RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

IT19441 – OPERATING SYSTEM LABORATORY

RECORD NOTEBOOK

NAME	:	_____
YEAR/BRANCH/SECTION	:	_____
ROLL NUMBER	:	_____
SEMESTER	:	_____
ACADEMIC YEAR	:	_____

INDEX

EXP.NO	DATE	NAME OF THE EXPERIMENT	PAGE NUMBER	SIGNATURE
1.	23.02.2022	Study of Linux Commands		
2.	09.03.2022	Shell Scripting		
3.	16.03.2022	UNIX System Call		
		Implementation of I/O system calls using one file		
		Implementation of I/O system calls using two files		
4.	23.03.2022	Process Creation – Child Process		
		Process Creation – Orphan Process		
		Process Creation – Zombie Process		
5.	30.03.2022	Interprocess Communication Using Shared Memory		
6.	06.04.2022	Interprocess Communication using Named Pipes		
7.	13.04.2022	First Come First Serve Scheduling		
		Shortest Job First Scheduling		
8.	20.04.2022	Producer-Consumer Problem using Semaphores		
9.	27.04.2022	Implementation of Bankers Algorithm for Deadlock Avoidance		

EXP.NO	DATE	NAME OF THE EXPERIMENT	PAGE NUMBER	SIGNATURE
10.	04.05.2022	Contiguous Memory Allocation Strategies – First fit, Best Fit, Worst Fit		
11.	12.05.2022	Implement file allocation strategies – Sequential		
12.	12.05.2022	Implementation of page replacement algorithms – FIFO		

Exp. No: 1	STUDY OF LINUX COMMANDS

Aim:

To study various LINUX commands in detail.

Commands:

General commands:

NAME	SYNTAX	DESCRIPTION
Who	$who	Displays users who are currently logged on
Who am i	$who am i	Displays our own login terminal and other details
finger [user]	$ finger it3	Displays system biography on user `[user]'.
Date	$date	Displays the date which is stored in a particular format
Calendar	$cal<month><year>	Displays calendar of specified month and year
Clear	$clear	Clears the screen and displays the new screen
Uname	$uname	Displays the OS which is used.
	$uname –a	Displays all machine details
	$uname –s	Displays the OS name
	$uname –v	Displays the version of OS
	$uname –r	Displays the release of OS
	$uname –n	Displays the name of network mode
	$uname –m	Displays the type of OS
Binary calculator	$bc	Used for calculation
Identifier	$id	Displays userid and groupid
Present working Directory	$pwd	Displays the current working directory
Echo	$echo<text>	Displays the text given by the user
Man	$man <command>	Displays the details of the specified command
Touch	$touch <filename>	Creates a file
Tty	$tty	Displays the terminal number of the system

Listing Options

Pattern Searching Command

SYNTAX	DESCRIPTION
$ls	Lists all files in the present working directory

$ls –a	Displays all hidden files of the user
$ls –c	Lists all subdirectories of the file in a column-wise fashion
$ls –d	Displays the root directory of the present directory
$ls –r	Reverses the order in which files and subdirectories are displayed
$ls –R	Lists the files and subdirectories in hierarchical order
$ls –t	Displays the files in the order of modification
$ls –p	Display the files and subdirectories with a slash mark
$ls –i	Displays the node number of each file
$ls –l	Lists the permission given to each file
$grep <pattern> <filename>	Displays the line in which the given pattern is seen

File Manipulation Commands

NAME	SYNTAX	DESCRIPTION
Cat>>	$cat>><filename>	Edit contents of existing file
Cat	$cat <filename>	View the contents of the file
Cat	$cat –n filename	Displays the file with line numbers
More	$more <filename>	Displays the file page by page
More+	$more+10 <filename>	Files will be displayed from the 10th page onwards
Wc	$wc	Counts the number of words, characters, and lines for a given file
Wc –l	$wc –l	Displays the number of lines
Wc –w	$wc –w	Displays the number of words
Wc –c	$wc –c	Displays only the number of characters
Cp	$cp <filename1><filename2>	Copy a file
Mv	$mv<filename1><filename2>	Move a file from a directory to another directory
Rm	rm <filename>	removes a file
	rm –i filename	ask you for confirmation before actually deleting anything
diff	diff <filename1> <filename2 >	compares files, and shows where they differ
Ln –s	$ln –s <source filename><new filename>	Creates a soft link between files. Here contents are copied to a new file, but both files' memory addresses are the same.[After creating the link (i.e.,) naming a file with 2 different names, if the original file is deleted, the newly created file is automatically deleted]

Ln	$ln <source filename><new filename>	Creates a hard link between files. Here content is copied to a new file which is saved in a new address. . [After creating the link (i.e.,) naming a file with 2 different names, if the original file is deleted, the newly created file is not deleted]

Filter Commands

NAME	SYNTAX	DESCRIPTION
Head	$head <filename	Displays the top 10 lines of the file
Head - 5	$head -5 <filename>	Displays the top 5 lines of the file
Tail	$tail <filename>	Displays the last 10 lines of the file
Tail – t	$tail -5 <filename	Displays the last 5 lines of the file
Paste	$paste <filename1><filename2>	Paste two files in vertical manner
Sort	$sort <filename>	Sorts the contents of the file in alphabetical order
Sort –r	$sort –r<filename>	Sorts the contents of the file in reversed alphabetical order

Directory Commands

NAME	SYNTAX	DESCRIPTION
Present working Directory	$pwd	Displays the current working directory
Make directory	$ mkdir subdir	**mkdir** creates a new subdirectory inside of the directory where you are currently working
Change directory	$ cd Misc	**cd** moves you to another directory.
		To change back to your home directory: To change back to your home directory:
Remove directory	$rmdir filename	To remove a directory. If the directory contains a subdirectory or files, remove it first and then remove it.

Exp. No: 2a	SHELL SCRIPTING
	ARITHMETIC OPERATORS

Aim:

To write a UNIX shell program to find the arithmetic operations for the given numbers.

Algorithm:

Step 1: Start the program.

Step 2: Read two values.

Step 3: Do the arithmetic operation (i.e. addition, subtraction, multiplication, division) for the given values.

Step 4: Print the arithmetic operation values.

Step 5: Stop the program.

Program:

```
echo "Enter x value:"
 read x
echo "Enter y value:"
 read y
echo "Addition"
let "z = $(( x + y ))"
echo "z= $z"
echo "Subtraction"
 let "z = $((x - y ))"
 echo "z= $z"
 echo "Multiplication"
 let "z = $(( x * y ))"
 echo "z = $z"
 echo "Division"
  let "z = $(( x / y ))"
  echo "z = $z"
```

OUTPUT:

```
hari@HARIPRIYA:~$ ./arithmetic.sh
Enter x value:
2
Enter y value:
3
Addition
z= 5
Substraction
z= -1
Multiplication
z = 6
Division
z = 0
```

Exp. No: 2b	SHELL SCRIPTING
	GREATEST OF TWO NUMBERS

Aim:

To write a UNIX shell program to find the greatest of two numbers.

Algorithm:

Step 1: Start the program.

Step 2: Read two values.

Step 3: check the two values which is greater.

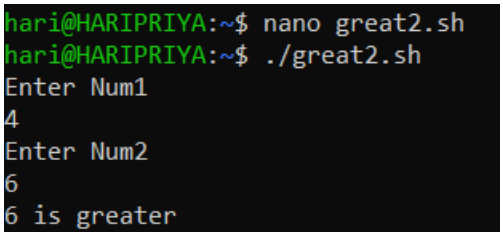
Step 4: Print the greater value.

Step 5: Stop the program.

Program:

```
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
if [ $num1 -gt $num2 ]
then
   echo "$num1 is greater"
else
  echo "$num2 is greater"
fi
```

Output:



Exp. No: 2c	SHELL SCRIPTING
	FINDING ODD OR EVEN

Aim:

To write a UNIX shell program to find the sum of two numbers.

Algorithm:

Step 1: Start the program.

Step 2: Read the values.

Step 3: Check whether the number is odd or even.

Step 4: Print the value.

Step 5: Stop the program.

Program:

```
echo -n "Enter a number:"
read n
echo -n "RESULT: "
if [ `expr $n % 2` == 0 ]
  then
      echo "$n is even"
else
     echo "$n is Odd"
fi
```

Output:

```
hari@HARIPRIYA:~$ ./odd.sh
Enter a number:4
RESULT: 4 is even
```

Exp. No: 2d	SHELL SCRIPTING
	GREATEST OF 3 NUMBERS

Aim:

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

Algorithm:

Step 1: Start the program.

Step 2: Read two values.

Step 3: check the three values which is greater.

Step 4: Print the greater value.

Step 5: Stop the program

Program:

```
echo "Enter Num1"
read num1
echo "Enter Num2"
read num2
echo "Enter Num3"
read num3
if [ $num1 -gt $num2 ] && [ $num1 -gt $num3 ]
then
    echo "$num1 is greater"
elif [ $num2 -gt $num1 ] && [ $num2 -gt $num3 ]
then
    echo "$num2 is greater"
else
     echo "$num3 is greater"
fi
```

Output:

```
hari@HARIPRIYA:~$ ./great3.sh
Enter Num1
4
Enter Num2
5
Enter Num3
6
6 is greater
```

Exp. No: 2e	SHELL SCRIPTING
	SIMPLE CALCULATOR

Aim:

To write a UNIX shell program for a simple calculator.

Algorithm:

Step 1: Start the program.

Step 2: Read two values.

Step 3: Do the arithmetic operation (i.e. addition, subtraction, multiplication, division) for the given values.

Step 4: Print the values obtained by the program.

Step 5: Stop the program

Program:

```
echo "Enter Two numbers : "
read a
read b
echo "Enter Choice :"
echo "1. Addition"
echo "2. Subtraction"
echo "3. Multiplication"
echo "4. Division"
read ch
case $ch in
1)res=`echo $a + $b | bc`
;;
2)res=`echo $a - $b | bc`
;;
3)res=`echo $a \* $b | bc`
;;
4)res=`echo "scale=2; $a / $b" | bc`
;;
esac
echo "Result : $res"
```

Output:

```
hari@HARIPRIYA:~$ ./calculator.sh
Enter Two numbers :
4
5
Enter Choice :
1. Addition
2. Subtraction
3. Multiplication
4. Division
1
Result : 9
```

Exp. No: 2f	SHELL SCRIPTING
	VOWEL OR CONSTANT

Aim:

To write a UNIX shell program to find the Given character is vowel or constant.

Algorithm:

Step 1: Start the program.

Step 2: Read the values.

Step 3: check the character is constant or vowel.

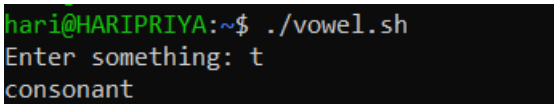
Step 4: Print the value.

Step 5: Stop the program.

Program:

```
read -p "Enter something: " char
if [[ "$char" == *[AEIOUaeiou]* ]]; then
  echo "vowel"
else
   echo "consonant"
fi
```

Output:



Exp. No: 2g	SHELL SCRIPTING
	SUM OF N NATURAL NUMBERS

Aim:

To write a UNIX shell program to find the sum of N natural numbers.

Algorithm:

Step 1: Start the program.

Step 2: Read the values.

Step 3: Add the n number of values.

Step 4: Print the value.

Step 5: Stop the program.

Program:

```
echo -n "Enter nth number's value:"
read digit
t=1
total=0
while test $t -le $digit
do
    total=`expr $total + $t`
    t=`expr $t + 1`
done
echo "SUM OF $DIGIT: $total "
```

Output:

```
hari@HARIPRIYA:~$ ./nnatural.sh
Enter nth number's value:5
SUM OF : 15
```

Exp. No: 2h	SHELL SCRIPTING
	FACTORIAL OF A GIVEN NUMBER

Aim:

To write a UNIX shell program to find the factorial of a given number.

Algorithm:

Step 1: Start the program.

Step 2: Read the values.

Step 3: Do the factorial operation of the given number .

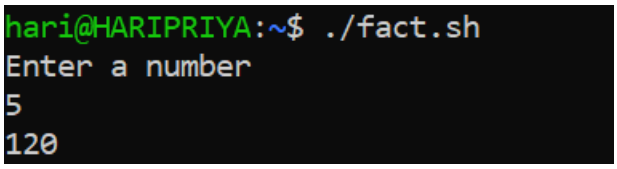
Step 4: Print the value.

Step 5: Stop the program.

Program:

```
echo "Enter a number"
read num
 fact=1
 while [ $num -gt 1 ]
do
   fact=$((fact * num))  #fact = fact * num
   num=$((num - 1))      #num = num - 1
done
echo $fact
```

Output:



Exp. No: 2i	SHELL SCRIPTING
	FIBONACCI SERIES

Aim:

To write a UNIX shell program to find the Fibonacci series of a given number.

Algorithm:

Step 1: Start the program.

Step 2: Read the values.

Step 3: Do the Fibonacci operation of the given number .

Step 4: Print the value.

Step 5: Stop the program.

Program:

```
echo "Enter a number:"
read N
a=0
b=1
echo "The Fibonacci series is : "
for (( i=0; i<N; i++ ))
do
   echo -n "$a "
   fn=$((a + b))
   a=$b
   b=$fn
done
echo "done"
```

Output:

```
hari@HARIPRIYA:~$ ./fibonacci.sh
Enter a number:
6
The Fibonacci series is :
0 1 1 2 3 5 done
```

Exp. No: 3a	UNIX SYSTEM CALLS

Aim:

To write a C program to implement the basic UNIX System Calls.

Algorithm:

Step1: start the program.

Step2: Execute the fork () system call and then stop.

Step3: Execute the exec () system call and then stop.

Step4: Execute the wait () and sleep () system calls and then stop.

Step5: Execute the stat () system call.

Step6: stop the program.

Program 1:

fork() System Call

```
#include<stdio.h>
#include<unistd.h>
int main(){
    printf("Unix system call");
    fork();
    fork();
    fork();
    printf("fork system call\n");
    printf("process id = %d\n",getpid());}
```

Output:

```
hari@HARIPRIYA:~$ gcc unix1.c
hari@HARIPRIYA:~$ ./a.out
Unix system callfork system call
process id = 201
Unix system callfork system call
process id = 205
Unix system callfork system call
process id = 202
Unix system callfork system call
Unix system callfork system call
hari@HARIPRIYA:~$ process id = 207
Unix system callfork system call
process id = 203
process id = 206
Unix system callfork system call
Unix system callfork system call
process id = 208
process id = 204
```

2)fork() System Call

```
#include<stdio.h>
#include<unistd.h>
int main()
{
pid_t p;
p=fork();
if(p<0)
    printf("Error in creating process\n.");
else if(p==0)
    printf("Child is executing  process id = %d, parent process is %d\n", getpid(),getppid());
else
    printf("Parent is executing = %d\n",getppid());
}
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
Parent is executing = 9
Child is executing  process id = 219, parent process is 218
```

Exec() system call

Program:

Add.c

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    printf("process of new process ....PID =%d\n",getpid());
    printf("Addition = %d\n",20+10);
}
```

Output:

```
hari@HARIPRIYA:~$ gcc unix3.c
hari@HARIPRIYA:~$ ./a.out
process of new process ....PID =225
Addition = 30
```

Exec.c

```
#include<stdio.h>
#include<unistd.h>
int main()
{
        printf("Current program in execution ... PID = %d\n",getpid());
        char *a[]={"./add",NULL};
        execv(a[0],a);
        printf("Back to current process");
}
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
Current program in execution ... PID = 233
Process of new process .. PID = 233
Addition = 16
```

Failure.c

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(int argc,char * argv[])
{
    execvp(argv[1],&argv[1]);
    perror("exec failure");
    exit(1);
}
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
Segmentation fault
```

Wait () system call

Program:

```
#include<stdio.h>
#include<unistd.h>
int main(){
pid_t p;
int a,b;
```

```
p=fork();
if(p<0)
    printf("ERROR");
else if (p==0)
{
    printf("child process PID = %d\n",getpid());
    printf("Enter a & b value for addition \n");
    scanf("%d %d ",&a,&b);
    printf("call by child = %d\n",a+b);
}
else
{
    printf("Enter a & b for mul \n");
    scanf("%d %d",&a,&b);
    printf("Call by parent = %d\n",a*b);
    printf("parent process PID = %d\n",getpid());
}
}
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
Enter a & b for mul
child process PID = 261
Enter a & b value for addition
5 6
Call by parent = 30
parent process PID = 260
hari@HARIPRIYA:~$ call by child = 1665010473
```

Sleep() system call

Program:

```
#include<stdio.h>
#include<unistd.h>
int main()
{
   pid_t p;
    int a,b;
    p=fork();
    if(p<0)
```

```
 printf("ERROR");
 else if (p==0)
 {
        printf("child process PID = %d\n",getpid());
        printf("Enter a & b value for addition \n");
         scanf("%d %d",&a,&b);
          printf("call by child = %d\n",a+b);
   }
else
 {
        wait();
        printf("Enter a & b for mul \n");
        scanf("%d %d",&a,&b);
         printf("call by parent = %d\n",a*b);
        printf("parent process PID = %d\n",getpid());
    }
 }
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
child process PID = 272
Enter a & b value for addition
4 5
call by child = 9
Enter a & b for mul
4 5
call by parent = 20
parent process PID = 271
```

Stat() system call

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/stat.h>
#include<sys/types.h>
int main()
{
   struct stat buf;
   stat("unix8.c",&buf);
     printf("FILE MODE = %o\n",buf.st_mode);
```

```
 printf("FILE SIZE = %ld\n",buf.st_size);
 printf("FILE BLOCK SIZE = %ld\n",buf.st_blksize);
 printf("PROCESS ID =%d\n",buf.st_gid);
 printf("NO OF BLOCKS ALLOCATED = %ld\n",buf.st_blocks);
 printf("NO OF HARD LINK = %u\n",(unsigned int)buf.st_nlink);
printf("File permissions User\n");
printf((buf.st_mode & S_IRUSR)?"r":"-");
printf((buf.st_mode & S_IWUSR)?"w":"-");
printf((buf.st_mode & S_IXUSR)?"x":"-");
printf("\nFile permissions Group\n");
printf((buf.st_mode & S_IRGRP)?"r":"-");
printf((buf.st_mode & S_IWGRP)?"x":"-");
 printf((buf.st_mode & S_IXGRP)?"x":"-");
 printf("\nFile Permissions Other\n");
 printf((buf.st_mode & S_IROTH)?"r":"-");
  printf((buf.st_mode & S_IWOTH)?"w":"-");
 printf("\n");
return 0;
}
```

Output:

```
hari@HARIPRIYA:~$ gcc unix8.c
hari@HARIPRIYA:~$ ./a.out
FILE MODE = 100777
FILE SIZE = 884
FILE BLOCK SIZE = 4096
PROCESS ID =1000
NO OF BLOCKS ALLOCATED = 8
NO OF HARD LINK = 1
File permissions User
rwx
File permissions Group
rxx
File Permissions Other
rw
```

Exp. No:3b	IMPLEMENTATION OF I/O SYSTEM CALLS USING ONE FILES

Aim:

To write a C program to illustrate the concept of the I/O system call using one file.

Algorithm:

Step1: start the program.

Step2: Create the empty file(i.e.sample.txt).

Step3: Run the program and then enter the text for the program input.

Step4: Now you open the empty file and now you can see the text entered as the input for the Program in the empty file.

Step5: Stop the program.

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<fcntl.h>
#include<string.h>
int main()
{
      char str[100],ch[1];
       int c=0,i=0,l,fd;
       fd=open("sample.txt",O_RDWR,O_APPEND);
        if(fd==-1)
        {
         printf(" Error!\n");
         exit(1);
         }
        while(c==0)
      {
       fflush(stdin);
       printf("\n Enter the text:");
       scanf("%s",str);
      scanf(str,"\n");
```

```
        write(fd,str,strlen(str));
         fflush(stdin);
        printf("Press 0 to Continue");
        scanf("%d",&c);
        }
        close(fd);
        fd = open("sample.txt",O_RDONLY);
        printf("\nContent of file:");
        while(read(fd,ch,1)>0);
        printf("%c",ch[i]);
        close(fd);
    }
```

Output:

```
hari@HARIPRIYA:~$ gcc lab4.1.c
hari@HARIPRIYA:~$ ./a.out

 Enter the text:hari prasath
Press 0 to Continue
 Enter the text:Press 0 to Continue
the great karikalan
```

Sample.txt

```
hari@HARIPRIYA: ~
  GNU nano 4.8
hariprasaththegreatkarikalan
```

Exp. No:3c	IMPLEMENTATION OF I/O SYSTEM CALLS USING TWO FILES

Aim:

To write a C program to illustrate the concept of the I/O system call using two file.

Algorithm:

Step1: start the program.

Step2: Create the two empty files (i.e.samp.txt and New.txt).

Step3: Run the program and then enter the text for the program input.

Step4: Now you open the empty file and now you can see the text entered as the input for the Program in the empty file.

Step5: Now open the New.txt file you can see the text entered in the samp.txt in the New.txt file.

Step6: Stop the program.

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<string.h>
#include<stdlib.h>
void main()
 {
   int fd1,fd2;
  char ch[1];
  fd1=open("samp.txt",O_RDONLY);
  fd2=open("New.txt",O_RDWR|O_APPEND);
  while(read(fd1,ch,1)>0)
    {
      write(fd2,ch,1);
     }
   close(fd2);
   close(fd1);
 fd2=open("New.txt",O_RDONLY);
```

```
  while(read(fd2,ch,1)>0)
 {
      printf("%c",ch[0]);
  }
       close(fd2);
}
```

Output:

```
hari@HARIPRIYA:~$ ./a.out

HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
```

Samp.txt

```
hari@HARIPRIYA: ~
  GNU nano 4.8
HARI PRASATH
AIML
REC
```

New.txt

```
hari@HARIPRIYA: ~
  GNU nano 4.8

HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
HARI PRASATH
AIML
REC
```

Exp. No: 4a	PROCESS CREATION
	CHILD PROCESS

Aim:

To write a C program to illustrate the concept of executing the child process using fork ().

Algorithm:

Step1: start the program.

Step2: Assign pid = fork ().

Step3: Run the program.

Step4: If (pid == 0)then the parent id and parent of parent id gets printed ,else the parent id gets Printed.

Step5: Stop the program.

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
  int pid;
  pid=fork();
  if(pid==0)
  printf("child process is in execution .....processID=%u and parent PID=%u\n",
        getpid(),getppid());
  else
   printf("parent process is in execution.....processID=%u\n",getpid());
   return 0;
   }
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
parent process is in execution.....processID=313
child process is in execution .....processID=314 and parent PID =313
```

Exp. No:4b	PROCESS CREATION
	ORPHAN PROCESS

Aim:

To write a program to illustrate the concept of the orphan process using a system call.

Algorithm:

Step1: start the program.

Step2: Assign pid = fork ().

Step3: Run the program.

Step4: If (pid > 0)then the parent id gets printed , else (pid == 0) parent of parent id and parent id gets Printed.

Step5: Stop the program.

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main(){
 int pid = fork();
 if(pid>0) {
 printf("parent process ID :%d\n",getpid());
 printf("\n Process Terminated\n"); }
 else if (pid==0)
{ sleep(15);
 printf("child process .....ID:%d\n",getpid());
 printf("Parent-ID :%d\n\n",getppid()); }
return 0;
}
```

Output:

```
hari@HARIPRIYA:~$ nano lab4.5.c
hari@HARIPRIYA:~$ gcc lab4.5.c
hari@HARIPRIYA:~$ ./a.out
parent process ID :328

 Process Terminated
hari@HARIPRIYA:~$ child process .....ID:329
Parent-ID :8
```

Exp. No:4c	PROCESS CREATION
	ZOMBIE PROCESS

Aim:

To write a program to illustrate the concept of the orphan process using a system call.

Algorithm:

Step1: start the program.

Step2: Assign pid = fork ().

Step3: Run the program.

Step4: If (pid == 0)then the parent id and parent of parent id gets printed ,else the parent id gets Printed.

Step5: Stop the program.

Program:

```
#include<stdio.h>
#include<unistd.h>
#include<sys/wait.h>
#include<sys/types.h>
#include<unistd.h>
int main(){
        int i;
        int pid = fork();
       if (pid == 0)
           for (i=0; i<3; i++)
                printf("I am Child....PID = %u my parent's PID = %u\n",getpid(),getppid());
       else {
                printf("I am Parent......PID = %u\n",getpid());
                 sleep(15);
               printf("\n parent terminated\n");    }
 }
```

Output:

```
hari@HARIPRIYA:~$ ./a.out
I am Parent......PID = 357
I am Child....PID = 358 my parent's PID = 357
I am Child....PID = 358 my parent's PID = 357
I am Child....PID = 358 my parent's PID = 357

 parent terminated
```

Exp. No:5	INTERPROCESS COMMUNICATION USING SHARED MEMORY

Aim:

To write a program to illustrate the concept of IPC using shared memory.

Algorithm:

Step1: start the program.

Step2: Assign child = fork ().

Step3: Run the program.

Step4: If (! Child) then key of shared memory is printed and parent is writed and we enter the data Into the memory. Else Child is readed and data is read from the memory.

Step5: Stop the program.

Program:

```
#include<stdio.h>
#include<sys/ipc.h>
#include<sys/shm.h>
#include<string.h>
#include<unistd.h>
void main()
{
    int child,i,shmid;
    char*shmadd,buff[100];
    child=fork();
    if(!child)
    {
      shmid=shmget(1041,32,0666|IPC_CREAT);
      printf("Key of Shared memory is %d\n",shmid);
      shmadd=shmat(shmid,0,0);
      printf("PARENT IS WRITING\n");
      printf("Enter some data to write to shared memory\n");
      read(0,buff,100);
      strcpy(shmadd,buff);
    }
    else
```

```
        {
           sleep(15);
            shmid=shmget(1041,32,0666);
            printf("Key of Shared memory is %d\n",shmid);
            shmadd=shmat(shmid,0,0);
            printf("process attached at %p\n",shmadd);
           printf("CHILD IS READING\n");
           printf("Data read from memory is %s\n",(char*)shmadd);
           shmdt(NULL);
           shmctl(shmid,IPC_RMID,0);
        }
    }
```

Output:

```
hari@HARIPRIYA:~$ nano ipc.c
hari@HARIPRIYA:~$ gcc ipc.c
hari@HARIPRIYA:~$ ./a.out
Key of Shared memory is 0
PARENT IS WRITING
Enter some data to write to shared memory
AIML
Key of Shared memory is 0
process attached at 0x7f0dcb0be000
CHILD IS READING
Data read from memory is AIML
```

Exp. No:6	INTERPROCESS COMMUNICATION USING NAMED PIPES

Aim:

To write a program to illustrate the concept of IPC using named pipes.

Algorithm:

Step1: start the program.

Step2: create two files (i.e. sender.c and receiver.c).

Step3: Run the program.

Step4: In the sender.c program you give the input for the program and the entered text is received When the receiver.c program is executed.

Step5: Stop the program.

Program:

Sender.c

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
 #include<unistd.h>
 int main()
{
    int fd;
    char*myfifo = "/tmp/myfifo";
    mkfifo(myfifo,0666);
   char arr1[8],arr2[80];
    while(1)
    {
          fd=open(myfifo,O_WRONLY);
          fgets(arr2,80,stdin);
          write(fd,arr2,strlen(arr2)+1);
           close(fd);  }
    return 0;
}
```

Receiver.c

```
#include<stdio.h>
#include<string.h>
#include<fcntl.h>
#include<sys/stat.h>
#include<sys/types.h>
#include<unistd.h>
int main(){
    int fd1;
    char*myfifo = "/tmp/myfifo";
     mkfifo(myfifo,0666);
      char str1[80],str2[80];
       while(1)
            {
            fd1 = open(myfifo,O_RDONLY);
             read(fd1,str1,80);
             close(fd1);}
        return 0;
    }
```

Output:

```
hari@HARIPRIYA:~$ gcc sender.c
hari@HARIPRIYA:~$ ./a.out
AIML
HARI PRASATH
201501014
```

```
hari@HARIPRIYA:~$ gcc receiver.c
hari@HARIPRIYA:~$ ./a.out
AIML
HARI PRASATH
201501014
```

Exp. No: 7a	FIRST COME FIRST SERVE SCHEDULING

Aim:

To write a python program to implement first come first serve scheduling.

Algorithm:

Step1: start the program.

Step2: Run the program.

Step3: Enter the number of process.

Step4: Then enter the burst time for each process. Now the average waiting time and average turn Around time is calculated and it got printed.

Step5: Stop the program

Program:

```
bt=[]
print("Enter the number of process: ")
n=int(input())
print("Enter the burst time of the processes: \n")
for i in range(n):
bt.append(int(input()))
wt=[]
avgwt=0
tat=[]
avgtat=0
wt.insert(0,0)
tat.insert(0,bt[0])
for i in range(1,len(bt)):
wt.insert(i,wt[i-1]+bt[i-1])
 tat.insert(i,wt[i]+bt[i])
avgwt+=wt[i]
avgtat+=tat[i]
avgwt=float(avgwt)/n
avgtat=float(avgtat)/n
print("\n")
print("Process\t Burst Time\t Waiting Time\t Turn Around Time")
```

```
for i in range(0,n):
print(str(i)+"\t\t"+str(bt[i])+"\t\t"+str(wt[i])+"\t\t"+str(tat[i]))
print("\n")
print("Average Waiting time is: "+str(avgwt))
print("Average Turn Around Time is: "+str(avgtat))
```

Output:

```
Enter the number of process:
5
Enter the burst time of the processes:

1
2
3
4
5


Process  Burst Time      Waiting Time    Turn Around Time
0               1               0               1
1               2               1               3
2               3               3               6
3               4               6               10
4               5               10              15


Average Waiting time is: 4.0
Average Turn Around Time is: 6.8
```

Exp. No:7b	SHORTEST JOB FIRST SCHEDULING

Aim:

To write a python program to implement the CPU scheduling algorithm for the shortest job first.

Algorithm:

Step1: start the program.

Step2:Assign the proper conditions and then save it.

Step3: Run the program.

Step4: Enter the number of process.

Step5: Then the shortest job is assigned first and it got executed. Now the average waiting time and average turn Around time is calculated and it got printed.

Step6: Stop the program

Program:

```
bt=[]
print("Enter the number of process: ")
n=int(input())
processes=[]
for i in range(0,n):
print ("process"),i+1,"burst time "
processes.insert(i,i+1)
bt.append(int(input()))
for i in range(0,len(bt)-1):
for j in range(0,len(bt)-i-1):
if(bt[j]>bt[j+1]):
  temp=bt[j]
  bt[j]=bt[j+1]
 bt[j+1]=temp
  temp=processes[j]
 processes[j]=processes[j+1]
  processes[j+1]=temp
wt=[]
avgwt=0
```

```
 tat=[]
 avgtat=0
wt.insert(0,0)
 tat.insert(0,bt[0])
 for i in range(1,len(bt)):
 wt.insert(i,wt[i-1]+bt[i-1])
 tat.insert(i,wt[i]+bt[i])
  avgwt+=wt[i]
  avgtat+=tat[i]
  avgwt=float(avgwt)/n
  avgtat=float(avgtat)/n
  print("\n")
  print("Process\t Burst Time\t Waiting Time\t Turn Around Time")
  for i in range(0,n):
    print(str(processes[i])+"\t\t"+str(bt[i])+"\t\t"+str(wt[i])+"\t\t"+str(tat[i]))
  print("\n")
  print("Average Waiting time is: "+str(avgwt))
  print("Average Turn Around Time is: "+str(avgtat))
```

Output:

```
Enter the number of process:
3
process
1
process
2
process
3


Process  Burst Time      Waiting Time    Turn Around Time
3               1               0               1
1               2               1               3
2               3               3               6


Average Waiting time is: 1.3333333333333333
Average Turn Around Time is: 3.0
```

Exp. No: 8	PRODUCER-CONSUMER PROBLEM USING SEMAPHORES

Aim:

To write a C program to implement the Producer & consumer Problem (Semaphore).

Algorithm:

Step1: start the program.

Step2: Assign the values and save it.

Step3: Now run the program.

Step4: Enter the number of item to be produced and then the item is inserted. After assigning the Producer and consumer process is executed.

Step5: Stop the program

Program:

```
#include<stdio.h>
#include <pthread.h>
#include <semaphore.h>
#define MaxItems 5;
#define BufferSize 5
sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;
void *producer(void *pno)
{
int item;
for(int i=0;i<3;i++)
{
  sem_wait(&empty);
  pthread_mutex_lock(&mutex);
  printf("Enter item to be PRODUCED: ");
  scanf("%d",&item);
  buffer[in] = item;
```

```
            printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
            in = (in+1)%BufferSize;
            pthread_mutex_unlock(&mutex);
            sem_post(&full);
          }
        }
        void *consumer(void *cno)
        {
          int item;
         for(int i=0;i<3;i++)
         {
              sem_wait(&full);
              pthread_mutex_lock(&mutex);
              item = buffer[out];
               printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
              out = (out+1)%BufferSize;
              pthread_mutex_unlock(&mutex);
               sem_post(&empty);
            }
        }
int main()
 {
        pthread_t pro[5],con[5];
        pthread_mutex_init(&mutex, NULL);
        sem_init(&empty,0,BufferSize);
        sem_init(&full,0,0);
         int a[5] = {1,2,3};
         for(int i = 0; i < 3; i++)
        {
           pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
         }
        for(int i = 0; i < 3; i++)
        {
         pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
```

```
    }
   for(int i = 0; i < 3; i++)
   {
    pthread_join(pro[i], NULL);
   }
  for(int i = 0; i < 3; i++)
  {
   pthread_join(con[i], NULL);
  }
 pthread_mutex_destroy(&mutex);
 sem_destroy(&empty);
 sem_destroy(&full);
 return 0;
}
```

Output:

```
Enter item to be PRODUCED: 5
Producer 3: Insert Item 5 at 0
Enter item to be PRODUCED: 7
Producer 3: Insert Item 7 at 1
Enter item to be PRODUCED: 8
Producer 2: Insert Item 8 at 2
Enter item to be PRODUCED: 1
Producer 1: Insert Item 1 at 3
Consumer 3: Remove Item 5 from 0
Consumer 3: Remove Item 7 from 1
Enter item to be PRODUCED: 4
Producer 3: Insert Item 4 at 4
Consumer 2: Remove Item 8 from 2
Consumer 3: Remove Item 1 from 3
Consumer 1: Remove Item 4 from 4
Enter item to be PRODUCED: 5
Producer 1: Insert Item 5 at 0
Enter item to be PRODUCED: 6
Producer 1: Insert Item 6 at 1
Enter item to be PRODUCED: 6
Producer 2: Insert Item 6 at 2
Enter item to be PRODUCED: 7
Producer 2: Insert Item 7 at 3
Consumer 1: Remove Item 5 from 0
Consumer 1: Remove Item 6 from 1
Consumer 2: Remove Item 6 from 2
Consumer 2: Remove Item 7 from 3
```

Exp. No: 9	IMPLEMENTATION OF BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

Aim:

To write a UNIX C Program to implement deadlock avoidance using Banker's Algorithm.

Algorithm:

Step1: start the program.

Step2: Assign the values and save it.

Step3: Now run the program.

Step4: Enter the number of allocation matrix and the need matrix and available resources for the Program as an input.

Step5: Now the Safe Sequence is generated.

Step6: Stop the program

Program:

```
#include <stdio.h>
 int main()
 {
   int n, r, i, j, k;
   n = 5;
   r = 3;
   int alloc[5][3];
   printf("Enter the allocation matrix:\n");
  for (i = 0 ; i < n ; i++)
  {
  for (j = 0 ; j < r ; j++)
  {
     scanf("%d",&alloc[i][j]);
  }
 }
   int max[5][3];
 printf("Enter the Max Matrix:\n");
 for (i = 0 ; i < n ; i++)
{
 for (j = 0 ; j < r ; j++)
{
```

```
            scanf("%d",&max[i][j]);
          }
         }
        int avail[3];
        printf("Enter the available resources:\n");
        for(i = 0 ; i < r ; i++)
        scanf("%d",&avail[i]);
        int f[n], ans[n], ind = 0;
        for (k = 0; k < n; k++)
        {
         f[k] = 0;
        }
         int need[n][r];
         for (i = 0; i < n; i++)
         {
            for (j = 0; j < r; j++)
            need[i][j] = max[i][j] - alloc[i][j];
         }
           int y = 0;
          for (k = 0; k < 5; k++)
          {
           for (i = 0; i < n; i++)
         {
          if (f[i] == 0)
        {
           int flag = 0;
          for (j = 0; j < r; j++)
           {
           if (need[i][j] > avail[j])
           {
           flag = 1;
            break;
        }
}
```

```
if (flag == 0)
{
ans[ind++] = i;
for (y = 0; y < r; y++)
avail[y] += alloc[i][y];
f[i] = 1;
}
}
}
}
printf("Th SAFE Sequence is as follows\n");
for (i = 0; i < n - 1; i++)
printf(" P%d ->", ans[i]);
printf(" P%d\n", ans[n - 1]);
return (0);
}
```

Output:

```
hari@HARIPRIYA:~$ gcc banker.c
hari@HARIPRIYA:~$ ./a.out
Enter the allocation matrix:
1 2 3
1 2 3
1 2 3
1 2 3
1 2 3
Enter the Max Matrix:
4 5 6
5 6 7
1 2 4
3 2 1
5 6 8
Enter the available resources:
90 80 70
Th SAFE Sequence is as follows
 P0 -> P1 -> P2 -> P3 -> P4
```