deadlock detection was executed and verified successfully.

**EX.NO.10 IMPLEMENTATION OF THREADING AND SYNCHRONIZATION APPLICATION**

**AIM**

To write a program to implement threading concept.

**ALGORITHM**

1. Create a thread.
2. Find the id of the thread.
3. Check whether the id is equal to the process thread id.
4. If it is ,print first thread processing.
5. Otherwise print second thread processing.

**PROGRAM**

#include <unistd.h>

#include <sys/types.h>

#include <errno.h>

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <semaphore.h>

void handler ( void \*ptr );

pthread\_barrier\_t barrier;

int worker = 2; // number of workers

int job = 4; // number of jobs for each worker

int main()

{

int i = 0;

pthread\_t thread\_a;

pthread\_barrier\_init(&barrier, NULL, worker);

for (i; i < worker; i++)

{

int \*n\_workers = malloc(sizeof(\*n\_workers));

\*n\_workers = i;

pthread\_create (&thread\_a, NULL, (void \*) &handler, n\_workers);

}

pthread\_join(thread\_a, NULL);

pthread\_barrier\_destroy(&barrier);

pthread\_exit(0);

}

void handler ( void \*ptr )

{

int x = \*((int \*) ptr);

int i = 0;

for (i; i < job; i++)

{

printf("Worker %d: Doing Job %d\n", x, i);

pthread\_barrier\_wait(&barrier);

}

}

**OUTPUT**

cc thread3.c -lpthread

./a.out

main():creating thread 0

main():creating thread 1

main():creating thread 2

main():creating thread 3

main():creating thread 4

Hello World! Thread ID 0

Hello World! Thread ID 1

Hello World! Thread ID 2

Hello World! Thread ID 3

Hello World! Thread ID 4

**RESULT**

Thus the program for implementation of threads was executed successfully.

**EX:NO:11(A) MEMORY ALLOCATION METHODS FOR FIXED PARTITION**

**FIRST FIT**

**AIM**

To write a C program for first fit algorithm for memory management.

**ALGORITHM**

1. Input memory blocks with size and processes with size.
2. Initialize all memory blocks as free.
3. Start by picking each process and check if it can be assigned to current block.
4. If size-of-process <= size-of-block if yes then assign and check for next process.
5. If not then keep checking the further blocks.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp;

static int bf[max],ff[max];

clrscr();

printf("\n\tMemory Management Scheme - First Fit");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

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

{

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

{

if(bf[j]!=1)

{

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

if(temp>=0)

{

ff[i]=j;

break;

}

}

}

frag[i]=temp;

bf[ff[i]]=1;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

printf("\n%d\t\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             1                              5                             4

2                              4             3                              7                             3

**RESULT**

Thus the C program for first fit algorithm for memory management was executed and verified successfully.

**EX:NO:11(B) MEMORY ALLOCATION METHODS FOR FIXED PARTITION**

**WORST FIT**

**AIM**

To write a C program worst fit algorithm for memory management.

**ALGORITHM**

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the maximum block size that can be assigned to current process i.e., find max(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,highest=0;

static int bf[max],ff[max];

clrscr();

printf("\n\tMemory Management Scheme - Worst Fit");

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

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

{

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

{

if(bf[j]!=1) //if bf[j] is not allocated

{

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

if(temp>=0)

if(highest<temp)

{

ff[i]=j;

highest=temp;

}

}

}

frag[i]=highest;

bf[ff[i]]=1;

highest=0;

}

printf("\nFile\_no:\tFile\_size :\tBlock\_no:\tBlock\_size:\tFragement");

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

printf("\n%d\t\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                             3                              7                              6

2                              4                             1                              5                              1

**RESULT**

Thus the C program for worst fit algorithm for memory management was executed and verified successfully.

**EX:NO:11(C) MEMORY ALLOCATION METHODS FOR FIXED PARTITION**

**BEST FIT**

**AIM**

To write a C program best fit algorithm for memory management.

**ALGORITHM**

1. Input memory blocks and processes with sizes.
2. Initialize all memory blocks as free.
3. Start by picking each process and find the minimum block size that can be assigned to current process i.e., find min(bockSize[1], blockSize[2],.....blockSize[n]) > processSize[current], if found then assign it to the current process.
4. If not then leave that process and keep checking the further processes.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

#define max 25

void main()

{

int frag[max],b[max],f[max],i,j,nb,nf,temp,lowest=10000;

static int bf[max],ff[max];

clrscr();

printf("\nEnter the number of blocks:");

scanf("%d",&nb);

printf("Enter the number of files:");

scanf("%d",&nf);

printf("\nEnter the size of the blocks:-\n");

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

{

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

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

}

printf("Enter the size of the files :-\n");

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

{

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

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

}

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

{

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

{

if(bf[j]!=1)

{

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

if(temp>=0)

if(lowest>temp)

{

ff[i]=j;

lowest=temp;

}

}

}

frag[i]=lowest;

bf[ff[i]]=1;

lowest=10000;

}

printf("\nFile No\tFile Size \tBlock No\tBlock Size\tFragment");

for(i=1;i<=nf && ff[i]!=0;i++)

printf("\n%d\t\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

**RESULT**

Thus the C program for best fit algorithm for memory management was executed and verified successfully.

**EX:NO:12 IMPLEMENTATION OF PAGING TECHNIQUE OF MEMORY**

**MANAGEMENT**

**AIM**

To write a C program to implement the paging technique of memory management.

**ALGORITHM**

* 1. First when I enter physical address and page size according to the page number I can enter my jobs.
  2. Second Each job can fit in the number of pages according to the job size.
  3. Third Each time I enter job the Memory Block Table (MBT) should reload and tells how much memory available or occupied.
  4. Finally if there is not enough space to put the larger job it gives Error.

**PROGRAM**

#include<stdio.h>  
void main()  
{  
int memsize=15;  
int pagesize,nofpage;  
int p[100];  
int frameno,offset;  
int logadd,phyadd;  
int i;  
int choice=0;  
printf("\nYour memsize is %d ",memsize);  
printf("\nEnter page size:");  
scanf("%d",&pagesize);  
nofpage=memsize/pagesize;  
for(i=0;i<nofpage;i++)  
{  
printf("\nEnter the frame of page%d:",i+1);  
scanf("%d",&p[i]);  
}  
do  
{  
printf("\nEnter a logical address:");  
scanf("%d",&logadd);  
frameno=logadd/pagesize;  
offset=logadd%pagesize;  
phyadd=(p[frameno]\*pagesize)+offset;  
printf("\nPhysical address is:%d",phyadd);  
printf("\nDo you want to continue(1/0)?:");  
scanf("%d",&choice);  
}while(choice==1);  
}

**OUTPUT**

Your memsize is 15

Enter page size:5

Enter the frame of page1:2

Enter the frame of page2:4

Enter the frame of page3:7

Enter a logical address:3

Physical address is:13

Do you want to continue(1/0)?:1

Enter a logical address:1

Physical address is:11

Do you want to continue(1/0)?:0

**RESULT**

Thus the C program to implement the paging technique of memory management was executed and verified successfully.

**EX.NO:13(A) FIFO PAGE REPLACEMENT ALGORITHM**

**AIM**

To write a c program to implement FIFO(First In First Out) page replacement algorithm.

**ALGORITHM**

1. Start the process

2. Declare the size with respect to page length

3. Check the need of replacement from the page to memory

4. Check the need of replacement from old page to new page in memory

5. Forma queue to hold all pages

6. Insert the page require memory into the queue

7. Check for bad replacement and page fault

8. Get the number of processes to be inserted

9. Display the values

10. Stop the process

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;

void main()

{

clrscr();

printf("\n \t\t\t FIFI PAGE REPLACEMENT ALGORITHM");

printf("\n Enter no.of frames....");

scanf("%d",&nof);

printf("Enter number of reference string..\n");

scanf("%d",&nor);

printf("\n Enter the reference string..");

for(i=0;i<nor;i++)

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

printf("\nThe given reference string:");

for(i=0;i<nor;i++)

printf("%4d",ref[i]);

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

frm[i]=-1;

printf("\n");

for(i=0;i<nor;i++)

{

flag=0;

printf("\n\t Reference np%d->\t",ref[i]);

for(j=0;j<nof;j++)

{

if(frm[j]==ref[i])

{

flag=1;

break;

}

}

if(flag==0)

{

pf++;

victim++;

victim=victim%nof;

frm[victim]=ref[i];

for(j=0;j<nof;j++)

printf("%4d",frm[j]);

}

}

printf("\n\n\t\t No.of pages faults...%d",pf);

getch();

}

**OUTPUT**

FIFO PAGE REPLACEMENT ALGORITHM

Enter no.of frames....4

Enter number of reference string..

6

Enter the reference string..

5 6 4 1 2 3

The given reference string:

...................................... 5 6 4 1 2 3

Reference np5-> 5 -1 -1 -1

Reference np6-> 5 6 -1 -1

Reference np4-> 5 6 4 -1

Reference np1-> 5 6 4 1

Reference np2-> 2 6 4 1

Reference np3-> 2 3 4 1

No.of pages faults...6

**RESULT**

Thus the C program implement FIFO(First In First Out) page replacement algorithm was executed and verified successfully.

|  |  |
| --- | --- |
| **EX.NO:13(B)** | **LRU PAGE REPLACEMENT ALGORITHM** |

**AIM**

To write a c program to implement LRU (Least Recently Used) 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<stdio.h>

#include<conio.h>

int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;

int recent[10],lrucal[50],count=0;

int lruvictim();

void main()

{

clrscr();

printf("\n\t\t\t LRU PAGE REPLACEMENT ALGORITHM");

printf("\n Enter no.of Frames....");

scanf("%d",&nof);

printf(" Enter no.of reference string..");

scanf("%d",&nor);

printf("\n Enter reference string..");

for(i=0;i<nor;i++)

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

printf("\n\n\t\t LRU PAGE REPLACEMENT ALGORITHM ");

printf("\n\t The given reference string:");

printf("\n………………………………..");

for(i=0;i<nor;i++)

printf("%4d",ref[i]);

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

{

frm[i]=-1;

lrucal[i]=0;

}

for(i=0;i<10;i++)

recent[i]=0;

printf("\n");

for(i=0;i<nor;i++)

{

flag=0;

printf("\n\t Reference NO %d->\t",ref[i]);

for(j=0;j<nof;j++)

{

if(frm[j]==ref[i])

{

flag=1;

break;

}

}

if(flag==0)

{

count++;

if(count<=nof)

victim++;

else

victim=lruvictim();

pf++;

frm[victim]=ref[i];

for(j=0;j<nof;j++)

printf("%4d",frm[j]);

}

recent[ref[i]]=i;

}

printf("\n\n\t No.of page faults...%d",pf);

getch();

}

int lruvictim()

{

int i,j,temp1,temp2;

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

{

temp1=frm[i];

lrucal[i]=recent[temp1];

}

temp2=lrucal[0];

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

{

if(temp2>lrucal[j])

temp2=lrucal[j];

}

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

if(ref[temp2]==frm[i])

return i;

return 0;

}

**OUTPUT**

LRU PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3

Enter no.of reference string..6

Enter reference string..6 5 4 2 3 1

LRU PAGE REPLACEMENT ALGORITHM

The given reference string:

…………………. 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4

Reference NO 2-> 2 5 4

Reference NO 3-> 2 3 4

Reference NO 1-> 2 3 1

No.of page faults...6

**RESULT**

Thus the C program implement LRU (Least Recently Used) page replacement algorithm was executed and verified successfully.

**EX.NO:13(C) OPTIMAL (LFU) PAGE REPLACEMENT ALGORITHM**

**AIM**

To write a c program to implement LFU page replacement algorithm

**ALGORITHM**

1. Start the process

2. Declare the size

3. Get the number of pages to be inserted

4. Get the value

5. Declare counter and stack

6. Select the least frequently used page by counter value

7. Stack them according the selection.

8. Display the values

9. Stop the process

**PROGRAM**

#include<stdio.h>

#include<conio.h>

int i,j,nof,nor,flag=0,ref[50],frm[50],pf=0,victim=-1;

int recent[10],optcal[50],count=0;

int optvictim();

void main()

{

clrscr();

printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHN");

printf("\n.................................");

printf("\nEnter the no.of frames");

scanf("%d",&nof);

printf("Enter the no.of reference string");

scanf("%d",&nor);

printf("Enter the reference string");

for(i=0;i<nor;i++)

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

clrscr();

printf("\n OPTIMAL PAGE REPLACEMENT ALGORITHM");

printf("\n................................");

printf("\nThe given string");

printf("\n....................\n");

for(i=0;i<nor;i++)

printf("%4d",ref[i]);

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

{

frm[i]=-1;

optcal[i]=0;

}

for(i=0;i<10;i++)

recent[i]=0;

printf("\n");

for(i=0;i<nor;i++)

{

flag=0;

printf("\n\tref no %d ->\t",ref[i]);

for(j=0;j<nof;j++)

{

if(frm[j]==ref[i])

{

flag=1;

break;

}

}

if(flag==0)

{

count++;

if(count<=nof)

victim++;

else

victim=optvictim(i);

pf++;

frm[victim]=ref[i];

for(j=0;j<nof;j++)

printf("%4d",frm[j]);

}

}

printf("\n Number of page faults: %d",pf);

getch();

}

int optvictim(int index)

{

int i,j,temp,notfound;

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

{

notfound=1;

for(j=index;j<nor;j++)

if(frm[i]==ref[j])

{

notfound=0;

optcal[i]=j;

break;

}

if(notfound==1)

return i;

}

temp=optcal[0];

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

if(temp<optcal[i])

temp=optcal[i];

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

if(frm[temp]==frm[i])

return i;

return 0;

}

**OUTPUT:**

OPTIMAL PAGE REPLACEMENT ALGORITHM

Enter no.of Frames....3

Enter no.of reference string..6

Enter reference string..6 5 4 2 3 1

OPTIMAL PAGE REPLACEMENT ALGORITHM

The given reference string:

…………………. 6 5 4 2 3 1

Reference NO 6-> 6 -1 -1

Reference NO 5-> 6 5 -1

Reference NO 4-> 6 5 4

Reference NO 2-> 2 5 4

Reference NO 3-> 2 3 4

Reference NO 1-> 2 3 1

No.of page faults...6

**RESULT:**

Thus the C program implement LFU page replacement algorithm was executed and verified successfully.

**EX.No:14(A) SINGLE LEVEL DIRECTORY USING FILE ORGANIZATION**

**TECHNIQUES**

**AIM**

To implement Single level directory structure in C.

**ALGORITHM**

1. Start the program

2. Declare the number, names and size of the directories and file names.

3. Get the values for the declared variables.

4. Display the files that are available in the directories.

5. Stop the program.

**PROGRAM**

#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir;

void main()

{

int i,ch;

char f[30];

clrscr();

dir.fcnt = 0;

printf("\nEnter name of directory -- ");

scanf("%s", dir.dname);

while(1)

{

printf("\n\n 1. Create File\t2. Delete File\t3. Search File \n 4. Display Files\t5. Exit\nEnter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter the name of the file -- ");

scanf("%s",dir.fname[dir.fcnt]);

dir.fcnt++;

break;

case 2: printf("\n Enter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is deleted ",f);

strcpy(dir.fname[i],dir.fname[dir.fcnt-1]);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

else

dir.fcnt--;

break;

case 3: printf("\n Enter the name of the file -- ");

scanf("%s",f);

for(i=0;i<dir.fcnt;i++)

{

if(strcmp(f, dir.fname[i])==0)

{

printf("File %s is found ", f);

break;

}

}

if(i==dir.fcnt)

printf("File %s not found",f);

break;

case 4: if(dir.fcnt==0)

printf("\n Directory Empty");

else

{

printf("\n The Files are -- ");

for(i=0;i<dir.fcnt;i++)

printf("\t%s",dir.fname[i]);

}

break;

default: exit(0);

}

}

getch();

}

**OUTPUT**

Enter name of directory -- CSE

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- A

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- B

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 1

Enter the name of the file -- C

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 4

The Files are -- A B C

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 3

Enter the name of the file – ABC

File ABC not found

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 2

Enter the name of the file – B

File B is deleted

1. Create File 2. Delete File 3. Search File

4. Display Files 5. Exit Enter your choice – 5

**RESULT**

Thus the program to implement single level directory was executed successfully.

**Ex.No:14(B) TWO LEVEL DIRECTORY USING FILE ORGANIZATION**

**TECHNIQUES**

**AIM**

To implement Two-level directory structure in C.

**ALGORITHM**

1. Start the program

2. Declare the number, names and size of the directories and subdirectories and file names.

3. Get the values for the declared variables.

4. Display the files that are available in the directories and subdirectories.

5. Stop the program.

**PROGRAM**

#include<stdio.h>

struct

{

char dname[10],fname[10][10];

int fcnt;

}dir[10];

void main()

{

int i,ch,dcnt,k;

char f[30], d[30];

clrscr();

dcnt=0;

while(1)

{

printf("\n\n 1. Create Directory\t 2. Create File\t 3. Delete File");

printf("\n 4. Search File \t \t 5. Display \t 6. Exit \t Enter your choice -- ");

scanf("%d",&ch);

switch(ch)

{

case 1: printf("\n Enter name of directory -- ");

scanf("%s", dir[dcnt].dname);

dir[dcnt].fcnt=0;

dcnt++;

printf("Directory created");

break;

case 2: printf("\n Enter name of the directory -- ");

scanf("%s",d);

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

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",dir[i].fname[dir[i].fcnt]);

dir[i].fcnt++;

printf("File created");

break;

}

if(i==dcnt)

printf("Directory %s not found",d);

break;

case 3: printf("\nEnter name of the directory -- ");

scanf("%s",d);

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

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is deleted ",f);

dir[i].fcnt--;

strcpy(dir[i].fname[k],dir[i].fname[dir[i].fcnt]);

goto jmp;

}

}

printf("File %s not found",f);

goto jmp;

}

}

printf("Directory %s not found",d);

jmp : break;

case 4: printf("\nEnter name of the directory -- ");

scanf("%s",d);

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

{

if(strcmp(d,dir[i].dname)==0)

{

printf("Enter the name of the file -- ");

scanf("%s",f);

for(k=0;k<dir[i].fcnt;k++)

{

if(strcmp(f, dir[i].fname[k])==0)

{

printf("File %s is found ",f);

goto jmp1;

}

}

printf("File %s not found",f);

goto jmp1;

}

}

printf("Directory %s not found",d);

jmp1: break;

case 5: if(dcnt==0)

printf("\nNo Directory's ");

else

{

printf("\nDirectory\tFiles");

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

{

printf("\n%s\t\t",dir[i].dname);

for(k=0;k<dir[i].fcnt;k++)

printf("\t%s",dir[i].fname[k]);

}

}

break;

default:exit(0);

}

}

getch();

}

**OUTPUT**

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR1

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 1

Enter name of directory -- DIR2

Directory created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1

Enter name of the file -- A1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR1

Enter name of the file -- A2

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 2

Enter name of the directory – DIR2

Enter name of the file -- B1

File created

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 5

Directory Files

DIR1 A1 A2

DIR2 B1

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 4

Enter name of the directory – DIR

Directory not found

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice -- 3

Enter name of the directory – DIR1

Enter name of the file -- A2

File A2 is deleted

1. Create Directory 2. Create File 3. Delete File

4. Search File 5. Display 6. Exit Enter your choice – 6

**RESULT**

Thus the program to implement two level directory was executed successfully**.**

|  |  |
| --- | --- |
| **EX.NO:15(A)** | **SEQUENTIAL FILE ALLOCATION** |

**AIM**

Write a C Program to implement Sequential File Allocation method.

**ALGORITHM**

Step 1: Start the program.

Step 2: Get the number of memory partition and their sizes.

Step 3: Get the number of processes and values of block size for each process.

Step 4: First fit algorithm searches the entire entire memory block until a hole which is

big enough is encountered. It allocates that memory block for the requesting

process.

Step 5: Best-fit algorithm searches the memory blocks for the smallest hole which can

be allocated to requesting process and allocates if.

Step 6: Worst fit algorithm searches the memory blocks for the largest hole and

allocates it to the process.

Step 7: Analyses all the three memory management techniques and display the best

algorithm which utilizes the memory resources effectively and efficiently.

Step 8: Stop the program.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

main()

{

int n,i,j,b[20],sb[20],t[20],x,c[20][20];

clrscr();

printf("Enter no.of files:");

scanf("%d",&n);

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

{

printf("Enter no. of blocks occupied by file%d",i+1);

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

printf("Enter the starting block of file%d",i+1);

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

t[i]=sb[i];

for(j=0;j<b[i];j++)

c[i][j]=sb[i]++;

}

printf("Filename\tStart block\tlength\n");

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

printf("%d\t %d \t%d\n",i+1,t[i],b[i]);

printf("Enter file name:");

scanf("%d",&x);

printf("File name is:%d",x);

printf("length is:%d",b[x-1]);

printf("blocks occupied:");

for(i=0;i<b[x-1];i++)

printf("%4d",c[x-1][i]);

getch();

}

**OUTPUT**

Enter no.of files: 2

Enter no. of blocks occupied by file1 4

Enter the starting block of file1 2

Enter no. of blocks occupied by file2 10

Enter the starting block of file2 5

Filename Start block length

1 2 4

2 5 10

Enter file name: rajesh

File name is:12803 length is:0blocks occupied.

**RESULT**

Thus the program for Sequential File Allocation method was executed and verified successfully.

|  |  |
| --- | --- |
| **EX.NO:15(B)** | **INDEXED FILE ALLOCATION** |

**AIM**

Write a C Program to implement Indexed File Allocation method.

**ALGORITHM**

Step 1: Start.

Step 2: Let n be the size of the buffer

Step 3: check if there are any producer

Step 4: if yes check whether the buffer is full

Step 5: If no the producer item is stored in the buffer

Step 6: If the buffer is full the producer has to wait

Step 7: Check there is any cosumer. If yes check whether the buffer is empty

Step 8: If no the consumer consumes them from the buffer

Step 9: If the buffer is empty, the consumer has to wait.

Step 10: Repeat checking for the producer and consumer till required

Step 11: Terminate the process.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

main()

{

int n,m[20],i,j,sb[20],s[20],b[20][20],x;

clrscr();

printf("Enter no. of files:");

scanf("%d",&n);

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

{ printf("Enter starting block and size of file%d:",i+1);

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

printf("Enter blocks occupied by file%d:",i+1);

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

printf("enter blocks of file%d:",i+1);

for(j=0;j<m[i];j++)

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

} printf("\nFile\t index\tlength\n");

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

{ printf("%d\t%d\t%d\n",i+1,sb[i],m[i]);

}printf("\nEnter file name:");

scanf("%d",&x);

printf("file name is:%d\n",x);

i=x-1;

printf("Index is:%d",sb[i]);

printf("Block occupied are:");

for(j=0;j<m[i];j++)

printf("%3d",b[i][j]);

getch();}

**OUTPUT**

Enter no. of files:2

Enter starting block and size of file1: 2 5

Enter blocks occupied by file1:10

enter blocks of file1:3

2 5 4 6 7 2 6 4 7

Enter starting block and size of file2: 3 4

Enter blocks occupied by file2:5

enter blocks of file2: 2 3 4 5 6 File index length

1 2 10

2 3 5

Enter file name: venkat

file name is:12803

Index is:0

**RESULT**

Thus the program for Indexed File Allocation method was executed and verified successfully.

|  |  |
| --- | --- |
| **EX.N0:15(C)** | **LINKED FILE ALLOCATION** |

**AIM**

Write a C Program to implement Linked File Allocation method.

**ALGORITHM**

Step 1: Create a queue to hold all pages in memory

Step 2: When the page is required replace the page at the head of the queue

Step 3: Now the new page is inserted at the tail of the queue

Step 4: Create a stack

Step 5: When the page fault occurs replace page present at the bottom of the stack

Step 6: Stop the allocation.

**PROGRAM**

#include<stdio.h>

#include<conio.h>

struct file

{

char fname[10];

int start,size,block[10];

}f[10];

main()

{

int i,j,n;

clrscr();

printf("Enter no. of files:");

scanf("%d",&n);

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

{

printf("Enter file name:");

scanf("%s",&f[i].fname);

printf("Enter starting block:");

scanf("%d",&f[i].start);

f[i].block[0]=f[i].start;

printf("Enter no.of blocks:");

scanf("%d",&f[i].size);

printf("Enter block numbers:");

for(j=1;j<=f[i].size;j++)

{

scanf("%d",&f[i].block[j]);

}

}

printf("File\tstart\tsize\tblock\n");

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

{

printf("%s\t%d\t%d\t",f[i].fname,f[i].start,f[i].size);

for(j=1;j<=f[i].size-1;j++)

printf("%d--->",f[i].block[j]);

printf("%d",f[i].block[j]);

printf("\n");

}

getch();

}

**OUTPUT**

Enter no. of files:2

Enter file name:venkat

Enter starting block:20

Enter no.of blocks:6

Enter block numbers: 4

12

15

45

32

25

Enter file name:rajesh

Enter starting block:12

Enter no.of blocks:5

Enter block numbers:6

5

4

3

2

File start size block

venkat 20 6 4--->12--->15--->45--->32--->25

rajesh 12 5 6--->5--->4--->3--->2

**RESULT**

Thus the program for Linked File Allocation method was executed and verified successfully