

DEPARTMENT VISION

Nurturing globally competent Computer Science graduates capable of taking challenges in the Industry and Research & Development activities.

DEPARTMENT MISSION

M1: Imparting quality education to meet the needs of industry, and to achieve excellence in teaching and learning.

M2: Inculcating value-based, socially committed professionalism for development of society.

M3: Providing support to promote quality research.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

PEO1: Graduate will be successfully employed in industry or academia

PEO2: Graduate shall be able to develop innovative computing solutions

PROGRAM OUTCOME (PO) & PROGRAM SPECIFIC OUTCOMES (PSO)

S No	PO
1	Engineering Knowledge
2	Problem Analysis
3	Design / Development of Solutions
4	Conduct Investigations of Complex Problems
5	Modern Tool Usage
6	The Engineer and society
7	Environment and Sustainability
8	Ethics
9	Individual and Team Work
10	Communication
11	Project Management and Finance
12	Life Long Learning
S No	PSO
1	Proficiency in Core Areas
2	Software Project Management

COURSE OUTCOMES

Student will be able to:

Co No.	Course Outcome	Knowledge Level
CO1	Familiarize Basic Linux commands and systems calls in Operating Systems.	Apply(K3)
CO2	Implement Process Creation and Inter Process Communication in Operating Systems	Apply(K3)
CO3	Implement First Come First Served, Shortest Job First, Round Robin and Priority based CPU Scheduling Algorithms.	Apply(K3)
CO4	Illustrate the performance of First In First Out, Least Recently Used and Least Frequently Used Page Replacement Algorithms.	Apply(K3)
CO5	Implement modules for Deadlock Detection and Deadlock Avoidance in Operating Systems.	Apply(K3)
CO6	Implement modules for Storage Management and Disk Scheduling in Operating Systems.	Apply(K3)

TABLE OF CONTENTS

SI No:	Contents	Course Outcome	Page No:
1	Basic Linux commands	CO1	01
2	Shell programming -Command syntax -Write simple functions with basic tests, loops, patterns	CO1	08
3	Write a program to create a process in Linux.	CO1	15
4	Write programs using the following system calls of Linux operating system: fork, exec, getpid, exit, wait, close, stat, opendir, readdir	CO1	17
5	Write programs using the I/O system calls of Linux operating system (open, read, write)	CO1	21
6	Given the list of processes, their CPU burst times and arrival times, display/print the Gantt chart for FCFS and SJF. For each of the scheduling policies, compute and print the average waiting time and average turnaround time	CO3	23
7	Write a C program to simulate following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time. a)FCFS b) SJF c) Round Robin (pre-emptive) d) Priority	CO3	33
8	Write a C program to simulate following contiguous memory allocation techniques. a) Worst-fit b) Best-fit c) First-fit	CO6	45
9	Write a C program to simulate paging technique of memory management.	CO6	53
10	Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.	CO5	56
11	Write a C program to simulate disk scheduling algorithms a)FCFS b) SCAN c) C-SCAN	CO6	63

12	Write a C program to simulate page replacement algorithms a) FIFO, b) LRU, c) LFU	CO4	71
13	Implement programs for Inter Process Communication using Shared Memory .	CO2	78
14	Implement Semaphores.	CO2	81
15	Write a C program to simulate producer-consumer problem using semaphores.	CO2	84
16	Write a program for file manipulation for display a file and directory in memory.	CO6	90
17	Write a C program to simulate following file allocation strategies. a)Sequential b) Indexed c) Linked	CO6	93
18	Course project	CO2	101

Student Assessment Sheet

Exp No.	Algorithm (10)	Program (10)	Output (30)	Total (30)	Viva (15)	Total (45)
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						

Experiment No : 01

Date : 29 April 2021

Basics Of Unix Commands

Aim :

To study about the basis of UNIX and basic UNIX commands

UNIX :

It is a multi-user operating system developed at AT & T Bell Industries, USA in 1969. Ken Thomson along with Dennis Ritchie developed it from MULTICS OS. By 1980, UNIX had been completely rewritten using C language.

LINUX :

It is similar to UNIX which is created by Linus Torvalds. All UNIX commands work in Linux. Linux is a open source software. The main feature of Linux is coexisting with other OS such as windows and UNIX.

Structure of a Linux System :

It consists of three parts.

- a) UNIX kernel

b) Shells

c) Tools and applications

UNIX Kernel :

Kernel is the core of the UNIX OS. It controls all tasks, schedule all processes and carries out all functions of OS.

Decides when one program stops and another starts

Shell :

Shell is the command interpreter in the UNIX OS. It accepts command from the user and analyses and interprets them.

→ Contents :

a) date - used to check the date and time

Syn : \$ date

Format	Purpose	Example	Result
+ %m	To display only month	\$ date + %m	06
+ %b	To display month name	\$ date + %b	Juru
+ %d	To display day of the month	\$ date + %d	01
+ %y	To display last two digits of year	\$ date + %y	09
+ %H	To display hours	\$ date + %H	10
+ %M	To display minutes	\$ date + %M	45
+ %S	To display seconds	\$ date + %S	55

b) cal - used to display the calendar

Syn : \$ cal 2 2009

d) ls - Used to list the files

Syn : \$ ls

ls -s : All files (including files with prefix)

ls -l : Detailed (Provides file statistics)

ls -t : Order by creation time

ls -u : Sort by access time

ls -S : Order by size

ls -r : Reverse order

ls -d : Directories with /, executable with *, symbolic links with @, local sockets with =

ls -s : Show file size

ls -h : Show file size in kilo bytes & Mega bytes.

ls [a-m]* : List all the files whose name begin with alphabets from "a" to "m".

ls [a]* : List all files whose name begins with "a" or "A".

e) lp - Used to take printouts

Syn : \$ lp filename

f) man - used to provide manual help on every UNIX commands.

Syn : \$ man UNIX command

eg : \$ man cat

g) who & whoami : It displays data about all users who have logged into the system currently. The next command displays the current user only.

Syn : \$who
\$whoami

h) uptime : tells you how long the computer has been running since its last reboot or power-off

Syn : \$uptime

i) uname : It displays the system information such as hardware platform, system name and processor, OS type

Syn : \$uptime

j) hostname - display and set system host name

Syn : \$hostname

k) bc - stands for best calculator

Syn : \$bc
10/2 * 3

15

\$bc -1

scale=2

s(3.14)

0

\$bc
scale=1
2.25 +1

3.35

quit

\$bc
scale(196)

11

quit

\$bc

for (i=1; i<3; i=i+1)

1

2

3

quit

File manipulation commands:

a) cat - this create, view and concatenate files

Syn : \$ cat > filename → creation

\$ cat filename → viewing

\$ cat >> filename → adding some text

\$ cat file1 file2 > file3 → concatenate

\$ cat file1 file2 >> file3 → No over writing.

b) grep - Used to search a particular word or pattern related to ~~exist~~ that word from the file.

Syn : \$ grep search word filename

Eg : \$ grep any student

c) rm - delete a file from the file system

Syn : \$ rm filename

d) touch - used to create a blank space file

Syn : \$ touch filename

e) cp - copies the file or directories

Syn : \$ cp source file destination file

f) mv - to rename the file or directory

Syn : \$ mv old file new file

g) cut - it cuts or pickup a given number of characters or fields of the file

Syn : \$ cut <option> <filename>

Eg : \$ cut -c filename

\$ cut -c 1-10 emp -c : cutting column

\$ cut -f 3,6 emp -f : cutting field

b) head - displays 10 lines from the head (top) of a given file

Syn : \$ head filename

Eg : \$head student

\$ head -2 student : To display top 2 lines

i) tail - To display 10 lines of a file

Syn : \$ tail filename

j) chmod - used to change the permission of a file or directory

Syn : \$ chmod category operation permission file

where, Category - It is the user type

Operation - It is used to remove or assign permission

Permission - It is the type of permission

File - Used to assign or remove permission all

Eg : \$ chmod u-wx student

Removes write and execute permission for users

\$ chmod g=rwx student

Assigns absolute permission for group of all
read, write and execute permissions.

k) wc - it counts the number of lines, words, characters in a specific file (s) with the options as
-l, -w, -c

Category	Operation	Permission
U - users	+ assign	r - read
g - groups	- remove	w - write
O - other	= assign absolutely	x - execute

Exn : \$ wc -l filename

\$ wc -w filename

\$ wc -c filename

Result :

Understood basic of UNIX and basic UNIX commands

Experiment No : 02

Date : 29 April 2021

Simple Shell Programs

Aim :

To write a simple shell program by using conditional, branching and looping statements

Shell Variables

The first useful command to know about in building a shell program is "echo", which allows you to perform output from your shell program :

`echo "That is a test!"`

This sends the string "This is a test!" to standard output. It is recommended that your shell programs generate some output to inform the user of what they are doing.

The shell allows you to store values in variables. All you have to do to declare a variable is assign a value to it.

`stvar = "This is a test!"`

The string is enclosed in double quotes to ensure that the variable allows the entire string and there are no spaces around the "=". The value of the shell variable can be obtained by preceding it with a "\$":

`echo $ strvar`

This displays "This is a test!".

Decision Making and Loop constructs

The flow of control with sh script is done using four main constructs: if....then....elif...else, do....while, for and case.

1. Do....while

The Do....while takes the following generic form:

while condition
do list
done

2. For

The syntax of the for command is:
for variable in word...

do list
done

3. Case

The case construct has the following syntax:

```
case word in
  pattern) list ;;
  .....
esac
```

I write a shell program to check the given number is odd or even ?

Program :

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

Result :

Enter a number

6

6 is even number.

II Write a shell program to find the factorial of a number using for loop.

Program :

```
echo "Enter a number"
```

```

read n
fact=1
for (( i=1 ; i<=n ; i++ ))
do
fact=$[ $fact * $i ]
done
echo "Factorial is $fact"

```

Result :

Enter a number

5

Factorial is 120.

III Write a shell program to find the sum of first n numbers using while loop

Program :

```

echo "Enter a number"
read num
sum=0
i=1
while test $i -le $num
do
sum=$(( $sum + $i ))
i=$(( $i + 1 )) u post
done
echo "sum of first $num numbers is $sum"

```

Result :

Enter a number

10

sum of first 10 numbers is 55

IV Write a shell programs to swap two integers

Program :

```
echo " Enter two numbers A and B"
read a
read b
echo " Before swapping A = $a and B = $b"
temp = $a
a = $b
b = $temp
echo " After swapping A = $a and B = $b"
```

Result :

Enter two numbers as A and B

50

60

Before swapping A = 50 and B = 60

After swapping A = 60 and B = 50

V To write a shell program to implement calculator.

Program :

res = 0

i = "y"

while [\$i = "y"]

do

echo " Enter first and second number"

read a

```

read b
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
echo "Enter your choice."
read opt
case $opt in
    1) res=$(( $a + $b ))
       echo "Sum is $res" ;;
    2) res=$(( $a - $b ))
       echo "Result is $res" ;;
    3) res=$(( $a * $b ))
       echo "Product is $res" ;;
    4) res=$(( $a / $b ))
       echo "Result is $res" ;;
*) echo "Invalid choice" ;;
esac
echo "Do you want to continue (y/n)"
read i
if [ $i != y ]
then
    exit
fi
done.

```

Result :

Enter first and second number

10

20

1. Addition
2. Subtraction
3. Multiplication
4. Division

Enter your choice

3

Product is 200

Do you want to continue (y/n)

y

Enter first and second number

33

22

1. Addition
2. Subtraction
3. Multiplication
4. Division

Enter your choice

1

Sum is 55

Do you want to continue (y/n)

n

Experiment No : 03

Date : 06 May 2021

Process Creation In Linux

Aim :

To write a C program to create a process in Linux

Algorithm :

1. Start
2. Declare variable child-pid as a pid_t datatype
3. Initialize pid_t to fork() system call which creates a child process.
4. If (child-pid < 0)
 - 4.1 Return "fork failed"
5. Else if (child-pid == 0)
 - 5.1 Print "child process created"
 - 5.2 Display process id of child and parent processes using getpid() & getppid()
6. Else
 - 6.1 Display "parent process created"
 - 6.2 Display process id of child and parent processes using getpid() and getppid()
- 7 Stop

Program :

```
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main( )
{
    pid_t child_pid;
    child_pid = fork(); // Create a new child process;
    if (child_pid < 0)
    {
        printf("fork failed"); return 1;
    }
    else if (child_pid == 0)
    {
        printf ("\nchild process successfully created!\n");
        printf ("child_PID = %d, parent_PID = %d\n", getpid(), getppid());
    }
    else
    {
        printf ("\nparent process successfully created!\n");
        printf ("child_PID = %d, parent_PID = %d", getpid(), getppid());
    }
    return 0;
}
```

Output :

```
Parent process successfully created!

Child process successfully created!
Child_PID = 2653, Parent_PID = 2649
Child_PID = 2649, Parent_PID = 2648

...Program finished with exit code 0
Press ENTER to exit console.
```

Result :

The program was compiled and executed successfully.

Experiment No : 04

Date : 20 May 2021

Linux System Calls

Aim :

To write a C program using the following system calls of UNIX operating system fork, exec, getpid, exit, wait, close, stat, opendir, readdir.

1. Program using stat, opendir, and readdir commands.

Algorithm :

1. Start
2. Create struct dirent
3. Declare the variables buff and pointer dptr
4. Get the directory name
5. Open the directory
6. Read the contents in directory and print it
7. Close the directory.
8. Stop

Program :

```
#include<stdio.h>
#include<dirent.h>
#include<stdlib.h>
#include <sys/stat.h>
struct dirent *dptr;
```

```

int main(int argc, char *argv[])
{
    char buff[100];
    struct stat stats;
    DIR *dirp;
    printf("\n\n ENTER DIRECTORY NAME : ");
    scanf("%s", buff);
    if((dirp=open(dirp,buff))==NULL || stat(buff, &stats))
    {
        printf("The given directory does not exist");
        exit(1);
    }
    while(dptr=readdir(dirp))
    {
        printf("%s\n",dptr->d_name);
    }
    closedir(dirp);
}

```

Output :

```

Enter directory name : Lab
The given directory does not exist

...Program finished with exit code 0
Press ENTER to exit console.

```

Result :

Program compiled and executed successfully.

2. Program for system call of UNIX operating system using fork, getpid, exit, and wait.

Algorithm :

1. Start
2. Declare the variables pid, pid1, pid2 .

3. Call fork() system call to create process
4. If pid == -1
 - 4.1 Exit
5. If pid != -1
 - 4.2 Get the process id from getpid()
6. Print the process id
7. Stop.

Program :

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
void main()
{
    int pid,pid1,pid2, c_pid; pid=fork();
    if(pid<0)
    {
        printf("ERROR IN PROCESS CREATION \n");
        exit(1);
    }
    if(pid!=0)
    {
        c_pid = wait(NULL);
        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 :

```
The parent process ID is 4614  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Result :

Program compiled and executed successfully.

Experiment No : 05

Date : 20 May 2021

I / O System Calls Of Linux

Aim :

To write a C program using the I/O system calls of linux operating system (open, read, write)

Algorithm :

1. Start
2. Create an Integer fd
3. Declare a character as buf [80]
4. Open a file by initializing fd = open ("myfile.txt", O_CREAT | O_WRONLY, 0600)
5. if fd = -1
 5.1 Print error message and exit.
6. write to the file, write (fd, "Hello world \n", 13)
7. Close fd
8. Open myfile, fd = open ("myfile.txt", O_RDONLY)
9. If fd = -1
 9.1 Print error message and exit
10. Read and copy the content to buf [13]
11. Close fd
12. Print buf
13. Stop

Program :

```
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
void main()
{
    int fd; char buf[14];
    fd = open("myfile.txt",O_CREAT | O_WRONLY,0600);
    if( fd == -1){
        printf("\n couldn't create file and open the file");
        exit(1);
    }
    write(fd,"Hello World!\n",13);
    close(fd); fd = open("myfile.txt",O_RDONLY);
    if( fd == -1){
        printf("\n couldn't open file and read the file");
        exit(1);
    }
    read(fd,buf, 13);
    buf[13]='\0'; close(fd);
    printf("%s\n",buf);
}
```

Output :

```
Hello World!

...
...Program finished with exit code 0
Press ENTER to exit console.

1 Hello World!
2
```

Result :

Program compiled and executed successfully.

Experiment No : 06

Date : 27 May 2021

Process Scheduling

Aim :

To write a C program to enter a list of processes, their CPU burst times, arrival times and display/print the Gantt chart for FCFS and SJF. For each of the scheduling policies, compute and print the average waiting time and average turnaround time.

Algorithm :

1. Start
2. Enter your choice.
3. If choice = 1, then use the FCFS method.
 - 3.1 Input the number of processes
 - 3.2 Input burst time and arrival time for each processes.
 - 3.3 Sort the process according to their burst time.
 - 3.4 After sorting insert the process details to linked List.
 - 3.5 The function finddetails() is called.
 - 3.5.1 Process completion time (Pc) = first response + burst_time.
 - 3.5.2 Turn around time = Pc + arrival time
 - 3.5.3 Waiting time = Turn around time - burst_time
 - 3.5.4 First response (fr) = fr + burst_time.

3.6 Function printGantt() is called.

3.6.1 The Gantt chart is displayed using values in process-completion time, PC.

3.7 Function displayDetails() is called

3.7.1 while (root != NULL)

3.7.1.1 Print turn around time and waiting time of each process

4. If choice = 2, then use the SJF method

4.1 Input number of process

4.2 Input process number and store in array P[].

4.3 Input burst time to bt []

4.4 Sort the array bt [] and P [] according to burst_time

4.5 from i=0 to i<n

4.5.1 from j=0 to j<n

$$4.5.1.1 w[i] = w[i] + bt[j]$$

$$4.5.2 tat[i] = wt[i] + bt[i]$$

$$4.5.3 awt = awt + wt[i]$$

$$4.5.4 atat = atat + tat[i]$$

$$4.6 awt = awt/n$$

$$4.7 atat = atat/n$$

4.8 Display all the details

5. If choice = 3, then exit from the program

6. Stop

Program :

```

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int pno;
    int at,bt,pc,tat,wt;
    struct node *link;
};
struct node
*temp,*newnode,*start=NULL,*ready=NULL,*newnode1,*temp1,*temp2,*newnode2;
struct pr
{
    int at,bt,no;
};
void findDetails(struct node *root);
void printGantt(struct node *root,int n);
void displayDetails(struct node *root,int n);
void findprocess(int time);
int main()
{
    int n,i=0,j=0;
    int ch=0;
    while (ch!=3)
    {
        printf("\n1.Use FCFS\n2.Use SJF\n3.Exit\nEnter a Choice : ");
        scanf("%d",&ch);
        switch (ch)
        {
            case 1:
            {
                printf("Enter the number of Processes : ");
                scanf("%d",&n);
                struct pr pp[n].tv;
                for (i = 0; i<n; i++)
                {
                    printf("Enter Details of Process %d",i+1);
                    printf("\nEnter Arrival Time : ");
                    scanf("%d",&pp[i].at);
                    printf("Burst Time : ");

```

```

        scanf("%d",&pp[i].bt);
        printf("\nProcess Number : ");
        scanf("%d",&pp[i].no);
    }
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            if (pp[j].at>pp[j+1].at)
            {
                tv = pp[j];
                pp[j]=pp[j+1];
                pp[j+1]=tv;
            }
        }
    }
    for (i = 0; i < n; i++)
    {
        newnode = (struct node*)malloc(sizeof(struct
node));
        newnode->link = NULL;
        newnode->at = pp[i].at;
        newnode->bt = pp[i].bt;
        newnode->pno = pp[i].no;
        if (start==NULL)
        {
            start = newnode;
            temp = newnode;
        }
        else
        {
            temp->link=newnode;
            temp=newnode;
        }
    }
    findDetails(start);
    printGantt(start,n);
    displayDetails(start,n);
    break;
}

```

case 2:

```

{
int i,j,n,t,p[30],bt[30],wt[30],tat[30];
float awt=0,atat=0;
printf("Enter the Number of process : ");
scanf("%d",&n);
printf("Enter the Process Number : ");
for (i = 0; i < n; i++)
{
    scanf("%d",&p[i]);
}
printf("Enter the Burst time of the processes : ");
for (i = 0; i < n; i++)
{
    scanf("%d",&bt[i]);
}
for (i = 0; i < n; i++)
{
    for (j = 0; j < n-i-1; j++)
    {
        if (bt[j] > bt[j+1])
        {
            t = bt[j];
            bt[j] = bt[j+1];
            bt[j+1] = t;
            t = p[j];
            p[j] = p[j+1];
            p[j+1] = t;
        }
    }
}
for (i = 0; i < n; i++)
{
    wt[i]=0;
    tat[i]=0;
    for (j = 0; j < i; j++)
    {
        wt[i] = wt[i]+bt[j];
    }
    tat[i] = wt[i]+bt[i];
}

```

```

awt = awt + wt[i];
atat = atat + tat[i];
printf("\n");
printf(" p%d\t",p[i]);
printf(" %d\t",bt[i]);
printf(" %d\t",wt[i]);
printf(" %d\t",tat[i]);
printf("\n");
}
printf("\n\nGantt Chart\n");
for (i = 0; i < n; i++)
{
    printf(".....");
}
printf("\n");
for (i = 0; i < n; i++)
{
    printf("\tP%d\t",p[i]);
}
printf("|");
printf("\n");
for (i = 0; i < n; i++)
{
    printf(".....");
}
printf("\n");
printf("0");
for (i = 0; i < n; i++)
{
    printf("\t\t%d",tat[i]);
}
awt = awt/n;
atat = atat/n;
printf("\nAverage Waiting time = %f",awt);
printf("\nAverage Turn Around time = %f",atat);
break;
}
case 3:
{
break;
}

```

```

        }
        default:
        {
            printf("\nInvalid Case");
            break;
        }
    }
    return 0;
}
void displayDetails(struct node *root,int n)
{
    float avgat=0,avgwt=0;
    printf("\n\nThe Details of the Process :");
    while (root!=NULL)
    {
        printf("\n\nProcess %d",root->pno);
        printf("\nTurn Around Time : %d",root->tat);
        printf("\nWaiting Time %d",root->wt);
        avgat = avgat+root->tat;
        avgwt = avgwt+root->wt;
        root = root->link;
    }
    avgat = avgat/n;
    avgwt = avgwt/n;
    printf("\nAverage Turn Around Time : %.2f",avgat);
    printf("\nAverage Waiting Time : %.2f",avgwt);
}
void printGantt(struct node *root,int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf(".....");
    }
    printf("\n");
    temp=root;
    for (i = 0; i < n; i++)
    {
        printf("\tP%d\t",temp->pno);
    }
}

```

```

        temp = temp->link;
    }
    printf("|\n");
    printf("\n");
    for (i = 0; i < n; i++)
    {
        printf(".....");
    }
    printf("\n");
    printf("%d",start->at);
    temp = root;
    for (i = 0; i < n; i++)
    {
        printf("\t\t%d",temp->pc);
        temp = temp->link;
    }
}
void findDetails(struct node *root)
{
    int fr=0,ad;
    while (root!=NULL)
    {
        ad = fr-root->at;
        if (ad<0)
        {
            fr = fr + (ad*-1);
        }
        root->pc = fr+root->bt;
        root->tat = root->pc-root->at;
        root->wt = root->tat-root->bt;
        fr = fr+root->bt;
        root = root->link;
    }
}

```

Output :

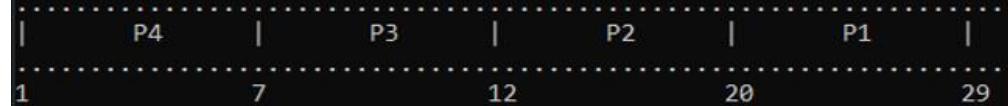
```

1.Use FCFS
2.Use SJF
3.Exit
Enter a Choice : 1
Enter the number of Processes : 4
Enter Details of Process 1
Arrival Time : 5
Burst Time : 9

```

```
Enter Details of Process 2
Arrival Time : 4
Burst Time : 8
Enter Details of Process 3
Arrival Time : 2
Burst Time : 5
Enter Details of Process 4
Arrival Time : 1
Burst Time : 6
```

Gantt chart



process	waiting time	turnaround time
4	0	6
3	5	10
2	8	16
1	15	24

Average Turn Around Time : 14.00

Average Waiting Time : 7.00

1.Use FCFS

2.Use SJF

3.Exit

Enter a Choice : 2

Enter the Number of process : 5

Enter the Process Number :

1

2

3

4

5

Enter the Burst time of the processes :

5

8

2

3

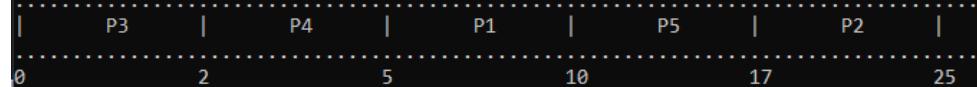
7

process	burst time	waiting time	turnaround time
p3	2	0	2
p4	3	2	5
p1	5	5	10
p5	7	10	17
p2	8	17	25

Average Waiting time = 6.800000

Average Turn Around time = 11.800000

Gantt Chart



```
1.Use FCFS
2.Use SJF
3.Exit
Enter a Choice : 3

-----
(program exited with code: 0)

Press any key to continue . . . ■
```

Result :

Program compiled and executed successfully.

Experiment No : 07

Date : 03 June 2021

CPU Scheduling

Aim :

To write a C program to simulate following non – preemptive CPU scheduling algorithms to find turnaround time and waiting time.

- a) First Come First Serve (FCFS)
- b) Shortest Job First (SJF)
- c) Round Robin Scheduling (Pre – emptive)
- d) Priority Scheduling

Algorithm :

1. Start.
2. Enter your choice.
3. If choice = 1, then use the FCFS method.
 - 3.1 Define necessary variables
 - 3.2 Read the value n, assign process number as i and get the value of $P[i]$ and burst time.
 - 3.3 Assign waiting time of $P[0]$ as zero and total time as burst time.
 - 3.4 From $i=1$ to $i < n$
 - 3.4.1 $wt[i] = 0$
 - 3.4.2 From $j=0$ to $j < i$
 $wt[i] += bt[j]$
 - 3.5 Display process, burst time, waiting time and turn around time.
 - 3.6 Calculate the time.

- 3.7 Display the time.
- 4 If choice = 2, then SJF method is called.
- 4.1 Declare necessary variables
- 4.2 Sort the data according to order of burst time.
- 4.3 Assign the value of $P[0] = 0$ and total time as burst time.
- 4.6 From $i=1$ to $i < n$ do,
- 4.6.1 $WT[i] = 0$
- 4.6.2 From $j=0$ to $j < i$ do,
- 4.6.2.1 $WT[i] += BT[j]$
- 4.6.2.2 $Total += WT[i]$
- 4.7 Display the time for every process.
- 4.8 Calculate the average waiting time and average turn around time by dividing it with total number of processes.
5. If choice = 3, then round robin method is called.
- 5.1 Declare necessary variables
- 5.2 Assign the value of turn around time & waiting time as zero.

- 5.3 Enter the total process number as n .
- 5.4 From $count = 0$ to $count < n$ accept the burst time for each process and copy $bt[count] = rt[count]$
- 5.5 Read the value of quantum time from the user
- 5.6 For $time = 0$ & $count = 0$ till remain != 0 do,
- 5.6.1 if $rt[count] \leq time_quantum$ and $rt[count] > 0$, then
 $time += rt[count]$
 $rt[count] = 0$
 $\downarrow flag = 1$
- 5.6.2 else if $rt[count] > 0$
 $rt[count] -= time_quantum$
 $time += time_quantum$.
- 5.6.3 if $rt[count] == 0$ and $flag = 1$
 5.6.3.1 Decrement the number of processes and display wait time and turn around time
- 5.6.4 if $count == n-1$
 5.6.4.1 set $count = 0$
- 5.6.5 else if $rt[count+1] \leq time$
 5.6.5.1 $count++$
- 5.6.6 else
 5.6.6.1 set $count = 0$
- 5.7 Print the average waiting and turn-around time.

6. If choice = 4, then priority scheduling method is called.
 - 6.1 Create a structure with necessary values
 - 6.2 Define necessary variables
 - 6.3 Accept the number of processes
 - 6.4 Accept the burst time and priority for each processes.
 - 6.5 Sort the process based on the priority.
 - 6.6 Display the burst time, waiting time and total turn around time.
 - 6.7 From $i=0$ to $i < n-1$
 - 6.7.1 totwttime += p[i]. wtime = tbtm
 - 6.7.2 tbtm += p[i]. btime
 - 6.8 Print average waiting time, total turn around time and average waiting time
7. If choice = 5, then exit from the program
8. Else print "Invalid choice"
9. Stop.

Program :

```
#include <stdio.h>
#include <stdlib.h>
void FCFS();
```

```
void SJF();
void RR();
void PS();
int main()
{
    printf("\n\n 1. FCFS \n 2. SJF \n 3. Round Robin \n 4. Priority Scheduling \n 5.
Exit");
    printf("\n Enter your choice : ");
    int cho;
    scanf("%d",&cho);
    switch(cho)
    {
        case 1:
            FCFS();
            main();
            break;

        case 2:
            SJF();
            main();
            break;

        case 3:
            RR();
            main();
            break;

        case 4:
            PS();
            main();
            break;

        case 5:
            exit(0);
            break;

        default:
            printf("Invalid Choice");
            break;
    }
}
```

```

    }
void FCFS()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf(" Enter total number of processes : ");
    scanf("%d",&n);
    printf("\n Enter Process Burst Time : ");
    for(i=0;i<n;i++)
    {
        printf("\n P[%d] : ",i+1);
        scanf("%d",&bt[i]);
    }
    wt[0]=0;
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
        {
            wt[i]+=bt[j];
        }
    }
    printf("\n Process\tBurst Time\tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("\n P[%d] \t %d \t %d \t %d",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("\n\n Average Waiting Time:%d",avwt);
    printf("\n Average Turnaround Time:%d",avtat);
}
void SJF()
{
    int bt[20],p[20],wt[20],tat[20],i,j,n,total=0,pos,temp;
    float avg_wt,avg_tat;
    printf(" Enter number of process:");
    scanf("%d",&n);

```

```

printf("\n Enter Process Burst Time : ");
for(i=0;i<n;i++)
{
    printf("\n P[%d] : ",i+1);
    scanf("%d",&bt[i]);
    p[i]=i+1;
}
for(i=0;i<n;i++) // sorting of burst times
{
    pos=i;
    for(j=i+1;j<n;j++)
    {
        if(bt[j]<bt[pos])
        {
            pos=j;
        }
    }
    temp=bt[i];
    bt[i]=bt[pos];
    bt[pos]=temp;
    temp=p[i];
    p[i]=p[pos];
    p[pos]=temp;
}
wt[0]=0;
for(i=1;i<n;i++)
{
    wt[i]=0;
    for(j=0;j<i;j++)
    wt[i]+=bt[j];
    total+=wt[i];
}
avg_wt=(float)total/n;
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)
{
    tat[i]=bt[i]+wt[i];
    total+=tat[i];
    printf("\n p%d\t %d\t %d\t %d",p[i],bt[i],wt[i],tat[i]);
}

```

```

    }
    avg_tat=(float)total/n;
    printf("\n\nAverage Waiting Time=%f",avg_wt);
    printf("\n\nAverage Turnaround Time=%f\n",avg_tat);
}
void RR()
{
    int count,n,time,remain,flag=0,time_quantum;
    int wait_time=0,turnaround_time=0,at[10],bt[10],rt[10];
    printf(" Enter Total Process:\t");
    scanf("%d",&n);
    remain=n;
    printf("\n Enter Process Burst Time : ");
    for(count=0;count<n;count++)
    {
        printf("\n P[%d] : ",count+1);
        at[count] = count;
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf(" Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
    printf("\nProcess\t Turnaround Time \tWaiting Time\n");
    for(time=0,count=0;remain!=0;)
    {
        if(rt[count]<=time_quantum && rt[count]>0)
        {
            time+=rt[count];
            rt[count]=0;
            flag=1;
        }
        else if(rt[count]>0)
        {
            rt[count]-=time_quantum;
            time+=time_quantum;
        }
        if(rt[count]==0 && flag==1)
        {
            remain--;
        }
    }
}

```

```

        printf("P[%d]\t%d\t%d\n",count+1,time-at[count], time
        -at[count]-bt[count]);
        wait_time+=time-at[count]-bt[count];
        turnaround_time+=time-at[count];
        flag=0;
    }
    if(count==n-1)
    {
        count=0;
    }
    else if(at[count+1]<=time)
    {
        count++;
    }
    else
    {
        count=0;
    }
}
printf("\n Average Waiting Time= %f\n",wait_time*1.0/n);
printf(" Avg Turnaround Time = %f",turnaround_time*1.0);
}

void PS()
{
    typedef struct
    {
        int pno;
        int pri;
        int btime;
        int wtime;
    }sp;
    int i,j,n;
    int tbm=0,totwtime=0,totttime=0;
    sp *p,t;
    printf("\n Enter the no of process : ");
    scanf("%d",&n);
    p=(sp*)malloc(sizeof(sp));
    printf(" Enter the burst time and priority : ");
    for(i=0;i<n;i++)

```

```

{
    printf("\n P[%d] : ",i+1);
    scanf("%d%d",&p[i].btime,&p[i].pri);
    p[i].pno=i+1;
    p[i].wtime=0;
}
for(i=0;i<n-1;i++)
{
    for(j=i+1;j<n;j++)
    {
        if(p[i].pri>p[j].pri)
        {
            t=p[i];
            p[i]=p[j];
            p[j]=t;
        }
    }
}
printf("\n Process\tBurst time\tWaiting time\tTurnaround Time\n");
for(i=0;i<n;i++)
{
    totwtime+=p[i].wtime=tbm;
    tbm+=p[i].btime;
    printf("\n%d\t%d",p[i].pno,p[i].btime);
    printf("\t%d\t%d",p[i].wtime,p[i].wtime+p[i].btime);
}
tottime=tbm+totwtime; printf("\n total waiting time:%d",totwtime);
printf("\n Average waiting time:%f",(float)totwtime/n);
printf("\n Total turnaround time:%d",tottime);
printf("\n Avg turnaround time:%f",(float)tottime/n);
}

```

Output :

```

1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
Enter your choice : 1
Enter total number of processes : 4

Enter Process Burst Time :
P[1] : 6

```

```

P[2] : 7
P[3] : 8
P[4] : 2

Process      Burst Time      Waiting Time      Turnaround Time
P[1]           6                  0                  6
P[2]           7                  6                  13
P[3]           8                 13                  21
P[4]           2                 21                  23

Average Waiting Time:10
Average Turnaround Time:15

1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
Enter your choice : 2
Enter number of process:4

Enter Process Burst Time :
P[1] : 6

P[2] : 5

P[3] : 7

P[4] : 3

Process      Burst Time      Waiting Time      Turnaround Time
p4            3                  0                  3
p2            5                  3                  8
p1            6                  8                  14
p3            7                 14                  21

Average Waiting Time=6.250000
Average Turnaround Time=11.500000

1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
Enter your choice : 3
Enter Total Process:    4

Enter Process Burst Time :
P[1] : 5

P[2] : 4

P[3] : 8

P[4] : 1
Enter Time Quantum:    3

Process      Turnaround Time      Waiting Time
P[4]           7                  6
P[1]           12                 7
P[2]           12                 8
P[3]           16                 8

Average Waiting Time= 7.250000
Avg Turnaround Time = 47.000000

```

```
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
Enter your choice : 4

Enter the no of process : 4
Enter the burst time and priority :
P[1] : 5 8

P[2] : 4 2

P[3] : 9 7

P[4] : 4 1

      Process          Bursttime        Waiting time    Turnaround Time
p4           4                  0                 4
p2           4                  4                 8
p3           9                  8                17
p1           5                17                22
total waiting time:29
Average waiting time:7.250000
Total turnaround time:51
Avg turnaround time:12.750000
1. FCFS
2. SJF
3. Round Robin
4. Priority Scheduling
5. Exit
Enter your choice : 5

-----
(program exited with code: 0)

Press any key to continue . . .
```

Result :

Program compiled and executed successfully.

Experiment No : 08

Date : 09 September 2021

Memory Allocation

Aim :

To write a C program to simulate the following contiguous memory allocation techniques :

- a) Worst-fit
- b) Best-fit
- c) First-fit.

Algorithm :

1. Start
2. Read the number of blocks as b
3. Read the size of each block, blocks []
4. Read the number of process as p and size of each process as process []
5. If the choice == 1, then user defined function first fit is called
 - 5.1 From i = 0 to i < p

Scan each block and allocate it to process
 if (block size \geq process size) else set
 allocation[i] = -999
6. If the choice == 2, the best fit user defined function is called.

6.1 From $i=0$ to $i < p$

6.1.1 From $j=0$ to $j < b$

↓

Set $small = block[j]$
 $small = bwfinder(j, process[i], small, 1)$ function is called and
 Allocate $small$ to $process[i]$ if
 $process[i] \leq block[j]$
 Else, no blocks can allocate then set
 $allocate[i] = -999$

7. If the choice == 3 then user defined function worst fit is called

7.1 From $i=0$ to $i < p$

7.1.1 From $j=0$ to $j < b$

↓

Set $big = block[j]$
 If $process[i] \leq block[j]$ then
 $big = bwfinder(j, process[i], small, 1)$ function is called.
 Allocate big to $process[i]$
 Else, no blocks can allocate then
 set $allocate[i] = -999$.

8. bwFinder(int startIndex, int psize, int curValue, int mode) is declared.
- 8.1 Find the smallest or biggest blocks to be allocated to the process based on which mode is selected
- 8.2 If mode = 1 then smallest block that can accommodate process is returned which is < curValue.
- 8.3 Else the biggest block that can accommodate the process is returned which is > curValue
9. Display the process number and block allocated
10. If the choice = 4, then exit
11. Stop

Program :

```
#include <stdio.h>
int block[20], process[20], isAllocated[20] = {0}, allocated[20], b, p, choice, flag = 0;
void firstFit();
void bestFit();
void worstFit();
int bwFinder(int startIndex, int pSize, int curValue, int mode);
void display();
int main()
{
```

```
printf("Enter the number of blocks : ");
scanf("%d", &b);
printf("Enter the size of each block\n");
for (int i = 0; i < b; i++)
{
    printf("B%d : ", i);
    scanf("%d", &block[i]);
}
printf("Enter the number of process : ");
scanf("%d", &p);
printf("Enter the size of each process\n");
for (int i = 0; i < p; i++)
{
    printf("P%d : ", i);
    scanf("%d", &process[i]);
}
while (choice != 4)
{
    printf("\n1.First Fit\n2.Best Fit\n3.Worst Fit\n4.Exit\n");
    printf("Enter a choice : ");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:
        {
            firstFit();
            display();
            break;
        }
        case 2:
        {
            bestFit();
            display();
            break;
        }
        case 3:
        {
            worstFit();
            display();
            break;
        }
        case 4:
        {
            break;
        }
    default:
```

```

    {
        printf("\nInvalid choice");
    }
}

void display()
{
    printf("\nProcess\t\tProcess size\t\tBlock Allocated\n");
    for (int i = 0; i < p; i++)
    {
        if (allocated[i] == -999)
        {
            printf("%d\t%d\tNot allocated", i, process[i]);
            printf("\n");
        }
        else
        {
            printf("%d\t%d\t%d", i, process[i], allocated[i]);
            printf("\n");
        }
    }
}

void firstFit()
{
    for (int i = 0; i < b; i++)
    {
        isAllocated[i] = 0;
    }
    for (int i = 0; i < p; i++)
    {
        flag = 0;
        for (int j = 0; j < b; j++)
        {
            if (process[i] <= block[j] && isAllocated[j] == 0)
            {
                allocated[i] = block[j];
                isAllocated[j] = 1;
                flag = 1;
                break;
            }
        }
        if (flag == 0)
        {
            allocated[i] = -999;
        }
    }
}

```

```

        }
    }
void bestFit()
{
    for (int i = 0; i < b; i++)
    {
        isAllocated[i] = 0;
    }
    for (int i = 0; i < p; i++)
    {
        int small = 0;
        for (int j = 0; j < b; j++)
        {
            if (process[i] <= block[j] && isAllocated[j] != 1)
            {
                small = block[j];
                isAllocated[j] = 1;
                small = bwFinder(j, process[i], small, 1);
                allocated[i] = small;
                break;
            }
        }
        if (small == 0)
        {
            allocated[i] = -999;
        }
    }
}
void worstFit()
{
    for (int i = 0; i < b; i++)
    {
        isAllocated[i] = 0;
    }
    for (int i = 0; i < p; i++)
    {
        int big = 0;
        for (int j = 0; j < b; j++)
        {
            if (process[i] <= block[j] && isAllocated[j] != 1)
            {
                big = block[j];
                isAllocated[j] = 1;
                big = bwFinder(j, process[i], big, 2);
                allocated[i] = big;
                break;
            }
        }
    }
}

```

```

        }
    }
    if (big == 0)
    {
        allocated[i] = -999;
    }
}
int bwFinder(int startIndex, int pSize, int curValue, int mode)
{
    int lastBlock = startIndex;
    for (int i = startIndex + 1; i < b; i++)
    {
        if (pSize <= block[i] && (mode == 1 ? block[i] < curValue : block[i] >
curValue) && isAllocated[i] != 1)
        {
            isAllocated[lastBlock] = 0;
            lastBlock = i;
            curValue = block[i];
            isAllocated[i] = 1;
        }
    }
    return curValue;
}

```

Output :

```

Enter the number of blocks : 5
Enter the size of each block
B0 : 36
B1 : 95
B2 : 84
B3 : 76
B4 : 29
Enter the number of process : 5
Enter the size of each process
P0 : 63
P1 : 25
P2 : 10
P3 : 98
P4 : 47

1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter a choice : 1

```

Process	Process size	Block Allocated
0	63	95
1	25	36
2	10	84
3	98	Not allocated
4	47	76

1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter a choice : 2

Process	Process size	Block Allocated
0	63	76
1	25	29
2	10	36
3	98	Not allocated
4	47	84

1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter a choice : 3

Process	Process size	Block Allocated
0	63	95
1	25	84
2	10	76
3	98	Not allocated
4	47	Not allocated

1.First Fit
2.Best Fit
3.Worst Fit
4.Exit
Enter a choice : 4

Result :

Program compiled and executed successfully.

Experiment No : 09

Date : 09 September 2021

Paging Technique

Aim :

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

Algorithm :

1. Start
2. Declare variables
3. Read the number of pages in a memory as n
4. Read the page size as 'ps' and number of frames as t
5. From $i=0$ to $i < n$
 - 5.1 Assign $\text{Page}[i] = -1$
6. Using for loop read the page table
7. Read the logical address, page number as pno and offset as off
8. If $\text{page}[pno] == -1$, then print the required page is not available in any frame

- q. Else print the frame number and offset in available page [Pno]
10. Stop

Program :

```
#include<stdio.h>
int main()
{
    int page[50],i,n,f,ps,off,pno;
    printf("\nEnter the no of pages in memory : ");
    scanf("%d",&n);
    printf("\nEnter page size : ");
    scanf("%d",&ps);
    printf("\nEnter no of frames : ");
    scanf("%d",&f);
    for(i=0;i<n;i++)
    {
        page[i] = -1;
    }
    printf("\nEnter the page table");
    printf("(Enter frame no as -1 if that page is not present in any frame) \n");
    printf("\nPage No :\tFrame No :");
    for(i=0;i<n;i++)
    {
        printf("\n%d\t\t,i);
        scanf("%d",&page[i]);
    }
    printf("\nLogical address (page no & offset) : ");
    scanf("%d%d",&pno,&off);
    if(page[pno]==-1)
    {
        printf("\nThe required page is not available in any of frames");
    }
    else
    {
        printf("\nPhysical address(frame no & offset) : %d, %d\n",page[pno],off);
    }
}
```

```
        return 0;  
    }
```

Output :

```
Enter the no of pages in memory : 6  
Enter page size : 20  
Enter no of frames : 4  
Enter the page table(Enter frame no as -1 if that page is not present in any frame)  
Page No :      Frame No :  
0             5  
1             2  
2             7  
3             8  
4             9  
5             1  
Logical address (page no & offset) : 3 10  
Physical address(frame no & offset) : 8, 10
```

Result :

Program compiled and executed successfully.

Experiment No : 10

Date : 24 June 2021

Bankers Algorithm

Aim :

To write a C program to simulate Bankers Algorithm for the purpose of deadlock avoidance.

Algorithm :

1. Start .
2. Input function is invoked,
 2. 1 Accept total number of resources in system to variable m.
 2. 2 Accept total number of processes.
 2. 3 Declare necessary 1D and 2D arrays.
 2. 4 Read available resources from user, store it in available array and display.
 2. 5 Read currently allocated resources for each process from user, store it in allocation array and display.
 2. 6 Read maximum allocated resources for each process.
 2. 7 Safety function is invoked by passing available, Max, Allocation arrays as arguments.
3. In safety function,
 3. 1 Declare 1D array Work of size m to copy Available, Finish of size n to indicate which all processes has completed their allocation, 2D array need of size n,m to indicate the remaining resources needed for each process.

3.2 From $i=0$, till i reaches m .

$\text{Work}[i] = \text{Available}[i]$.

Increment i .

3.3 From $i=0$, till i reaches n .

$\text{Finish}[i] = 0$.

3.4 From $i=0$, till i reaches n .

3.4.1 From $j=0$, till j reaches m .

3.4.1.1 $\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$.

3.5 Display Need array.

3.6 Declare 1D array safe sequence of size n , initialize as 0, count as 0 which is used to repeat the allocation of cycles n times.

3.7 While count not equal to n .

3.7.1 From $i=0$ to i reaches n .

Initialize counter as 0 which is used as a counter variable.

From $j=0$, till j reaches m .

If $\text{Need}[i][j] \leq \text{Work}[i][j]$ &

$\text{Finish}[i] == 0$

Increment counter by 1

Else, break

- Increment j .
- Assign $\text{Finish}[i]$ as 1, $\text{Safe sequence}[s]$ as i , Increment s .
- 3.7.2 Increment count by one.
- 3.8 Assign $\text{flag} = 0$, which is used as counter variable.
- 3.8.1 From $i=0$ till i reaches s .
 If $\text{Finish}[i] == 1$ then,
 Increment flag by one.
- 3.8.2 If $\text{flag} == n$
 Display system is in safe state and safe sequence.
- 3.8.3 Else
 Display system is not in safe state.
4. Stop.

Program :

```
#include <stdio.h>
#include <stdlib.h>
int m, n;
void Safety(int Available[], int Max[][m], int Allocation[][m])
{
    int Work[m], Finish[n], Need[n][m];
    for (int i=0;i<m;i++)
    {
        Work[i] = Available[i];
    }
    for (int i=0;i<n;i++)
    {
        Finish[i] = 0;
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (Allocation[i][j] < Max[i][j])
            {
                Need[i][j] = Max[i][j] - Allocation[i][j];
            }
            else
            {
                Need[i][j] = 0;
            }
        }
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            if (Work[j] >= Need[i][j])
            {
                Finish[i]++;
                Work[j] -= Need[i][j];
            }
        }
    }
    if (Finish[n-1] == n)
    {
        printf("System is in Safe State\n");
    }
    else
    {
        printf("System is not in Safe State\n");
    }
}
```

```

{
    for (int j = 0; j < m; j++)
    {
        Need[i][j] = Max[i][j] - Allocation[i][j];
    }
}
printf("\nNeed\n");
printf("====\n");
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++){
        printf("%d ", Need[i][j]);
        printf("\n");
    }
}
int s = 0;
int safeSequence[n];
int count = 0;
while (count != n)
{
    for (int i = 0; i < n; i++)
    {
        int counter = 0;
        for (int j = 0; j < m; j++)
        {
            if (Need[i][j] <= Work[j] && Finish[i] == 0)
            {
                counter++;
            }
            else
            {
                break;
            }
        }
        if (counter == m)
        {
            for (int j = 0; j < m; j++){
                Work[j] = Work[j] + Allocation[i][j];
            }
            Finish[i] = 1;
        }
    }
}

```

```
        safeSequence[s] = i;
        s++;
    }
}
count++;

}

int flag = 0;
for(int i = 0; i < s; i++)
{
    if (Finish[i] == 1){
        flag++;
    }
}
if (flag == n)
{
    printf("\nSystem is in Safe State!\n");
    printf("Safe Sequence:");
    for (int i = 0; i < s; i++){
        printf("P%d ", safeSequence[i]);
    }
    printf("\n");
}
else
{
    printf("\nSystem is not in safe state");
}
}

void input()
{
    int i, j;
    printf("Total resources in system:");
    scanf("%d", &m);
    printf("Enter the number of processes:");
    scanf("%d", &n);
    int Available[m], Max[n][m], Allocation[n][m];
    printf("\nEnter the available resources:\n");
    for (i = 0; i < m; i++)
    {
        printf(":");
        scanf("%d", &Available[i]);
    }
}
```

```
}

printf("\nAvailable\n");
printf("=====\\n");
for (i = 0; i < m; i++)
{
    printf("%d ", Available[i]);
}
printf("\nEnter the currently allocated resources for each process:\\n");
for (i = 0; i < n; i++)
{
    for (j = 0; j < m; j++){
        printf(":");
        scanf("%d", &Allocation[i][j]);
    }
}
printf("\nAllocation\\n");
printf("=====\\n");
for(int i=0;i<n;i++)
{
    for(int j=0;j<m;j++){
        printf("%d ",Allocation[i][j]);
    }
    printf("\\n");
}
printf("\nEnter the maximum allocated resources for each process:\\n");
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < m; j++){
        printf(":");
        scanf("%d", &Max[i][j]);
    }
}
printf("\nMax\\n");
printf("==\\n");
for(int i=0;i<n;i++)
{
    for(int j=0;j<m;j++)
    {
        printf("%d ",Max[i][j]);
    }
}
```

```

        printf("\n");
    }
    Safety(Available, Max, Allocation);
}
int main()
{
    input();
    return 0;
}

```

Output :

```

C:\Windows\system32\cmd.exe
C:\Users\aadhi\Desktop>a.exe
Total resources in system : 4
Enter the number of processes : 5

Enter the available resources : 1 5 2 0

Available
=====
1 5 2 0
Enter the currently allocated resources for each process :  0 0 1 2 1 0 0 0 1 3 5 4 0 6 3 2 0 0 1 4

Allocation
=====
0 0 1 2
1 0 0 0
1 3 5 4
0 6 3 2
0 0 1 4

Enter the maximum allocated resources for each process : 0 0 1 2 1 7 5 0 2 3 5 6 0 6 5 2 0 6 5 6

Max
===
0 0 1 2
1 7 5 0
2 3 5 6
0 6 5 2
0 6 5 6

Need
===
0 0 0 0
0 7 5 0
1 0 0 2
0 0 2 0
0 6 4 2

System is in Safe State!
Safe Sequence : P0 P2 P3 P4 P1

C:\Users\aadhi\Desktop>_

```

Result :

Program compiled and executed successfully.

Experiment No : 11

Date : 23 September 2021

Disk Scheduling

Aim :

To write a C program to simulate the following disk scheduling algorithms :

- a) FCFS
- b) SCAN
- c) C – SCAN

Algorithm :

1. Start
2. Read the maximum range of the disks as range
3. Read the number of requests req
4. Read the disk position to be read starting from pos 1, requests [] .
5. Read the initial head position, head
6. Read a choice.
7. If choice = 1, then fcfs() is called
 - 7.1 Set request[0] = head
 - 7.2 Loop through the array request[] and find seek time.
 - 7.2.1 Seek = abs (request[i+1] - request[i])
 - 7.3 Add the individual seek times together and

- find total seek time.
- 7.4 Print the total seek time
8. If choice = 2, then scan() is called
- 8.1 Store the disk positions which are greater than head and less than head in two different arrays, $q_1[]$, $q_2[]$
- 8.2 Sort $q_2[]$ in ascending order and $q_1[]$ in descending order
- 8.3 Store $q_1[]$ and $q_2[]$ in a single array[] from position 1 with maxrange value in middle of the two array values.
- 8.4 Set queue[0] = head
- 8.5 Loop array queue[] and find individual seek times and add them to get total seektime
- 8.6 Print total seek time.
9. If choice = 3, then escan() is called
- 9.1 Repeat step 8.1
- 9.2 Sort both $q_1[]$ and $q_2[]$ in ascending order
- 9.3 Store $q_1[]$ into queue[] from pos 1
- 9.4 Add maxrange value to end of queue[]
- 9.5 Add 0 to end of queue []

- 9.6 Append $q_2[]$ to end of $queue[]$
- 9.7 Repeat step 8.5 to 8.6 for $queue[]$
10. Stop

Program :

```
#include<stdio.h>
#include<math.h>
int range,req,requests[50],head;
void fcfs();
void scan();
void cscan();
void sort(int *arr,int size,int mode);
void seektimeFinder(int *arr,int max);
int main()
{
    printf("\nEnter the max range of the disk : ");
    scanf("%d",&range);
    printf("\nEnter the number of requests : ");
    scanf("%d",&req);
    printf("\nEnter the disk positions to be read\n");
    for (int i = 1; i <=req; i++)
    {
        scanf("%d",&requests[i]);
    }
    printf("\nEnter the initial head position : ");
    scanf("%d",&head);
    int choice=0;
    while (choice != 4)
    {
        printf("\n1.FCFS\n2.SCAN\n3.C-SCAN\n4.Exit\nEnter a choice : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {

```

```

        fcfs();
        break;
    }
    case 2:
    {
        scan();
        break;
    }
    case 3:
    {
        cscan();
        break;
    }
    case 4:
    {
        break;
    }
    default:
    {
        printf("\nInvalid case.. ");
    }
}
}

void seektimeFinder(int *arr,int max)
{
    int seek=0,diff=0;
    for (int i = 0; i <max-1; i++)
    {
        diff = abs(arr[i+1] - arr[i]);
        seek += diff;
        printf("\nDisc head moves from %d to %d with seek time
        %d",arr[i],arr[i+1],diff);
    }
    printf("\n\nTotal seek time : %d",seek);
}
void fcfs()
{
    requests[0] = head;
}

```

```

    seektimeFinder(requests,req+1);
}
void scan()
{
    int q1[20],q2[20],k1=0,k2=0,queue[50],t1=0;
    for (int i = 1; i <=req; i++)
    {
        if (requests[i] > head)
        {
            q2[k2] = requests[i];
            k2++;
        }
        else
        {
            q1[k1] = requests[i];
            k1++;
        }
    }
    sort(q2,k2,1);
    sort(q1,k1,2);
    t1=1;
    for (int i = 0; i <k2; i++)
    {
        queue[t1] = q2[i];
        t1++;
    }
    queue[0] = head;
    queue[t1] = range;
    t1++;
    for (int i = 0; i < k1; i++) //then reverse
    {
        queue[t1] = q1[i];
        t1++;
    }
    seektimeFinder(queue,t1);
}
void cscan()
{
    int q1[20],q2[20],k1=0,k2=0,queue[50],t1=0;
    for (int i = 1; i <=req; i++)

```

```

{
    if (requests[i] > head)
    {
        q2[k2] = requests[i];
        k2++;
    }
    else
    {
        q1[k1] = requests[i];
        k1++;
    }
}
sort(q2,k2,1);
sort(q1,k1,1);
t1=1;
for (int i = 0; i < k2; i++)
{
    queue[t1] = q2[i];
    t1++;
}
queue[0] = head;
queue[t1] = range;
t1++;
queue[t1] = 0;
t1++;
for (int i = 0; i < k1; i++)
{
    queue[t1] = q1[i];
    t1++;
}
seektimeFinder(queue,t1);
}
void sort(int *arr,int size,int mode)
{
    int i,j,temp;
    for (i = 0; i < size-1; i++)
    {
        for (j = 0; j < size-i-1; j++)
        {
            if (mode==1?arr[j] > arr[j+1]:arr[j] < arr[j+1])

```

```

    {
        temp = arr[j];
        arr[j] = arr[j+1];
        arr[j+1] = temp;
    }
}
}
}

```

Output :

```

Enter the max range of the disk : 199
Enter the number of requests : 9
Enter the disk positions to be read
18 90 160 150 38 184 5 58 39
Enter the initial head position : 100
1.FCFS
2.SCAN
3.C-SCAN
4.Exit
Enter a choice : 1

Disc head moves from 100 to 18 with seek time 82
Disc head moves from 18 to 90 with seek time 72
Disc head moves from 90 to 160 with seek time 70
Disc head moves from 160 to 150 with seek time 10
Disc head moves from 150 to 38 with seek time 112
Disc head moves from 38 to 184 with seek time 146
Disc head moves from 184 to 5 with seek time 179
Disc head moves from 5 to 58 with seek time 53
Disc head moves from 58 to 39 with seek time 19

Total seek time : 743
1.FCFS
2.SCAN
3.C-SCAN
4.Exit
Enter a choice : 2

Disc head moves from 100 to 150 with seek time 50
Disc head moves from 150 to 160 with seek time 10
Disc head moves from 160 to 184 with seek time 24
Disc head moves from 184 to 199 with seek time 15
Disc head moves from 199 to 90 with seek time 109
Disc head moves from 90 to 58 with seek time 32
Disc head moves from 58 to 39 with seek time 19
Disc head moves from 39 to 38 with seek time 1
Disc head moves from 38 to 18 with seek time 20
Disc head moves from 18 to 5 with seek time 13

```

```
Total seek time : 293
1.FCFS
2.SCAN
3.C-SCAN
4.Exit
Enter a choice : 3

Disc head moves from 100 to 150 with seek time 50
Disc head moves from 150 to 160 with seek time 10
Disc head moves from 160 to 184 with seek time 24
Disc head moves from 184 to 199 with seek time 15
Disc head moves from 199 to 0 with seek time 199
Disc head moves from 0 to 5 with seek time 5
Disc head moves from 5 to 18 with seek time 13
Disc head moves from 18 to 38 with seek time 20
Disc head moves from 38 to 39 with seek time 1
Disc head moves from 39 to 58 with seek time 19
Disc head moves from 58 to 90 with seek time 32

Total seek time : 388
1.FCFS
2.SCAN
3.C-SCAN
4.Exit
Enter a choice : 4
```

Result :

Program compiled and executed successfully.

Experiment No : 12

Date : 16 September 2021

Page Replacement Algorithms

Aim :

To write a C program to simulate the following page replacement algorithms.

- a) First in first out (FIFO)
- b) Least recently used (LRU)
- c) Least frequently used (LFU)

Algorithm :

1. Start.
2. Define required variables globally.
3. Receive the number of pages, reference string and number of frames from the user.
4. User defined function `Fifo()` is called
 - 4.1 Traverse the pages
 - 4.2 Push pages into the queue one at a time until queue reaches its maximum capacity or all page requests are fulfilled.
 - 4.3 If the current page is present in memory do nothing.
 - 4.4 Else, pop the topmost page from the queue as it was first inserted.
 - 4.5 Replace the topmost page with the current page from the string.
 - 4.6 Increment the page faults.
 - 4.7 Display the values

- 4.8 Return the value to pf1
5. User defined function lru() is called
 - 5.1 Assign values in page frame till it gets filled.
 - 5.2 If not filled
 - 5.2.1 Print the value with the unfilled spot printing "-".
 - 5.3 Else
 - 5.3.1 Choose the recently used page by the counter value.
 - 5.3.2 Assign the value to the display array.
 - 5.3.3 Replace the least recently used value with the next value
 - 5.4 Display the values
 - 5.5 Return the value to pf2
6. User defined function lfu() is called
 - 6.1 Assign pages in the free frames
 - 6.2 If free space is available
 - 6.2.1 Increase the value of c
 - 6.3 If no frames is free replace the page with the page that is least used.
 - 6.4 Print array which contains the frame
 - 6.5 Return the value to pf3

7. Print value of pf₁, pf₂, pf₃ as page fault of FIFO, LRU and LFU
8. Stop

Program :

```
#include<stdio.h>
int i,j,n,p[50],f;
int fifo();
int lru();
int lfu();
int main()
{
    int pf1,pf2,pf3;
    printf("Enter no of pages : ");
    scanf("%d",&n);
    printf("Enter the reference string : ");
    for(i=0;i<n;i++)
        scanf("%d",&p[i]);
    printf("Enter no of frames : ");
    scanf("%d",&f);
    pf1=fifo();
    pf2=lru();
    pf3=lfu();
    printf("\n\nPage fault : ");
    printf("\nfifo = %d",pf1);
    printf("\nlru = %d",pf2);
    printf("\nlfu = %d\n\n",pf3);
    return 0;
}
int fifo()
{
    int q[10],c = 0,s;
    printf("\nFirst In First Out");
    for(i=0;i<f;i++)
        q[i] = -1;
    for(i=0;i<n;i++)
    {
        s=0;
        for(j=0;j<f;j++)
        {
            if(p[i] == q[j])
```

```

    {
        s++;
        c--;
    }
}
c++;
if((c <= f) && (s == 0))
q[i] = p[i];
else if(s == 0)
q[(c - 1) % f] = p[i];
printf("\n");
for(j = 0; j < f; j++)
{
    if(q[j]==-1)
    printf("-\t");
    else
    printf("%d\t", q[j]);
}
return c;
}
int lru()
{
    int q[20],c=0,c1,d,k=0,r,t,b[20],c2[20];
    printf("\nLeast Recently Used");
    q[k]=p[k];
    printf("\n%d\t",q[k]);
    for(i=1;i<f;i++)
    printf("-\t");
    printf("\n");
    c++;
    k++;
    for(i=1;i<n;i++)
    {
        c1=0;
        for(j=0;j<f;j++)
        {
            if(p[i]!=q[j])
            c1++;
        }
        if(c1==f)
        {
            c++;
            if(k<f)
            {
                q[k]=p[i];
            }
        }
    }
}

```

```

        k++;
        for(j=0;j<k;j++)
        printf("%d\t",q[j]);
        for(i=0;i<f-k;i++)
        printf("-\t");
        printf("\n");
    }
else
{
    for(r=0;r<f;r++)
    {
        c2[r]=0;
        for(j=i-1;j<n;j--)
        {
            if(q[r]!=p[j])
            c2[r]++;
            else
            break;
        }
    }
    for(r=0;r<f;r++)
    b[r]=c2[r];
    for(r=0;r<f;r++)
    {
        for(j=r;j<f;j++)
        {
            if(b[r]<b[j])
            {
                t=b[r];
                b[r]=b[j];
                b[j]=t;
            }
        }
    }
    for(r=0;r<f;r++)
    {
        if(c2[r]==b[0])
        {
            q[r]=p[i];
        }
        printf("%d\t",q[r]);
    }
    printf("\n");
}
}
}

```

```

        return c;
    }
int lfu()
{
    int q[50],k, cntr[20], min, c=0;
    for(i=0;i<f;i++)
    {
        cntr[i]=0;
        q[i]=-1;
    }
    printf("\nLeast Frequently Used");
    for(i=0;i<n;i++)
    {
        for(j=0;j<f;j++)
        if(p[i]==q[j])
        {
            cntr[j]++;
            break;
        }
        if(j==f)
        {
            min = 0;
            for(k=1;k<f;k++)
            if(cntr[k]<cntr[min])
                min=k;
            q[min]=p[i];
            cntr[min]=1;
            c++;
        }
        printf("\n");
        for(j=0;j<f;j++){
            if(q[j]==-1)
                printf("-\t");
            else
                printf("%d\t",q[j]);
        }
    }
    return c;
}

```

Output :

```

Enter no of pages : 6
Enter the reference string : 5 7 8 1 0 7
Enter no of frames : 3

```

```
First In First Out
5      -      -
5      7      -
5      7      8
1      7      8
1      0      8
1      0      7
Least Recently Used
5      -      -
5      7      -
5      7      8
1      7      8
1      0      8
1      0      7
Least Frequently Used
5      -      -
5      7      -
5      7      8
1      7      8
0      7      8
0      7      8
Page fault :
fifo = 6
lru = 6
lfu = 5
```

Result :

Program compiled and executed successfully.

Experiment No : 13

Date : 10 June 2021

Inter – Process Communication

Aim :

To write a C program to implement inter – process communication using shared memory.

Algorithm :

Write to shared memory.

1. Start
2. Create necessary variables
3. Create a shared memory segment with a unique id and with read and write permissions
4. Attach the process to the shared memory segment using "shmatt" command.
5. Display the address of the memory segment attached with the process.
6. Accept some data from the user and write it to the memory segment.
7. Display the data entered.
8. Stop.

Read from shared memory.

1. Start
2. Create necessary variables.
3. Access the shared memory with the unique key used in the first program
4. Attach the new process to the shared memory segment.
5. Display the address to the memory segment.
6. Display the data in that memory segment.
7. Stop

Program :

Write to shared memory.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
    int i, shmid;
    void *shared_memory;
    char buff[100];
    shmid = shmget((key_t)2345, 1024, 0666|IPC_CREAT);
    printf("Key of shared memory is : %d\n",shmid);
    shared_memory=shmat(shmid,NULL,0);
    printf("Process attached at : %p\n",shared_memory);
    printf("Enter some data to write to shared memory\n");
    read(0,buff,100); strcpy(shared_memory,buff);
    printf("You wrote : %s\n",(char *)shared_memory);
}
```

Read from shared memory.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/shm.h>
#include<string.h>
int main()
{
    int i, shmid;
    void *shared_memory;
    char buff[100];
    shmid = shmget((key_t)2345, 1024, 0666);
    printf("Key of shared memory is : %d\n",shmid);
    shared_memory = shmat(shmid,NULL,0);
    printf("Process attached at : %p\n",shared_memory);
    printf("Data read from shared memory is : %s",*(char *)shared_memory);
}
```

Output :

Write to shared memory.

```
► gcc writeprocess.c
► ./a.out
Key of shared memory is : 0
Process attached at : 0x7f8fcdb88000
Enter some data to write to shared memory
C Programming
You wrote : C Programming
```

Read from shared memory.

```
► gcc readprocess.c
► ./a.out
Key of shared memory is : 0
Process attached at : 0x7f650f7f5000
Data read from shared memory is : C Programming
```

Result :

Program compiled and executed successfully.

Experiment No : 14

Date : 17 June 2021

Semaphores

Aim :

To write a C program to implement semaphores.

Algorithm :

1. Start
2. Declare necessary variables.
3. Function fun1() is defined
 - 3.1 Variable x is declared
 - 3.2 Wait function is invoked.
 - 3.3 Copy the value to x
 - 3.4 Print and increment the value.
 - 3.5 Sleep function is invoked.
 - 3.6 Copy the value to shared from x.
 - 3.7 Signal is invoked.
4. Function fun2() is defined
 - 4.1 Declare variable y.
 - 4.2 Wait function is invoked.
 - 4.3 Copy the value to y
 - 4.4 Print and decrement the value.
 - 4.5 Sleep function is invoked.

- 4.6 Copy the value to shared from y
 4.7 Signal is invoked.
5. Main function is invoked.
- 5.1 Semaphore variable s is initialized as 1
 5.2 Thread1 and thread2 are created and fun1 & fun2 respectively invokes the threads.
 5.3 Thread1 and thread2 are joined
 5.4 Print shared as final variable.
6. Stop

Program :

```
#include<pthread.h>
#include<stdio.h>
#include<semaphore.h>
#include<unistd.h>
void *fun1();
void *fun2();
int shared=1;
sem_t s;
int main()
{
    sem_init(&s,0,1);
    pthread_t thread1, thread2;
    pthread_create(&thread1, NULL, fun1, NULL);
    pthread_create(&thread2, NULL, fun2, NULL);
    pthread_join(thread1, NULL);
    pthread_join(thread2,NULL);
    printf("Final value of shared is : %d\n",shared);
}
void *fun1()
{
    int x;
    sem_wait(&s);
```

```

x=shared;
printf("Thread1 reads the value as : %d\n",x);
x = x + 1 ;
printf("Local updation by Thread1 : %d\n",x);
sleep(1);
shared=x;
printf("Value of shared variable updated by Thread1 is : %d\n",shared);
sem_post(&s);
}
void *fun2()
{
    int y;
    sem_wait(&s);
    y=shared;
    printf("Thread2 reads the value as : %d\n",y);
    y = y - 1;
    printf("Local updation by Thread2 : %d\n",y);
    sleep(1);
    shared=y;
    printf("Value of shared variable updated by Thread2 is : %d\n",shared);
    sem_post(&s);
}

```

Output :

```

Thread1 reads the value as : 1
Local updation by Thread1 : 2
Value of shared variable updated by Thread1 is : 2
Thread2 reads the value as : 2
Local updation by Thread2 : 1
Value of shared variable updated by Thread2 is : 1
Final value of shared is : 1

...Program finished with exit code 0
Press ENTER to exit console.■

```

Result :

Program compiled and executed successfully.

Experiment No : 15

Date : 17 June 2021

Produce – Consumer Problem

Aim :

To write a C program to simulate producer – consumer problem using semaphores.

Algorithm :

1. Start
2. Declare necessary variables
3. Input the bufsize , buff
4. Assign empty = buff
5. Input a choice
6. If choice = 1
 - 6.1 if mutex is equal to 1
 - 6.2 Change mutex value to zero
 - 6.3 Producer() function is called
 - 6.3.1 If empty = 0, then print buffer is full
 - 6.3.2 Else
 - 6.3.2.1 Enter the number of items to be produced.
 - 6.3.2.2 If current_items + n <= bufsize

Increment current_items by n.

Decrement empty by n.
 - 6.3.2.3 Else, cannot produce.

- 6.4 Set mutex to 1.
7. If choice = 2
 - 7.1 If mutex equal to 1
 - 7.2 mutex = wait(mutex)
 - 7.3 Consume() function is called.
 - 7.3.1 If empty == buff then, print empty buffer
 - 7.3.2 Else
 - 7.3.2.1 Enter the number of items to be consumed.
 - 7.3.2.2 If n <= current_items
current_items -= n
empty += n.
 - 7.3.2.3 Else, cannot consume
 - 7.4 mutex = signal(mutex)
 8. If choice = 3
 - 8.1 Print buffer details like empty and current_items
 9. If choice = 4
 - 9.1 Exit from the program.
 10. Else print invalid choice
 11. Stop.

Program :

```
#include<stdio.h>
int empty,current_items=0,buff;
int wait(int S);
```

```

int signal(int S);
void produce();
void consume();
int main()
{
    printf("Enter the Buffer Size : ");
    scanf("%d",&buff);
    empty = buff;
    int ch=0,mutex=1;
    while (ch!=4)
    {
        printf("\n1.Produce\n2.Consume\n3.Print buffer details\n4.Exit\nEnter a
choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:{

                if (mutex==1)
                {
                    mutex = wait(mutex);
                    produce();
                    mutex = signal(mutex);
                }
                break;
            }
            case 2:{

                if (mutex==1)
                {
                    mutex = wait(mutex);
                    consume();
                    mutex = signal(mutex);
                }
                break;
            }
            case 3:{

                printf("\n.....Buffer Details.....");
                printf("\nCurrent items in buffer : %d",current_items);
                printf("\nEmpty spaces : %d",empty);
                break;
            }
        }
    }
}

```

```

        case 4:{  

            break;  

        }  

        default:{  

            printf("\nInvalid choice...");  

            break;  

        }  

    }  

    return 0;  

}  

int wait(int S)  

{  

    while(S==0);  

    S = 0;  

    return S;  

}  

int signal(int S)  

{  

    S = 1;  

    return S;  

}  

void produce()  

{  

    int n;  

    if (empty == 0){  

        printf("\nBuffer Full...");  

    }  

    else  

    {  

        printf("\nEnter number of items to be produced : ");  

        scanf("%d",&n);  

        if ((current_items+n) <= buff){  

            printf("\nProduced %d items..",n);  

            current_items += n;  

            empty -= n;  

        }  

        else  

        {  

            printf("\nCannot produce..");  

        }
    }
}

```

```

        }
    }
}

void consume()
{
    int n;
    if (empty == buff){
        printf("\nBuffer Empty...");
    }
    else
    {
        printf("\nEnter number of items to be Consumed : ");
        scanf("%d",&n);
        if (n <= current_items){
            printf("\nConsumed %d items..",n);
            current_items -= n;
            empty += n;
        }
        else
        {
            printf("\nCannot consume..");
        }
    }
}

```

Output :

```

Enter the Buffer Size : 10

1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 3

.....Buffer Details.....
Current items in buffer : 0
Empty spaces : 10
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 1

Enter number of items to be produced : 8

```

```
Produced 8 items..
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 2

Enter number of items to be Consumed : 4

Consumed 4 items..
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 1

Enter number of items to be produced : 6

Produced 6 items..
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 1

Buffer Full...
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 2

Enter number of items to be Consumed : 3

Consumed 3 items..
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 3

.....Buffer Details.....
Current items in buffer : 7
Empty spaces : 3
1.Produce
2.Consume
3.Print buffer details
4.Exit
Enter a choice : 4

...Program finished with exit code 0
Press ENTER to exit console.[]
```

Result :

Program compiled and executed successfully.

Experiment No : 16

Date : 23 September 2021

File Directory

Aim :

To write a C program for file manipulation for display of a file and directory in memory.

Algorithm :

1. Start
2. Read the directory path from the user
3. Display the entered directory path
4. User defined function print_everything_in_directory is called and pass the entered value
 - 4.1 If the entered value is null then
 - 4.1.2 Exit from the program
 - 4.2 Get pointer to DIR type by calling opendir, dir
 - 4.3 If dir == NULL
 - 4.3.1 Print "unknown directory"
 - 4.3.2 Exit from the program.
 - 4.4 Else print all the files and directories.
 - 4.5 while (dir = readdir(d)) != NULL
 - 4.5.1 Use the switch statement and print the according type and file name.
5. Stop

Program :

```

#include <stdio.h>
#include <dirent.h>
void print_everything_in_directory(const char *directory_path);
int main(void)
{
    char directory_path[256] = {0};
    printf("[?] ENTER DIRECTORY_PATH: ");
    scanf("%s", directory_path);
    printf("![!] PATH = %s\n\n", directory_path);
    print_everything_in_directory(directory_path);
    return 0;
}
void print_everything_in_directory(const char *directory_path)
{
    if(directory_path == NULL){
        fprintf(stderr, "directory_path argument is null\n");
        return;
    }
    DIR *d = opendir(directory_path);
    if (d == NULL) {
        fprintf(stderr, "![!] Unknown directory path\n");
        return;
    }
    printf("=====\\n");
    printf("\tPrinting all files and directory\\n");
    printf("=====\\n");
    printf("TYPE \t| NAME\\n");
    printf("-----\\n");
    struct dirent *dir = NULL;
    while ((dir = readdir(d)) != NULL){
        switch(dir->d_type)
        {
            case DT_REG:
                printf("FILE\t| %s\\n",dir->d_name);
                break;

            case DT_DIR:
                printf("DIRECTORY\t| %s\\n",dir->d_name);
                break;

            case DT_FIFO:
                printf("FIFO\t| %s\\n",dir->d_name);
                break;
        }
    }
}

```

```

        case DT_SOCK:
            printf("SOCKET\t| %s\n",dir->d_name);
        break;

        case DT_CHR:
            printf("CHAR DEVICE\t| %s\n",dir->d_name);
        break;

        case DT_BLK:
            printf("BLOCK DEVICE\t| %s\n",dir->d_name);
        break;

        case DT_LNK:
            printf("SYMBOLIC LINK\t| %s\n",dir->d_name);
        break;

        case DT_UNKNOWN:
            printf("UNKOWN\t| %s\n", dir->d_name);
        break;
    }
}
printf("\n");
closedir(d);
}

```

Output :

```

[?] ENTER DIRECTORY_PATH: .
[!] PATH = .

=====
          Printing all files and directory
=====

TYPE      | NAME
-----
FILE      | main.c
DIRECTORY | ..
DIRECTORY | .
FILE      | a.out

...
...Program finished with exit code 0
Press ENTER to exit console.

```

Result :

Program compiled and executed successfully.

Experiment No : 17

Date : 16 September 2021

File Allocation

Aim :

To write a C program to simulate the following file allocation strategies.

- a) Sequential
- b) Indexed
- c) Linked

Algorithm :

1. Start
2. Create a structure node with data, nextblock & link part
3. Declare variables globally.
4. Main() function is executed.
5. If the entered value is 1, then sequential() user defined function is called
 - 5.1 Declare variables
 - 5.2 Accept the value of start block and length from the user.
 - 5.3 Set flag = 0
 - 5.4 Check whether the block is free for allocation if not set flag = 1
 - 5.5 If flag == 0
 - 5.5.1 From i = start to i < start + size do,
 - 5.5.2 mem Block[i].data = 1
 - 5.5.3 Print file allocated

- 5.6 Else , print file not allocated.
6. If the entwid value is 2, then linked() user defined function is called.
- 6.1 Declare variables
 - 6.2 Accept the start block and length value from the user
 - 6.3 Set flag = 0
 - 6.4 Check whether the block is free for allocation if not set flag = 1
 - 6.5 If flag == 0
 - 6.5.1 from i = start to i < start + size do


```
memBlock[i].data = 1
create node new
new → data = 1
new → link = NULL
if fint == NULL
  fint = new
  last = new
else
  last → link = new
  last = new
```
 - 6.6 If first == NULL then file is not allocated
 - 6.7 Else print allocated.
7. If the entwid value is 3, then indexed() user defined function is called

- 7.1 Declare variables
- 7.2 Accept the length from the user
- 7.3 From $j=0$ to $j < \text{SIZE}$ do
 - 7.3.1 If $\text{memBlock}[j].data == 0$
 - 7.3.1.1 If $\text{flag} == 0$
 - $st = j$
 - $\text{flag} = 1$
 - 7.3.1.2 Increment allocated
 - 7.3.1.3 Set $\text{memBlock}[j].data = 1$
 - 7.3.1.4 $\text{memBlock}[\text{prevBlock}].nextBlock = j$
 - 7.3.1.5 $\text{prevBlock} = j$
 - 7.3.2 If allocated == size then,
break
 - 7.4 Set $c = st$
 - 7.5 From $c=0$ to $c < \text{size}-1$ do,
 - 7.5.1 $c = \text{memBlock}[c].nextBlock$
 - 7.5.2 Print output
 8. If the entered value is 4, then exit from the program
 9. Stop

Program :

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 50
struct node
{
```

```

int data;
int nextBlock;
struct node *link;
}*first=NULL,*last,*temp,*new;
struct node memBlock[50];
int flag,prevBlock=-1;
int check(int start, int size);
void sequential();
void linked();
void indexed();
void main()
{
    int choice;
    printf("\n1.Sequential Allocation\n2.Linked Allocation\n3.Indexed
Allocation\n4.Exit\n");
    printf("Enter a choice : ");
    scanf("%d", &choice);
    switch (choice)
    {
        case 1:{
            sequential();
            main();
            break;
        }
        case 2:{ 
            linked();
            main();
            break;
        }
        case 3:{ 
            indexed();
            main();
            break;
        }
        case 4:{ 
            break;
        }
        default:{ 
            printf("\nInvalid choice");
            main();
        }
    }
}
int check(int start, int size)
{
    int i;

```

```

        for (i = start; i < start+size; i++)
    {
        if (memBlock[i].data == 1)
        {
            return 1;
            break;
        }
    }
    return 0;
}
void sequential()
{
    int i,start,size;
    printf("\nEnter the start block : ");
    scanf("%d",&start);
    printf("\nEnter the length of the file : ");
    scanf("%d",&size);
    flag = 0;
    flag = check(start, size);
    if (flag==0)
    {
        for (i = start; i < start+size; i++)
        {
            memBlock[i].data = 1;
            printf("%d ---> %d\n",i,memBlock[i].data);
        }
        printf("\nFile is allocated...\n");
    }
    else
    {
        printf("\nFile cannot be allocated");
    }
}
void linked()
{
    int i,start,size;
    printf("\nEnter the start block : ");
    scanf("%d",&start);
    printf("\nEnter the length of the file : ");
    scanf("%d",&size);
    flag=0;
    flag = check(start, size);
    if (flag == 0)
    {
        for(i=start;i<start+size;i++)
    {

```

```

        memBlock[i].data = 1;
        new = (struct node *)malloc(sizeof(struct node));
        new -> data = 1;
        new -> link = NULL;
        if(first == NULL)
        {
            first = new;
            last = new;
        }
        else
        {
            last -> link = new;
            last = new;
        }
    }
    if(first == NULL)
    {
        printf("\nFile cannot be allocated\n");
    }
    else
    {
        temp = first;
        while(temp != NULL)
        {
            for(i=start;i<start+size;i++)
            {
                printf("%d ---> %d\n",i,temp->data);
                temp = temp->link;
            }
        }
        printf("\nFile is allocated...\n");
    }
}
void indexed()
{
    int size,i,st,flag=0,j=0,allocated=0;
    printf("\nEnter the length of file : ");
    scanf("%d",&size);
    for (j = 0; j < SIZE; j++)
    {
        if (memBlock[j].data == 0)
        {
            if (flag==0)
            {
                st = j;

```

```

        flag=1;
    }
    allocated++;
    memBlock[j].data = 1;
    memBlock[prevBlock].nextBlock = j;
    prevBlock = j;
}
if (allocated==size)
{
    break;
}
int c = st;
printf("\n%d--->%d",st,1);
for (i = 0; i < size-1; i++)
{
    c = memBlock[c].nextBlock;
    printf("\n%d--->%d",c,1);
}
}

```

Output :

```

1.Sequential Allocation
2.Linked Allocation
3.Indexed Allocation
4.Exit
Enter a choice : 1

Enter the start block : 3

Enter the length of the file : 8
3 ---> 1
4 ---> 1
5 ---> 1
6 ---> 1
7 ---> 1
8 ---> 1
9 ---> 1
10 ---> 1

```

File is allocated...

```

1.Sequential Allocation
2.Linked Allocation
3.Indexed Allocation
4.Exit
Enter a choice : 2

```

```
Enter the start block : 9
Enter the length of the file : 4
File cannot be allocated
1.Sequential Allocation
2.Linked Allocation
3.Indexed Allocation
4.Exit
Enter a choice : 2

Enter the start block : 11
Enter the length of the file : 8
11 ---> 1
12 ---> 1
13 ---> 1
14 ---> 1
15 ---> 1
16 ---> 1
17 ---> 1
18 ---> 1

File is allocated...
1.Sequential Allocation
2.Linked Allocation
3.Indexed Allocation
4.Exit
Enter a choice : 3

Enter the length of file : 6
0----->1
1----->1
2----->1
19----->1
20----->1
21----->1
1.Sequential Allocation
2.Linked Allocation
3.Indexed Allocation
4.Exit
Enter a choice : 4
PS D:\Programming\C\S4> █
```

Result :

Program compiled and executed successfully.

A Course project on

Inter – process Communication And Process Management

Submitted By Group 01 of S4 CSE

- 1. Aadhithyanarayanan V A** (ASI19CS001)
- 2. Aakash A Nair** (ASI19CS002)
- 3. Abhay Krishnan M N** (ASI19CS003)
- 4. Abhay Shanker K** (ASI19CS004)
- 5. Abhijith E S** (ASI19CS005)



October 2021

Department of Computer Science and Engineering

**ADI SHANKARA INSTITUTE OF ENGINEERING & TECHNOLOGY
KALADY, KERALA**

AIM :

To write a C program for inter – process communication and process management.

ALGORITHM :

1. Algorithm for creating processes and attaching it to shared memory.
 1. Start
 2. Declare variables *all,*unall,sizeall,sizeunall, shmid
 3. Declare functions void creation(),void allocation(),void unallocated(),void menu()
 4. Main function is called
 - a. Shared memory is created
 - b. Array allocated and unallocated is attached to the shared memory
 5. If the choice entered by the user is 1, then
 - a. New process is created
 - b. If resource is available, it will be allocated and the process is moved to the allocated array
 - c. Else the process will be moved to the unallocated array
 6. If the choice entered by the user is 3, then
 - a. Both the arrays are detached form the shared memory.
 - b. Exit form the program
 7. Stop

2. Algorithm for monitoring the processes in the shared memory.
 1. Start
 2. Declare shmid,*all,*unall
 3. Declare functions void status()
 4. Main function is invoked
 - a. Shared memory previously created is opened
 - b. Status() function is called
 - i. If the choice entered is 1, then
 1. Print the processes in execution from allocated array
 - ii. If choice enter is 2, then
 1. Print the processes in waiting list from the unallocated array
 - iii. If coice entered is 3, then
 1. Detach the shared memory and arrays
 2. Exit from the program
 5. Stop

3. Algorithm for the termination of processes from shared memory.

1. Start
2. Declare function terminate and pointers all and unall
3. Create shared memory id
4. Attach shared memory to pointers all and unall
5. Function call terminate
 - a. If choice==1
 - i. If unall[i]!=0
 1. Assign id=unall[i]
 2. unall[i]=0
 - ii. Place waiting process to allocation list
 1. all[i]=id
 - iii. Recursive call ,terminate()
 - b. If choice==2
 - i. Detach all and unall from shared memory
6. stop

PROGRAM

1. Program for creating processes and attaching it to shared memory.

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include<sys/shm.h>
pid_t pid;
int *all,*unall,sizeall,sizeunall,shmid;
void creation();
void allocation();
void unallocated();
void menu();
int main()
{
    int i;
    key_t key =9681;
    sizeall = sizeof(*all)*3;
    sizeunall = sizeof(*all)*20;
    shmid = shmget((key_t)2345, 1024, 0666|IPC_CREAT);
    all = (int *)shmat(shmid, 0, 0);
    unall = (int *)shmat(shmid, 0, 0);
    for(i=1; i<=20; i++)
    {
        if(i <= 3)
        {
            all[i] = 0;
        }
        unall[i] = 0;
    }
    menu();
    return 0;
}
```

```

    }
void menu()
{
    int choice;
    printf("\n\nSelect an option : ");
    printf("\n1. Create a process and allocate resource");
    printf("\n2. Exit from the program");
    printf("\nEnter your choice : ");
    scanf("%d",&choice);
    if(choice == 1){
        creation();
    }
    else{
        shmdt((void *) all);
        shmdt((void *) unall);
        exit(0);
    }
}
void creation()
{
    pid = fork();
    if(pid > 0)
    {
        printf("Process Created with id : %2d",pid);
        allocation();
    }
}
void allocation()
{
    int i;
    for(i=1;i<=3;i++)
    {
        if(all[i] == 0)
        {
            all[i] = pid;
            printf("\nProcess attached to sharedmemory and resource %d
allocated",i);
            menu();
            break;
        }
        if(all[3] != 0)
        {
            unallocated();
        }
    }
}
void unallocated()
{
    int j;
    for(j=0;j<=20;j++)
    {
        if(unall[j] == 0 )
        {

```

```

        unall[j] = pid;
        printf("\nProcess attached to sharedmemory and waiting for r
esource ");
        menu();
        break;
    }
}
}

```

2. Program for monitoring the status of processes in shared memory.

```

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include<sys/shm.h>
int shmid,*all,*unall;
void status();
int main()
{
    key_t key = 2345;
    shmid = shmget(key, 50*sizeof(int), IPC_EXCL);
    all = shmat(shmid, 0, SHM_RDONLY);
    unall = shmat(shmid, 0, SHM_RDONLY);
    status();
    return 0;
}
void status()
{
    int choice,i,j;
    printf("\n\n1.Current process in execution");
    printf("\n2.Process in waiting list");
    printf("\n3.Exit");
    printf("\nEnter your choice : ");
    scanf("%d",&choice);
    if(choice == 1){
        printf("\nProcess Id\t----->\tOrder\n\n");
        for(i=1;i<=3;i++)
        {
            printf("\t%d\t----->\t%d\n",all[i],i);
        }
        status();
    }
    if(choice == 2){
        printf("\nProcess Id\t----->\tOrder\n\n");
        for(j=4;j<=24;j++)
        {
            if(unall[j] != 0)
            {
                printf("\t%d\t----->\t%d\n",unall[j],j-3);
            }
        }
        status();
    }
}

```

```

        }
        if(choice == 3){
            shmdt((void *) all);
            shmdt((void *) unall);
            exit(0);
        }
    }
}

```

3. Program for termination of processes in shared memory.

```

#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include <sys/types.h>
#include<sys/shm.h>
int shmid,*all,*unall,i,j;
void terminate();
int main()
{
    int shmid;
    key_t key = 2345;
    shmid = shmget(key, 50*sizeof(int), IPC_EXCL);
    all = (int *)shmat(shmid, 0, 0);
    unall = (int *)shmat(shmid, 0, 0);
    terminate();
    return 0;
}
void terminate()
{
    int cho,id;
    printf("\n\n1.Terminate finished process");
    printf("\n2.Exit");
    printf("\nEnter your choice : ");
    scanf("%d",&cho);
    if(cho == 1){
        id=unall[4];
        for(int i=5;i<=24;i++){
            unall[i-1]=unall[i];
        }
        all[1] = id;
        for(int j=1;j<=3;j++){
            printf("%d\n",all[j]);
        }
        terminate();
    }
    if(cho == 2)
    {
        shmdt((void *) all);
        shmdt((void *) unall);
        exit(0);
    }
}

```

OUTPUT/SAMPLE SCREEN SHOTS

1. Program for creating processes and attaching it to shared memory.

```
Select an option :
1. Create a process and allocate resource
2. Exit from the program
Enter your choice : 1
Process Created with id : 59
Process attached to sharedmemory and resource 1 allocated

Select an option :
1. Create a process and allocate resource
2. Exit from the program
Enter your choice : 1
Process Created with id : 60
Process attached to sharedmemory and resource 2 allocated

Select an option :
1. Create a process and allocate resource
2. Exit from the program
Enter your choice : 1
Process Created with id : 61
Process attached to sharedmemory and resource 3 allocated

Select an option :
1. Create a process and allocate resource
2. Exit from the program
Enter your choice : 1
Process Created with id : 62
Process attached to sharedmemory and waiting for resource

Select an option :
1. Create a process and allocate resource
2. Exit from the program
Enter your choice : 2
```

2. Program for monitoring the status of processes in shared memory.

```
1.Current process in execution
2.Process in waiting list
3.Exit
Enter your choice : 1

Process Id -----> Order

59      -----> 1
60      -----> 2
61      -----> 3

1.Current process in execution
2.Process in waiting list
3.Exit
Enter your choice : 2

Process Id -----> Order

62      -----> 1

1.Current process in execution
2.Process in waiting list
3.Exit
Enter your choice : 3
~/Course-Project$ █
```

3. Program for termination of processes in shared memory.

```
1.terminate finished process
2.Exit
Enter your choice : 1
171
169
170

1.terminate finished process
2.Exit
Enter your choice : 1
172
169
170

1.terminate finished process
2.Exit
Enter your choice : 
```

RESULT :

Program compiled and executed successfully.

REFERENCES :

<https://www.geeksforgeeks.org/inter-process-communication-ipc/>
<https://stackoverflow.com/questions/21227270/>
<https://www.geeksforgeeks.org/introduction-of-process-management/>
<https://www.geeksforgeeks.org/ipc-technique-pipes/>