OS LABORATORY

NAME : ANIRBAN HAZRA

ROLL : 2005643

SECTION : CSE - 16

INDEX

Name ANIRBAN HAZRA

Class 4th SEM Section CSE 16 Roll No. 2005643 Year 2022

Subject OPERATING SYSTEMS (OS LAB)

LAB Sl.	DATE LAB	DESCRIPTION	START PAGE	END PAGE	GRADE
1	11/01	Linux Commands	1	3	
		Files/Directories	4	6	
2	18/01	Shell Scripting	7	10	
3	25/01	SS - contd.	11	14	
4	01/02	SS - (if, else)	15	26	
		Switch Case.			
5	8/02	SS - Loops	27	32	
6	15/02	SS - Arrays	33	38	
7		FCFS, SJF	39	45	
	15/03	Algorithms in C			
8	22/03	SRTF, RR,	46	55	
		Priority Algo.			
9	29/03	Fork() call	56	61	
		Parent child.			
10	5/04	Pipe System Call	62	67	
		+ Programs.			
11	12/04	Inter Process	68	75	
		Communication			
		using Pipe.			
12	19/04	Two-Way Comm.	76	83	
		+ Thread Theory			

LABORATORY NO. 1

Anirban Hazra -
CSE 16, 2005643

COMMANDS

1. $pwd - shows the present working directory.
2. $mkdir - used to create a directory
3. $touch filename - used to empty a file with 0 bytes
4. $cat>file1 - opens a file with name file1, creates file1 if it doesnt exist
5. $cat f1 f2 > f3 - concatenates f1 and f2 into f3
6. $cp file1 file2 → copies file1 contents into file2
7. $cp f1 f2 f3 → copies f1 and f2 into f3
8. $rm f1 → removes files and directories
9. $mv f1 f2 → renames file f1 to f2.
10. $mv t1 t2 t3 → moves t1 and t2 to t3
11. $ls → lists all the files in the present directory.

Anirban Hazra -
CSE 16, 2005643

Anirban Hazra -
CSE 16, 2005643

(12) $ls -l → used to see the long listing of files and subdirectories inside current one.

(13) $cd dir1 → changes directory to dir1.

(14) $cd .. → moves back to the previous directory.

(15) $chmod → used to change the mode to add or remove permissions of a particular file.

(16) $bc → basic calculator, used to do operations which can be done using a calculator.

(17) $sort f1 f2 f3 → used to sort multiple files.

(18) $cut → a filter which cuts a given no. of char of fields from a specified file

(19) $ grep → searches specified input globally for a match, with the supplied pattern and display it.

(20) $ vi file1 / $gedit file1 → opens up the editor for a file, mostly used to write shell scripts into it.

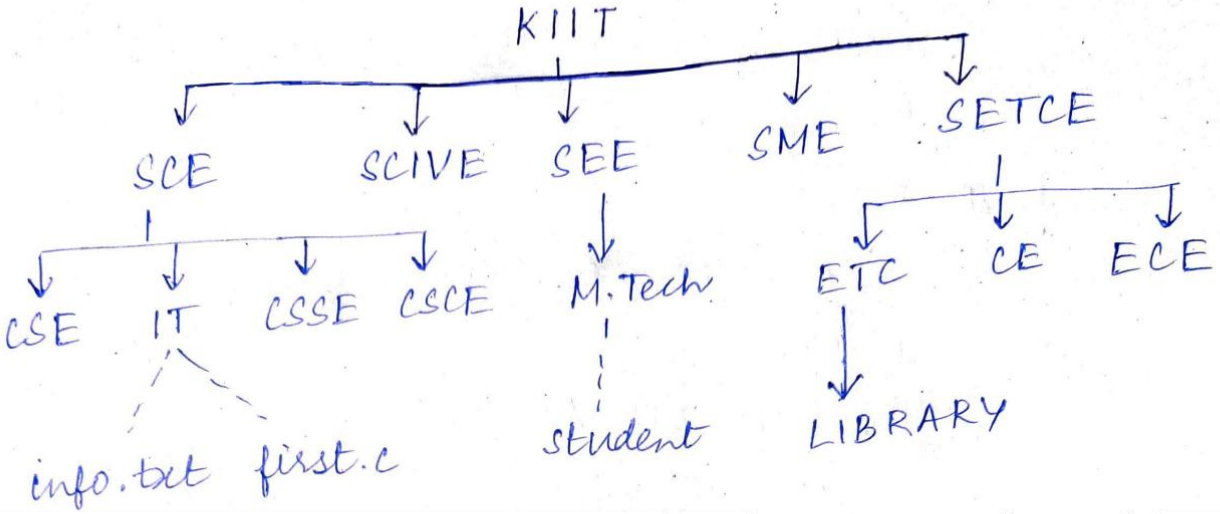
Anirban Hazra -
CSE 16, 2005643

ANIRBAN HAZRA
2005643

Anirban Hazra

OS LABORATORY 1

Q. Create a directory 'Roll Number', under that.



a) Create files and directories according to the above structure:

```
$ mkdir 2005643
$ cd 2005643
    $ mkdir KIIT
    $ cd KIIT
        $ mkdir SCE SCIVE SEE SME SETCE
        $ cd SCE
            $ mkdir CSE IT CSSE CSCE
            $ cd IT
```

Anirban Hazra -
CSE 16, 2005643

$ cat > info.txt

The name of this file is info.txt.

$ cat > first.c

The mane of this file is first.c

$ cd ..

$ cd ..

$ cd SEE

$ mkdir M.Tech

$ cd M.Tech

$ cat > student

The name of this file is student.

$ cd ..

$ cd ..

$ cd SETCE

$ mkdir ETC CE ECE

$ cd ETC

$ mkdir LIBRARY

$ cd ..

$ cd ..

b) Rename the file info.txt to itstudentdata.txt,

```
$ cd SCE/IT
    $ mv info.txt itstudentdata.txt.
```

c) Copy the file first.c into directory CE with the same name:

```
$ cp first.c /home/Kiit/2005643/KIIT/SETCE/CE
```

d) Copy the file first.c into directory SME with a new name hello.c.

```
$ cp first.c /home/Kiit/2005643/KIIT/SME/hello.c
```

e) Transfer file student into SCIVE and check if transferred or not.

```
        $ cd..
      $ cd..
   $ cd   SEE/M.Tech
      $ mv student /home/Kiit/2005643/KIIT/SCIVE
        $ cd..
      $ cd..
   $ cd SCIVE
      $ ls
Student
```

LABORATORY NO. 2

Anirban Hazra -
CSE 16, 2005643

OS LABORATORY 2

Q1. Write a Shell Script (WASS) to do calculator program using Arithmetic operators.

PROGRAM CODE:

```
# to do the calculator program
echo enter 2 numbers
read a b
c='expr $a + $b'
d=$((a * b))
e='expr $a - $b'
f='expr $a / $b'
echo Sum is $c
echo Product is $d
echo Difference is $e
echo Quotient is $f
```

OUTPUT:

```
enter 2 numbers
10 5
Sum is 15
Product is 50
Difference is 5
Quotient is 2
```

Anirban Hazra -
CSE 16, 2005643

Q2. WASS to find the average of 3 numbers.

PROGRAM CODE:

```
# TO find average of 3 numbers.
echo Enter 3 numbers
read a b c
d = 'expr $a + $b + $c'
e = 'expr $d / 3'
'echo Average of 3 numbers is $e
```

OUTPUT:

```
Enter 3 numbers
4 6 8
Average of 3 numbers is 6
```

Q3. WASS to find the Simple Interest, when P, R, T are given.

PROGRAM CODE:

```
# Calculate the Simple Interest
echo Enter P, R and T
read P R T
SI = 'expr $P \* $T \* $R / 100'
echo Simple Interest is $SI
```

OUTPUT:

```
Enter P, R and T
2000  4  10
Simple Interest is 800.
```


.Anirban Hazra -
CSE 16 , 2005643

Q4 WASS to find area of a triangle when base and height are given.

PROGRAM CODE:

```
# Area of a Triangle
echo Enter base and height of Triangle
read b h
ar = $(((b*h)/2))
echo Area of Triangle is $ar.
```

OUTPUT:

```
Enter base and Height of Triangle
5  4
Area of Triangle is 10.
```

Q5 WASS to find area and circumference of a circle.

PROGRAM CODE:

```
#Area and circumference of a circle.
echo Enter the Radius of circle
read r
ar = $(bc <<< "$r * $r * 3.14")
c = $(bc <<< " 2 * 3.14 * $r ")
echo circumference is $c
echo Area is $ar.
```

OUTPUT

```
Enter the Radius of Circle
5
circumference is 31.40
Area is 78.50.
```


.Anirban Hazra.
CSE 16, 2005643

LABORATORY NO. 3

.Anirban Hazra.
CSE 16, 2005643

OS LABORATORY - 3

Q1. WASS to calculate Compound Interest when P, R, T are given.

PROGRAM CODE:

```
# program to calculate compound Interest
echo Enter values of P, R, T
read P R T
ci=`echo "scale=2; $p*((1+$r*0.01)^$t)" | bc -l`
echo COMPOUND INTEREST is $ci
```

OUTPUT

```
Enter values of P, R, T
450 6.7 5
COMPOUND INTEREST IS 598.50
```

Q2. Calculate distance travelled by an object when acceleration, initial velocity and time are given.

PROGRAM CODE:

```
# calculate distance travelled by an object
echo Enter values of a, u, t
read a u t
s=`echo "scale=2; ($u*$t)+(0.5*$a*$t^2)" | bc -l`
echo DISTANCE travelled is $s.
```

OUTPUT

```
Enter values of a, u, t
4 6.6 7
DISTANCE travelled is 144.2
```


.Anirban Hazra.
CSE 16, 2005643

Q3 WASS to find the read roots of a quadratic equation

PROGRAM CODE:

```
# to find real roots of quadratic equation
echo Enter a,b,c of polynomial
read a b c
r1='echo "scale=2; (-$b - sqrt($b * $b - 4 * $a * $c))/(2 * $a)" | bc`
r2='echo "scale=2; (-$b + sqrt($b * $b - 4 * $a * $c))/(2 * $a)" | bc`
echo The roots are $r1 and $r2
```

OUTPUT

```
Enter a,b,c of polynomial
1 11 30
The roots are -6.00 and -5.00
```

Q4 WASS to calculate area and perimeter of a rectangle

PROGRAM CODE:

```
# Area and Perimeter of a rectangle
echo Enter length and breadth
read l b
p='echo "scale=2; (2 * $l + 2 * $b)" | bc -l`
a='echo "scale=2; (2 * $l * $b)" | bc -l`
echo Perimeter is $p
echo Area is $a
```

OUTPUT:

```
Enter length and breadth
5.5 6
Perimeter is 23.0
Area is 66.0
```


.Anirban Hazra -
CSE 16 , 2005643

Q5 WASS to convert Celcius to Fahrenheit

.Anirban Hazra -
CSE 16 , 2005643

PROGRAM CODE:

```
# Convert Celcius to Fahrenheit
echo Enter value of Celcius
read c
f=`echo "scale=2; $c * (9/5) + 32" | bc -l `
echo The value of Fahrenheit is $f
```

OUTPUT:

```
Enter value of Celcius
32
The value of Fahrenheit is 89.60
```


.Anirban Hazra -
CSE 16 , 2005643

LABORATORY NO. 4

Anirban Hazra -
CSE 16, 2005643

.Anirban Hazra.
CSE 16, 2005643

OS LABORATORY - 4.

Q1. For an employee, if basic sal is less than 1500. Then HRA = 10% of basic, DA = 90% of basic. If basic sal >= 1500, HRA = 500, DA = 98%. Calculate Gross Sal using shell script.

PROGRAM CODE:

```
# to calculate gross salary of employee
echo Enter basic salary
read s
if [ $s -lt 1500 ]
then
hra='echo "scale=2; (0.1 * $s)" | bc -l`
da='echo "scale=2; (0.9 * $s)" | bc -l`
else
hra=500
da='echo "scale=2; (0.98 * $s)" | bc -l`
fi
g='echo "scale=2; ($s + $hra + $da)" | bc -l`
echo Gross Salary is $g
```

OUTPUT

```
Enter basic Salary
1800
Gross Salary is 4064.00
```

.Anirban Hazra.
CSE 16, 2005643

.Anirban Hazra.
CSE 16, 2005643

Q2. Find the greater between 2 numbers.

PROGRAM CODE:

```
#find greater between 2 numbers
echo Enter 2 numbers.
read a b
if test $a -lt $b
then
echo $b is greater than $a
else
echo $a is greater than $b.
fi
```

OUTPUT

```
Enter 2 numbers.
5  6
6 is greater than 5.
```

.Anirban Hazra.
CSE 16, 2005643

.Anirban Hazra.
CSE 16, 2005643

Q3 Find greatest of 3 numbers.

PROGRAM CODE:

```
#NESTED IF ELSE
echo Enter 3 nos.
read a b c
if [ $a -gt $b]
then
if [ $a -gt $c]
then
echo $a is greatest
else
echo $c is greatest
fi
elif [ $b -gt $c]
then
echo $b is greatest
else
echo $c is greatest
fi
```

OUTPUT

```
Enter 3 nos.
5 7 10
10 is greatest
```

.Anirban Hazra.
CSE 16, 2005643

.Anirban Hazra.
CSE 16, 2005643

```
# USING IF ELSE LADDER.
echo Enter 3 numbers.
read a b c
if [ $a -gt $b -a $a -gt $c ]
then
    echo $a is greatest
elif [ $b -gt $a -a $b -gt $c ]
then
    echo $b is greatest
else
    echo $c is greatest
fi
```

OUTPUT

```
Enter 3 numbers.
4 7 9
9 is greatest
```

.Anirban Hazra.
CSE 16, 2005643

Anirban Hazra -
CSE 16, 2005643

Q4 TO find if a number is odd or even

PROGRAM CODE:

```
# to find if odd or even
echo Enter a number
read x
p=`expr $x % 2`
if [ $p -eq 0 ]
then
 echo Even Number
else
 echo Odd Number.
fi
```

OUTPUT

```
Enter a number
5
Odd Number.

Enter a number
16
Even Number.
```

Anirban Hazra -
CSE 16, 2005643

.Anirban Hazra-
CSE 16, 2005643

Q5 WASS to calculate electricity bill in different slabs

PROGRAM CODE:

```
echo Enter number of units consumed
read n
if [ $n -lt 500]
then
bill = `echo "scale=2; $n * 50 " | bc -l`
elif [ $n -eq 500 -a $n -lt 1000]
then
bill = `echo " scale=2; $n * 100" | bc -l`
else
bill = `echo " scale=2; $n * 150 " | bc -l`
fi
echo Bill is Rs $bill.
```

OUTPUT:

```
Enter number of units consumed
650
Bill is 97500.
```

Q6 WASS to find the read roots of a quadratic equation

PROGRAM CODE:

```
# to find real roots of quadratic equation
echo Enter a,b,c of polynomial
read a b c
```

.Anirban Hazra-
CSE 16, 2005643

```
r1=`echo "scale=2; (-$b - sqrt($b * $b - 4 * $a * $c))/(2 * $a)" | bc`
r2=`echo "scale=2; (-$b + sqrt($b * $b - 4 * $a * $c))/(2 * $a)" | bc`

echo The roots are $r1 and $r2
```

OUTPUT

```
Enter a, b, c of polynomial
1 11 30
The roots are -6.00 and -5.00
```

Q7 Check if a number is +ve, -ve or zero.

```
echo Enter the number
read n
if [ $n -lt 0 ]
then
echo NUMBER IS NEGATIVE
elif [ $n -gt 0 ]
then
 echo NUMBER IS POSITIVE
else
echo NUMBER is 0.
fi
```

OUTPUT:

Enter the number 5 NUMBER IS POSITIVE	Enter the number 0 NUMBER IS 0	Enter the number -5 NUMBER IS NEGATIVE

.Anirban Hazra-
CSE 16, 2005643

Q8 Show the days of the week from number which is input using case.

PROGRAM CODE:

```
echo Enter a number
read n
case $n in
1)
 echo Day is Monday
 ;;
2)
 echo Day is Tuesday
 ;;
3)
echo Day is Wednesday
 ;;
4)
echo Day is Thursday
 ;;
5)
 echo Day is Friday
 ;;
6) echo Day is Saturday
 ;;
7)
 echo Day is Sunday
 ;;
*)
 echo WRONG INPUT
 ;;
esac
```

OUTPUT:

```
Enter a number
5
Day. is Friday
```

.Anirban Hazra-
CSE 16, 2005643

Q9. Enter an alphabet and check if it is vowel or consonant using case.

Anirban Hazra -
CSE 16, 2005643

PROGRAM CODE:

```
echo Enter an Alphabet
read $s
case $s in
a)
echo Alphabet is vowel
;;
e)
echo Alphabet is vowel
;;
i)
echo Alphabet is Vowel
;;
o)
echo Alphabet is vowel
;;
u)
echo Alphabet is vowel.
;;
*)
echo Alphabet is consonant.
;;
esac.
```

OUTPUT:

```
Enter an alphabet
f
Alphabet is consonant
```

```
Enter an alphabet
u
Alphabet is vowel.
```

Anirban Hazra -
CSE 16, 2005643

.Anirban Hazra.
CSE 16, 2005643

Q10. Implement Simple Calculator using switch case.

PROGRAM CODE:

```
echo Enter 2 numbers
read a b
echo 1 for ADD
echo 2 for SUBTRACT
echo 3 for MULTIPLY
echo 4 for DIVIDE
echo 5 for MODULUS

read c
case $c in
1)
c=`expr $a + $b`
echo SUM is $c
;;
2)
c=`expr $a - $b`
echo DIFFERENCE is $c
;;
3)
c=`expr $a \* $b`
echo PRODUCT is $c
;;
4)
c=`expr $a / $b`
echo QUOTIENT is $c
;;
5)
c=`expr $a % $b`
echo MODULUS is $c
;;
*)
echo WRONG INPUT
;;
esac
```

.Anirban Hazra.
CSE 16, 2005643

.Anirban Hazra-
CSE 16, 2005643

OUTPUT:

```
Enter 2 numbers.
5 8
1 for ADD
2 for SUBTRACT
3 for MULTIPLY
4 for DIVIDE
5 for MODULUS.
3
PRODUCT is 40
```

```
Enter 2 numbers.
5 8
1 for ADD
2 for SUBTRACT
3 for MULTIPLY
4 for DIVIDE
5 for MODULUS
1
SUM is 13
```

.Anirban Hazra-
CSE 16, 2005643

LABORATORY NO. 5

.Anirban Hazra-
CSE 16, 2005643

OS LABORATORY 5

Q1. WASS to check if a given number is prime or not.

```
# while loop.
echo Enter a number
read n
i=2
f=0
while [ $i -le `expr $n/2` ]
do
if [ `expr $n % $i` -eq 0 ]
then
f=1
fi
i=`expr $i + 1`
done
if [ $f -eq 1]
then
echo NOT PRIME
else
echo PRIME
fi
```

```
# until loop
echo Enter a number
read n
i=2
f=0
until
    [ $i -gt `expr $n/2` ]
do
if [ `expr $n % $i` -eq 0]
then
f=1
fi
i=`expr $i + 1`
done
if [ $f -eq 1 ]
then
echo NOT PRIME
else
echo PRIME
fi
```

OUTPUT

```
Enter a number
5
PRIME
```

```
Enter a number
6
NOT PRIME
```

.Anirban Hazra-
CSE 16, 2005643

.Anirban Hazra-
CSE 16, 2005643

Q2 WASS to find the reverse of a number

PROGRAM CODE:

```
#using while loop
echo Enter a number
read n
rev=0
p=0
while [ $n -gt 0]
 do
p='expr $n %. 10`
rev='expr $rev \* 10 + $p`
 n='expr $n /10`
done
echo Reverse Number $rev
```

```
#using until loop.
echo Enter a number
read n
rev=0
p=0
until [ $n -le 0]
do
p='expr $n %. 10`
rev='expr $rev \* 10 + $p`
n='expr $n /10`
done
echo Reverse Number $rev
```

OUTPUT:

```
Enter a number
567
Reverse Number 765
```

```
Enter a number
235.
Reverse Number 532
```

.Anirban Hazra-
CSE 16, 2005643

.Anirban Hazra-
CSE 16, 2005643

Q3 WASS to find sum of digits of a number

PROGRAM CODE:

```
# using while loop
echo Enter a number
read num
sum=0
while [ $num -gt 0]
do
mod='expr $num % 10`
sum='expr $sum + $mod`
num='expr $num/10'
done
echo Sum is $sum.
```

```
# using until loop
echo Enter a number
read num
sum=0
until [ $num -le 0]
do
mod='expr $num % 10`
sum='expr $sum + $mod`
num='expr $num/10`
done
echo Sum is $sum.
```

OUTPUT:

```
Enter a number
234
Sum is 9.
```

```
Enter a number
532
Sum is 10.
```

.Anirban Hazra-
CSE 16, 2005643

Q4 WASS to find factorial of a number

.Anirban Hazra-
CSE 16, 2005643

PROGRAM CODE:

```
echo Enter a number
read num
f=1
while [ $num -gt 1 ]
#until [ $num -le 1]
do
f=`expr $num - 1`
done
echo $f is the factorial
```

OUTPUT:

```
Enter a number
5
120 is the factorial.
```

Q5. WASS to check if a number is a palindrome or not.

PROGRAM CODE:

```
echo Enter a number
read n
z=$n
rev=0
p=0
while [ $n -gt 0]
#until [ $n -le 0]
```

.Anirban Hazra-
CSE 16, 2005643

```
do
 p='expr $n % 10`
rev='expr $rev \* 10 + $p`
 n='expr $n / 10`
 done
if [ $z -eq $rev ]
 then
 echo PALINDROME
 else
 echo NOT PALINDROME
```

OUTPUT :

① Enter a number
10101
PALINDROME

---

② Enter a number
11000
NOT PALINDROME

LABORATORY NO. 6

OS LABORATORY 6

Anirban Hazra -
CSE 16, 2005643

Q1 Write shell script to print array elements in reverse order.

PROGRAM CODE:

```
echo Enter the size of Array
read n
i=0
while [ $i -lt $n]
do
read a[$i]
i=`expr $i + 1`
done
echo Reverse Array is :
i= $n
while [$i -ge 0]
do
echo ${a[$i]}
i=`expr $i - 1`
done.
```

OUTPUT:

```
$ bash Array_Reverse
Enter the size of the Array.
5
1
2
3
4
6
Reverse Array is :
6
4
3
2
1.
```

Anirban Hazra -
CSE 16, 2005643

Q2 WASS to find largest of array elements

PROGRAM CODE:

```
echo Enter size:
read n
i=0
while [ $i -lt $n]
do
read a[$i]
i=`expr $i +1`
done
max=${a[0]}
for i in "${a[@]}"
do
if [[ "$i" -gt "$max" ]]
then
max = $i
fi
done
echo Largest element: $max
```

OUTPUT:

```
$ bash Array-max
Enter size
3
2
7
5
Largest element: 7
```

Q3. WASS to find sum of odd and even numbers in an array

PROGRAM CODE:

```
echo Enter the size of the Array:
read n
 i=0
while [$i -lt $n]
do
read a[$i]
i =`expr $i + 1`
done
even = 0
odd = 0
 i = 0
```


Anirban Hazra
CSE 16, 2005643

Anirban Hazra -
CSE 16, 2005643

```
while [$i -lt $n]
do
if [`expr ${a[$i]} % 2` -eq 0]
then
even =`expr ${a[$i]} + $even`
else
odd =`expr ${a[$i]} + $odd`
fi
i=`expr $i+1`
done
echo Sum of odd Numbers is $odd
echo Sum of even Numbers is $even
```

OUTPUT:

```
$ bash Array_oddevensum
Enter the size of Array:
4
2
5
6
7
Sum of Odd Numbers is 12
Sum of Even Numbers is 8
```

Q4. WASS to search particular element in Array.

PROGRAM CODE:

```
echo Enter number of elements
read n
echo Enter the Array elements
i=1
while [$i -le $n]
do
read a[$i]
```


Anirban Hazra -
CSE 16, 2005643

```
i =`expr $i + 1`
done
echo Enter element to be searched
read x
j= 1
while [ $j -lt $n -a $item -ne ${a[$j]} ]
do
j=`expr $j +1`
done
if [ $item -eq ${a[$j]} ]
then
echo $item is at position $j
else
echo $item is not present in Array.
fi
```

OUTPUT:

```
Enter the number of elements:
5
Enter the array elements:
2
10
5
3
11
Enter the element to be searched
10
10 is present at location 2.
```


Anirban Hazra -
CSE 16, 2005643

Q5. WASS to sort an Array.

Anirban Hazra -
CSE 16, 2005643

PROGRAM CODE:

```
# sorting an array with Bubble Sort
echo Enter size:
read n
i=0
while [ $i -lt $n ]
do
read a[$i]
i=`expr $i +1`
done
 i=0
while [ $i -lt $n ]
do
        s=`expr $n - $i - 1`
        j=0
        while [ $j -lt $s ]
        do
            if [ ${a [$j]} -gt ${a [$j+1]} ]
            then
                t = ${a[$j]}
                a[$j] = ${a[$j+1]}
                a[$j+1] = $t
            fi
            j=`expr $j +1`
        done
        i=`expr $i + 1`
done
i=0
echo Sorted Array:
while [ $i -lt $n ]
do
echo ${a[$i]}
i=`expr $i +1`
done
```

OUTPUT:

```
Enter size
3
  2
  7
  3
  Sorted Array:
  2
  3
  7
```


Anirban Hazra -
CSE 16, 2005643

LABORATORY NO. 7

OS LABORATORY 7

Anirban Hazra - CSE 16, 2005643

Q1. Write a program to implement FCFS algorithm using C.

```
#include <stdio.h>
void findWaitingTime(int processes[], int n,
                     int bt[], int wt[])
{  wt[0] = 0;
   for(int i = 0; i<n ; i++)
    wt[i] = bt[i-1] + wt[i-1];
}
void findTurnAroundTime(int processes[];
                        int n, int bt[], int wt[],
                        int tat[])
{  for(int i=0; i<n; i++)
     tat[i] = bt[i] + wt[i];
}
void findavgTime(int processes[], int n,
                 int bt[])
{  int wt[n], tat[n], total_wt = 0;
   int total_tat = 0;
   findWaitingTime(processes, n, bt, wt);
   findTurnAroundTime(processes, n, bt, wt,
                      tat);
   printf("\n Processes | Burst Time | Waiting Time
          | Turn Around Time\n");
   for(int i=0; i<n; i++)
   {  total_wt = total_wt + wt[i];
      total_tat = total_tat + tat[i];
      printf("%d", (i+1));
```


Anirban Hazra - CSE 16, 2005643

```
    printf("\t  | %d", bt[i] );
    printf("\t  | %d", wt[i]);
    printf("\t  | %d\n", tat[i]);
  }
  int s = (float) total_wt / (float) n;
  int t = (float) total_tat / (float) n;
  printf("\n Avg. Waiting Time = %d", s);
  printf("\n Avg. Turn Around Time = %d", t);
}
int main()
{ int processes[10], burst_time[10], n;
  printf(" Enter number of processes : ");
  scanf("%d", &n);
  for(int i=0; i<n; i++)
    processes[i] = i+1;
  printf(" Enter burst times of %d processes :",
                                    n);
  for(int i=0; i<n; i++)
  { int p=0;
    scanf("%d", &p);
    burst_time[i] = p;
  }
  findavgTime(processes, n, burst_time);
  return 0;
}
```


Anirban Hazra - CSE 16, 2005643

OUTPUT: (FCFS Algorithm)

Enter the number of processes : 3

Enter burst times of 3 processes : 4 5 8

Processes	Burst Time	Waiting Time	TA Time
1	4	0	4
2	5	4	9
3	8	9	17

Average Waiting Time = 4

Average TurnAround Time = 10

Q2 Write a program in C to implement the SJF algorithm of Scheduling.

```
#include<stdio.h>
int main()
{
    int bt[20], p[20], wt[20], tat[20], i, j;
    int n, total=0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of processes : ");
    scanf("%d", &n);
    printf("Enter Burst Time : \n");
    for(i=0; i<n; i++)
    {
        printf("p%d : ", i+1);
        scanf("%d", &bt[i]);
        p[i] = i+1;
    }
    for(i=0; i<n; i++)
    {
        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;
    }
```

Anirban Hazra -
CSE 16, 2005643

.Anirban Hazra.
CSE 16, 2005643

```
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  Burst Time \t Waiting Time
                    \t Turn Around Time");

for(i=0; i<n; i++)
{ tat[i] = bt[i] + wt[i];
  total += tat[i];
  printf("\np%d \t\t %d \t\t
        %d \t\t\t %d", p[i], bt[i], wt[i],
                          tat[i]);
}
avg_tat = (float) total/n;
printf("\n\n");
printf("Average Waiting Time= %f",
                          avg_wt);

printf("\n");
printf("Average Turn Around Time = %f",
                          avg_tat);
}
```


.Anirban Hazra.
CSE 16, 2005643

OUTPUT : (SJF Algorithm)

Enter number of processes : 3

Enter Burst Times :

p1 : 4

p2 : 5

p3 : 8

Process	Burst Time	Waiting Time	TurnAround Time
p1	4	0	4
p2	5	4	9
p3	8	9	17

Average Waiting Time = 4.333333

Average Turn Around Time = 10.000000

.Anirban Hazra.
CSE 16, 2005643

LABORATORY NO. 8

Anirban Hazra - CSE 16, 2005643

Q1. Write a program in C to implement SRTF Scheduling Algorithm

```
#include<stdio.h>
int main()
{ int at[10], bt[10], rt[10], endTime, smallest;
  int remain=0, i, n, time, sum_wait=0;
  int sum_turnaround=0;
  printf("Enter number of processes : ");
  scanf("%d", &n);
    for(i=0; i<n; i++)
     {
     printf("Enter arrival time for P%d :", i+1);
     scanf("%d", &at[i]);
     printf("Enter burst time for P%d :", i+1);
     scanf("%d", &bt[i]);
     rt[i] = bt[i];
    }

printf("\n\n Process \t| Turn Around Time|
                          Waiting Time\n");

rt[9] = 9999;

for(time=0; remain!=n; time++)
{ smallest = 9;
  for(i=0; i<n; i++)
  { if(at[i]<=time && rt[i]<rt[smallest]
                   && rt[i]>0)

    { smallest = i;
    }
```


Anirban Hazra - CSE 16, 2005643

```
    }
    rt[smallest]--;
    if(rt[smallest] == 0)
    {
      remain++;
      endTime = time + 1;
      printf("\nP[%d]\t|\t%d\t|\t%d",
          smallest+1, endTime - at[smallest],
          endTime - at[smallest] - bt[smallest]);
      sum_wait += endTime - bt[smallest]
                  - at[smallest];
      sum_turnaround += endTime -
                        at[smallest];
    }
  }
  printf("\n\n");
  printf("Average Waiting Time = %f\n",
                  sum_wait * 1.0/n);
  printf("\n");
  printf("Average TurnAround Time = %f",
                  sum_turnaround * 1.0/n);
  return 0;
}
```


Anirban Hazra -
CSE 16, 2005643

OUTPUT: (SRTF Algorithm)

```
Enter no. of Processes: 3
Enter Arrival Time of Process P1 : 1
Enter Busst Time for Process P1 : 4
Enter Arrival Time for Process P2 : 2
Enter Busst Time for Process P2 : 5
Enter Arrival Time for Process P3: 3
Enter Burst Time for Process P3: 8
```

Process	TurnAround Time	Waiting Time
P[1]	4	0
P[2]	8	3
P[3]	15	7

Average Waiting Time = 3.333333
Average TurnAround Time = 5.400000

.Anirban Hazra.
CSE 16, 2005643

32. WAP to implement Round Robin Scheduling Algorithm in C.

Anirban Hazra -
CSE 16, 2005643

```
# include <stdio.h>
int main()
{ int i, limit, total=0, x, counter=0;
int time_quantum, wait_time=0;
  int turnaround_time=0, arrival_time[10];
  int burst_time[10], temp[10];
  float average_wait_time;
  float average_turnaround_time;
  printf("\nEnter Total Number of Processes : \t");
  scanf("%d", &limit);
  x = limit;
  for(i=0; i<limit ; i++)
  { printf("Enter details of Process [%d]\n", i+1);
    printf("Arrival Time : \t");
    scanf("%d", &arrival_time[i]);
    printf("Burst Time : \t");
    scanf("%d", &burst_time[i]);
    temp[i] = burst_time[i];
  }
  printf("\nEnter Time quantum : \t");
  scanf("%d", &time_quantum);
  printf("\nProcess ID \t\t Burst Time \t
          TurnAround Time \t Waiting Time\n");
  for(total=0, i=0 ; x!=0 ; )
  { if(temp[i]<=time_quantum && temp[i]>0)
```

```
{   total = total + temp[i];
    temp[i] = 0;
    counter = 1;
  }
  else if (temp[i] > 0)
  {   temp[i] = temp[i] - time_quantum;
      total = total + time_quantum;
  }
  if (temp[i] == 0 && counter == 1)
  {   x--;
      printf("\nProcess[%d]\t\t%d\t\t%d
              \t\t\t%d", i+1, burst_time[i],
               total - arrival_time[i],
            total - arrival_time[i] - burst_time[i]);

      wait_time = wait_time + total - burst_time[i]
                  - arrival_time[i];

      turnaround_time = turnaround_time +
            total - arrival_time[i];
      counter = 0;
  }
  if (i == limit-1)  i=0;
  else if (arrival_time[i+1] <= total)  i++;
  else  i=0;
}
average_wait_time = wait_time * 1.0/limit;
average_turnaround_time = turnaround_time
                          * 1.0/limit;
printf("\nAverage Waiting Time :\t%f",
                  average_wait_time);
printf("\nAvg Turnaround Time :\t%f",
                  average_turnaround_time);
return 0;
}
```


Anirban Hazra - CSE 16, 2005643

OUTPUT : (Round Robin Algorithm)

Enter Total Number of Processes : 3

Enter details of Process [1]
Arrival Time : 1
Busst Time : 4

Enter details of Process [2]
Arrival Time : 2
Busst Time : 5

Enter details of Process [3]
Arrival Time : 3
Busst Time : 8

Enter Time Quantum : 3

Process Id	Busst Time	TurnAround Time	Waiting
Process [1]	4	9	5
Process [2]	5	10	5
Process [3]	8	14	6

Average Waiting Time : 5.333333
Average TurnAround Time : 11.000000

Anirban Hazra -
CSE 16, 2005643

Q3. WAP to implement Priority Scheduling Algorithm using c programming.

```
#include<stdio.h>
#include<string.h>
int main()
{ int et[20], at[10], n, i, j, temp, p[10], st[10];
  int ft[10], wt[10], ta[10], totwt=0, totta=0;
  float awt, ata;
  char pn[10][10], t[10];
  printf("Enter the number of processes :");
  scanf("%d", &n);
  for(i=0; i<n; i++)
  { printf("Enter process name, arrival time,
            execution time and priority : ");
    scanf("%s %d %d %d", pn[i], &at[i],
                         &et[i], &p[i]);
  }
  for(i=0; i<n; i++)
     for(j=0; j<n; j++)
     { if(p[i]<p[j])
       { temp=p[i];
         p[i]=p[j];
         p[j]=temp;
         temp=at[i];
         at[i]=at[j];
         at[j]=temp;
         temp=et[i];
         et[i]=et[j];
         et[j]=temp;
        strcpy(t, pn[i]);
        strcpy(pn[i], pn[j]);
        strcpy(pn[j], t);
```


Anirban Hazra - CSE 16, 2005643

```
        }
    }
for(i=0; i<n; i++)
{
    if (i==0)
    { st[i] = at[i];
      wt[i] = st[i] - at[i];
      ft[i] = st[i] + et[i];
      ta[i] = ft[i]  - at[i];
    }
    else
    { st[i] = ft[i-1];
      wt[i] = st[i] - at[i];
      ft[i] = st[i] + et[i];
      ta[i] = ft[i] - at[i];
    }
    totwt += wt[i];
    totta += ta[i];
}
awt = (float) totwt/n;
ata = (float) totta/n;
printf("\n Pname \t Arrival Time \t Execution Time
      \t Priority \t Waiting Time \t TA Time");
for (i=0; i<n; i++)
printf("\n%s \t %5d \t \t %5d \t \t %5d
      \t \t %5d \t \t %5d", pn[i], at[i],
      et[i], p[i], wt[i], ta[i]);
printf("\n Average Waiting Time is : %f", awt);
printf("\n Average TurnAround Time is : %f", ata);
return 0;
}
```


.Anirban Hazra-
CSE 16, 2005643

OUTPUT: (Priority Scheduling Algorithm)

Enter number of processes: 3
Enter process, AT, ET and priority: 1 2 3 4
Enter process, AT, ET and priority: 2 3 4 5
Enter process, AT, ET and priority: 3 4 5 6

Pname	Arrival	Execution	Priority	Waiting	TA
1	2	3	4	0	3
2	3	4	5	2	6
3	4	5	6	3	10

Average Waiting Time: 2.333333
Average TurnAround Time: 6.333333

Anirban Hazra -
CSE 16, 2005643

LABORATORY NO. 9

LABORATORY - 9

Anirban Hazra - CSE 16, 2005643

NOTES:

Process Creation Using Fork System Call.

- Fork system call is used for creation of child process. If fork is successfully executed, then its going to create a new child process, by duplicating call of parent process.
- Child Process and Parent Process run in separate memory spaces. At the time of fork, both memory space have same content.

Child Process is the exact duplicate of Parent Process, except child has it's own PId, which doesn't match with the ID of any existing process group.

Header Files → <sys/types.h>, <unistd.h>

Predefined Data Type → pid_t q; → value.

q < 0 → error

q == 0 → child process

q = fork(); → gives the value.

q > 0 → Parent Process

getpid() → gives ID of current process | getppid() → gives ID of parent process

Wait System call in C

- A call to wait function blocks the calling process until one of it's child process terminates or a signal is received.
- After child process terminates, parent process continues its execution, after wait system call instruction. If any process has more than 1 CP, then after calling wait(), parent process has to be in waiting state, if no child terminates. If only 1 child process is terminated, then wait() returns the pid of terminated child process. If many CP are terminates, it returns PID of any arbitrary child. If no CP is terminated, returns (-1). When wait returns, it also defines its exit status by a pointer.

Anirban Hazra -
CSE 16, 2005643

Q1. Write a program to find sum of even numbers in parent process and sum of odd numbers in child process.

PROGRAM CODE:

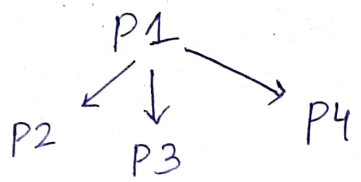
```
#include <stdio.h>
#include <unistd.h>
int main()
{ int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  int sumOdd = 0, sumEven = 0, n, i;
  n = fork();
  if (n > 0)
    { for (i = 0; i < 10; i++)
        { if (a[i] % 2 == 0)
            sumEven = sumEven + a[i];
        }
      printf ("Parent Process: ");
      printf ("Sum of even no.s is : %d\n", sumEven);
    }
  else
    { for (i = 0; i < 10; i++)
        { if (a[i] % 2 != 0)
            sumOdd = sumOdd + a[i];
        }
      printf ("Child Process: ");
      printf ("Sum of Odd nos. is : %d", sumOdd);
    }
  return 0;
}
```

OUTPUT:

```
Parent Process : Sum of Even nos. is : 30
Child Process : Sum of Odd nos. is : 25
```

Q2 Create Process Tree using Fork System Call.

Anirban Hazra -
CSE 16, 2005643



PROGRAM CODE:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <wait.h>

int main()
{
    pid_t q;
    for(int i=0 ; i<3 ; i++)
    {
        q = fork();
        if(q==0)
        {
            break;
        }
    }
    printf("Child PId - %d\n", getpid());
    printf("Parent PId - %d\n\n", getppid());
    wait(NULL);
    return 0;
}
```

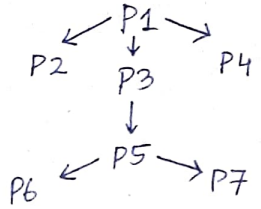
OUTPUT:

```
Child PId - 24187
Parent PId - 24180

Child PId - 24188
Parent PId - 24187

Child PId - 24190
Parent PId - 24187

Child PId - 24189
Parent PId - 24187
```


Anirban Hazra -
CSE 16, 2005643

Q3. create process tree using fork system call.

Anirban Hazra -
CSE 16, 2005643



PROGRAM CODE:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <wait.h>
int main()
{ pid_t q1, q2, q3;
  for(int i=0; i<3; i++)
    { q1 = fork();
      if(q1==0)
      { if(i==1)
          { q2 = fork();
            if(q2>0) break;
            else
            { for(int j=0; j<2; j++)
              { q3 = fork();
                if(q3==0) break;
              }
              break;
            }
          }
        break;
      }
    }
  printf("PId - %d and PPId - %d\n",
                    getpid(), getppid());
  wait(NULL);
  return 0;
}
```

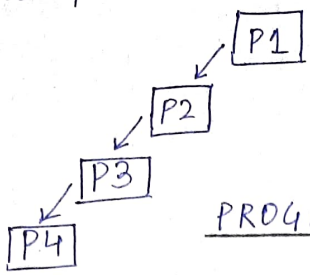
OUTPUT:

```
PId - 15676 and PPId - 15675
PId - 15675 and PPId - 15668
PId - 15677 and PPId - 15675
PId - 15678 and PPId - 15675
PId - 15679 and PPId - 15677
PId - 15681 and PPId - 15679
PId - 15680 and PPId - 15679
```

Anirban Hazra -
CSE 16, 2005643

Q4 Create process tree using fork system call.

Anirban Hazra
CSE 16, 2005643



PROGRAM CODE:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
int main()
{
  pid_t p, c1, c2;
  printf("parent pid = %d\n", getpid());
  p = fork();
  if (p<0) printf("error");
  else if (p==0)
  { printf("child_1 pid = %d, ppid = %d\n",
                    getpid(), getppid());
    c1 = fork();
    if (c1==0)
    { printf("child_2 pid = %d, ppid = %d\n",
                    getpid(), getppid());
      c2 = fork();
      if (c2==0)
      { printf("child_3 pid = %d, ppid = %d\n",
                    getpid(), getppid());
      }
      else wait(NULL);
    }
    else wait(NULL);
  }else wait(NULL);
  return 0;
}
```

OUTPUT:

```
parent pid = 5565
child_1 pid = 5569, ppid = 5565
child_2 pid = 5570, ppid = 5569
child_3 pid = 5571, ppid = 5570
```


Anirban Hazra
CSE 16, 2005643

LABORATORY NO. 10

LABORATORY - 10

Anirban Hazra.
CSE 16, 2005643

NOTES:

Pipe System Call : Process Communication

- A pipe is a technique used for interprocess communication. It is a mechanism by which the output of one process is directed towards the other. Thus it provides one way flow of data between related processes.
- Although pipe can be accessed as an ordinary file system, actually it occurs as a FIFO queue. A pipe file is created using pipe() system call. A pipe has an input end and output end. One can write into a pipe from input end, and read from output end.
- A pipe descriptors has an array that stores 2 pointers. (one for input, one for output)

Input and output end (we need to get location) so pointer is needed to access them.

Suppose 2 process A and B needs to communicate. In such case, it is important that process which writes, closes its read end, of the pipe, and process which reads closes it's write end of the pipe. Esentially, for communication of A to B, process A should keep write end operand read end closes, (vice versa for B)

- Pipe is given fixed size in bytes. When a process attempts to write input to a pipe, the write request is immediately accessed if pipe is full, If full, the process is blocked, unless state of pipe changes.
- Only one process can be accessed at a time.

Limitations

- As a channel of communications, pipe() operates in one direction only.
- Pipe cannot support broadcast. (at a time, multiple processes not allowed)
- Read end, reads anyway (doesn't matter which process is connected to right end of pipe). So it is an insecure mode of communication.

* `#include <unistd.h>`
`int pipe (int pipedes[2]);`
The pipe system call has 2 pipedes:
pipedes[0] → Reading, pipedes[1] → Writing
This call returns 0 as success, -1 as failure

To know the cause of failure, we can use errno. with errno. variable or perror function

```
#include <sys/types.h>
#include <sp/stat.h>
(1) #include <fentl.h>
int open(const char *pathname, int flags);
int open(const char *pathname, int flags,
                                mode_t mode);
```

Even though the basic operations for file are read and write, it is essential to open the file before performing the operations and closing the file after completion of reqd. opt.

- By default, 3 descriptors opened for every process, used for input (standard input stdin) output (stdout) and error (stderr) having file descriptors 0, 1, 2 resp.

(2) `#include <unistd.h>`
`int close (int fd)`
The above system call closes already opened file descriptor. This implies the file is no longer in

use and resources associated can be reused by any other process.

System call return 0 on success
return -1 on error.

```
# include <unistd.h>
```

(3)

```
SSize_t write(int fd, void * buf, size_t count);
```

Above system call is used to write from specified file with arguments of file descriptor fd, proper buffer with allocated memory (static or dynamic) and size of buffer.

error : errno, perror();

(4)

```
#include <unistd.h>
size_t read(int fd, void * buf, size_t count)
```

File needs to be opened before writing to the file, automatically opens in case of calling pipe() sys. call.

Returns no. of bytes read (or zero in case of EOF) on success and -1 in failure.

error : errno, perror();

Q Write Algorithm to read and write 2 messages using pipe();

1. Create pipe
2. Send a message to the pipe.
3. Retriene msg from pipe and write to stdout.
4. Repeat 2 and 3 steps

Q. Write Algorithm to do the above using parent child process.

1. Create pipe
2. Create child
3. Parent process writes to pipe
4. Child retrieves msg from pipe and writes to stdout
5. Repeat 3 and 4 steps

Q1. Read and Write 2 messages using pipe() system call.

Anirban Hazra - CSE 16, 2005643

PROGRAM CODE :

```
#include<stdio.h>
#include<unistd.h>
int main()
{ int pipefds[2];
  int returnstatus;
  char writemessages[2][20]={"ABC", "DEF"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
  if (returnstatus == -1)
  { printf("Unable to create pipe\n");
    return 1;
  }
  printf("Writing to pipe - Message 1 is %s\n",
                          writemessages[0]);
  write(pipefds[1], writemessages[0],
                  sizeof(writemessages[0]));
  read(pipefds[0], readmessage,
                  sizeof(readmessage));
  printf("Reading from pipe - Message 1 is %s\n",
                          readmessage);
  printf("      Writing to pipe - Message 2 is %s\n",
                          writemessages[0]);
  write(pipefds[1], writemessages[1],
                  sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe - Message 2 is %s\n",
                          readmessage);
  return 0;
}
```

OUTPUT

```
Writing to pipe - Message 1 is ABC
Reading from pipe - Message 1 is ABC
Writing to pipe - Message 2 is ABC
Reading to pipe - Message 2 is DEF.
```

Anirban Hazra - CSE 16, 2005643

Q2. Read and Write 2 messages through parent child process

Anirban Hazra
CSE 16, 2005643

PROGRAM CODE:

```
#include <stdio.h>
#include <unistd.h>
int main()
{ int pipefds[2], returnstatus, pid;
  char writemessages[2][20] = {"Anirban", "Hazra"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
  if (returnstatus == -1)
  { printf("Unable to create pipe\n");
    return 1;
  }
  pid = fork();
  if (pid == 0) // Child Process
  { read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Child Process: Reading: Message1 is %s\n",
                                    readmessage);
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Child Process: Reading: Message2 is %s\n",
                                    readmessage);
  }
  else // Parent Process
  {printf("Parent Process: Writing: Message 1 is %s\n",
                                    writemessages[0]);
  write(pipefds[1], writemessages[0], sizeof(
                                    writemessages[0]));
  printf("Parent Process: Writing: Message 2 is %s\n",
                                    writemessages[1]);
  write(pipefds[1], writemessages[1], sizeof(
                                    writemessages[1]));
  }
  return 0;
}
```

OUTPUT

```
Parent Process: Writing: Message 1 is Anirban
Parent Process: Writing: Message 2 is Hazra
  Child Process: Reading: Message 1 is Anirban
  Child Process: Reading: Message 2 is Hazra
```

LABORATORY NO. 11

LABORATORY - 11

Inter Process Communication Using Pipe

Q1. WAP in C to read and write a message using pipe() func.

Anirban Hazra -
CSE 16, 2005643

PROGRAM CODE:

```
#include<stdio.h>
#include<unistd.h>
int main()
{ int pipefds[2], returnstatus;
  char writemessages[1][20]={"Anirban"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
 if(returnstatus == -1)
 {printf("Unable to create pipe\n");
  return 1;
 }
printf("Writing to pipe - Message 1 is %s\n",
                    writemessages[0]);
write(pipefds[1], writemessages[0],
                    sizeof(writemessages[0]));
read(pipefds[0], readmessage, sizeof(
                    readmessage));
printf("Reading from pipe - Message 1 is %s\n",
                    readmessage);
return 0;
}
```

OUTPUT:

```
Writing to pipe - Message 1 is Anirban
Reading from pipe - Message 1 is Anirban
```


Anirban Hazra -
CSE 16, 2005643

Q2. Read and Write 2 messages using pipe() system call.

Anirban Hazra - CSE 16, 2005643

PROGRAM CODE :

```
#include<stdio.h>
#include<unistd.h>
int main()
{ int pipefds[2];
  int returnstatus;
  char writemessages[2][20]={"ABC","DEF"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
  if (returnstatus == -1)
  { printf("Unable to create pipe\n");
    return 1;
  }
  printf("Writing to pipe - Message 1 is %s\n",
                          writemessages[0]);
  write(pipefds[1], writemessages[0],
                  sizeof(writemessages[0]));
  read(pipefds[0], readmessage,
                  sizeof(readmessage));
  printf("Reading from pipe - Message 1 is %s\n",
                          readmessage);
  printf("Writing to pipe - Message 2 is %s\n",
                          writemessages[0]);
  write(pipefds[1], writemessages[1],
                  sizeof(writemessages[0]));
  read(pipefds[0], readmessage, sizeof(readmessage));
  printf("Reading from pipe - Message 2 is %s\n",
                          readmessage);
  return 0;
}
```

OUTPUT

```
Writing to pipe - Message 1 is ABC
Reading from pipe - Message 1 is ABC
Writing to pipe - Message 2 is ABC
Reading to pipe - Message 2 is DEF
```

Anirban Hazra - CSE 16, 2005643

Q3. WAP in C, in which read a message through pipe(), parent process writes message into pipe and child process retrieves and displays it.

PROGRAM CODE:

```
#include<stdio.h>
#include<unistd.h>
int main()
{ int pipefds[2], returnstatus, pid;
  char writemessages[1][20]={"Anirban"};
  char readmessage[20];
  returnstatus = pipe(pidefds);
  if(returnstatus == -1)
  { printf("Unable to create pipe\n");
    return 1;
  }
  pid = fork();
  if(pid ==0) //Child Process
  { read(pipefds[0], readmessage, sizeof(
                                readmessage));
    printf("Child Process - Reading from pipe :
            Message 1 is %.s\n", readmessage);
  }
  else //Parent Process
  { printf("Parent Process -      Writing to pipe :
            Message 1 is %.s\n", writemessages[0]);
    write(pipefds[1], writemessages[0], sizeof(
                                writemessage[0]));
  }
  return 0;
}
```

OUTPUT:

Parent Process : Reading from Pipe : Message 1 is Anirban
Child Process : Writing to pipe : Message 1 is Anirban.

Q 4 Read and Write 2 messages through parent child process, where CP writes message into pipe and parent process retsrieves and displays it.

PROGRAM CODE:

```
#include <stdio.h>
#include <unistd.h>
int main()
{ int pipefds[2], returnstatus, pid;
  char writemessages[2][20] = {"Anirban", "Hazra"};
  char readmessage[20];
  returnstatus = pipe(pipefds);
  if (returnstatus == -1)
  { printf("Unable to create pipe\n");
    return 1;
  }
  pid = fork();
  if (pid == 0) // Child Process
  { read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Child Process: Reading: Message1 is %s\n",
                                    readmessage);
    read(pipefds[0], readmessage, sizeof(readmessage));
    printf("Child Process: Reading: Message2 is %s\n",
                                    readmessage);
  }
  else // Parent Process
  {printf("Parent Process: Writing: Message 1 is %s\n",
                                    writemessages[0]);
   write(pipefds[1], writemessages[0], sizeof(
                                    writemessages[0]));
   printf("Parent Process: Writing: Message 2 is %s\n",
                                    writemessages[1]);
   write(pipefds[1], writemessages[1], sizeof(
                                    writemessages[1]));
  }
  return 0;
}
```

OUTPUT

```
Parent Process: Writing: Message 1 is Anirban
Parent Process: Writing: Message 2 is Hazra
Child Process: Reading: Message 1 is Anirban
Child Process: Reading: Message 2 is Hazra
```

Q 5 Write a program to achieve two-way comm. using pipes

PROGRAM CODE :

```
#include<stdio.h>
#include<unistd.h>
int main()
{ int pipefds[1], pipefds[2];
  int returnstatus1, returnstatus2, pid;
  char pipe1writemessage[20] = "Hi";
  char pipe2writemessage[20] = "Hello";
  char readmessage[20];
  returnstatus1 = pipe (pipefds1);
  returnstatus2 = pipe (pipefds2);
  if(returnstatus1 == -1)
  { printf("Unable to create pipe1 \n");
    return 1;
  }
  if(returnstatus2 == -1)
  { printf("Unable to create pipe2 \n");
    return 1;
  }
  pid = fork();
  if(pid != 0) // Parent process
  { close(pipefds1[0]);
    close[pipefds2[1]);
    printf("Parent : Writing : Pipe1 : Message is %s\n",
                                  pipe1writemessage);
    write(pipefds1[1], pipe1writemessage, sizeof
                                  (pipe1writemessage));
    read(pipefds[0], readmessage, sizeof
                                  (readmessage));
    printf("Parent : Reading : Pipe 2 : Message is %s \n",
                                  readmessage);
```


}

Anirban Hazra -
CSE 16, 2005643

```
else // Child Process
{ close(pipefds1[1]);
  close(pipefds2[0]);
  read(pipefds1[0], readmessage, sizeof(readmessage));
  printf("Child: Reading: Pipe1: Message is %s\n",
                                  readmessage);
  printf("Child: Writing: Pipe2: Message is %s\n",
                                  pipe2writemessage);
}
return 0;
}
```

OUTPUT:

```
Parent: Writing: Pipe1: Message is Hi
Child: Reading: Pipe1: Message is Hi
Child: Writing: Pipe2: Message is Hello
Parent: Reading: Pipe2: Message is Hello
```

Q6 Write a program in C to write 2 numbers from parent and read by child and add those numbers and print the sum

PROGRAM CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <time.h>
#include <sys/wait.h>
int main()
{ int p1[2], p2[2];
  if(pipe(p1) == -1) return 1;
  if(pipe(p2) == -1) return 1;
  int t = fork();
```


Anirban Hazra -
CSE 16, 2005643

Anirban Hazra -
CSE 16, 2005643

```
    if (t==0)
    {   close(p1[0]);
        close(p2[1]);
        int a, b;
        if(read(p2[0], &a, sizeof(a)) == -1) return 3;
        printf("1st number in child is %d\n", a);
        if(read(p2[0], &b, sizeof(b)) == -1) return 3;
        printf("2nd number in child is %d\n", b);
        int sum = a+b;
        printf("Sum in Parent Process is %d\n", sum);
        if(write(p1[1], &sum, sizeof(sum)) == -1) return 4;
        printf("Wrote from Child Process %d\n", sum);
        close(p1[1]);
        close(p2[0]);
    }
    else
    {   close(p1[1]);
        close(p2[0]);
        srand(time(NULL));
        int z = rand() % 100;
        int y = rand() % 100;
        if(write(p2[1], &y, sizeof(y)) == -1) return 5;
        printf("Wrote from parent process %d\n", y);
        if(write(p2[1], &z, sizeof(z)) == -1) return 6;
        printf("Wrote from parent process %d\n", z);
        int sump;
        if(read(p1[0], &sump, sizeof(sump)) == -1) return 7;
        printf("Result received in parent process: %d\n",
                                                    sump);
        wait(NULL);
        close(p1[0]);
        close(p2[1]);
    }
    return 0;
}
```

OUTPUT:

```
Wrote from Parent Process 24
Wrote from Parent Process 26
1st number in child is 24
2nd number in child is 26
Sum in parent process is 50
Wrote from Child Process 50
Result received in parent process is 50.
```

LABORATORY NO. 12

LABORATORY - 12.

Two Way Communication using Pipes().

Anirban Hazra -
CSE 16, 2005643

2 pipes → Pipe 1 : Write → Parent
Read → Child.
Pipe 2 : Write → Child
Read → Parent.

- Pipe Communication is viewed as only one way communication i.e either parent writes and child reads or vice versa (not both)
- However for both parent and child needs to read and write pipe simultaneously, the solution is 2 way communication.

1. Create 2 pipes.
   Pipe 1 → Parent writes.
   Child Reads
   Pipe 2 → Parent Reads.
   Child Writes.
2. Create a Child Process.
3. Close unwanted ends as only 1 end is reqd. for each communication
4. Close Read End of Pipe 1
   Write End of Pipe 2
   } Parent.
5. Close Write End of Pipe 1
   Read End of Pipe 2
   } Child.

Read → 0
write → 1.

Anirban Hazra -
CSE 16, 2005643

Thread Theory.

Anirban Hazra -
CSE 16, 2005643

POSIX Thread (p-thread \ library)

- Posix libraries are standard thread API for C and C++. It allows us to create multiple threads for concurrent message flow. It is most effective on multi-processor or multicore systems, where threads can be implemented on Kernel level
- By exploiting the latency in I/O or other system func. that may halt a process

. pthread.h → header file.

pthread_create : used to create a new thread

```
int pthread_create (pthread_t * thread,
const pthread_attr_t * attr, void * (
  * start_routine)(void *), void * arg);
```

1. thread → pointer to an unsigned integer val that returns thread id of thread created.
2. attr → pointer to a structure used to define thread attributes (detached state, scheduling policy, stack address). It is set to null for default attributes.
3. start_routine → pointer to subroutine, i.e executed by the thread. The return type and the parameter type of the subroutine has to be void *. The func has a single attribute, but if multiple values need to be passed to the func, struct must be used.

arcs → pointer to void that contains argument to the function defined in earlier argument

1) void pthread_exit(void *retval)

pthread_exit:
This method accepts a mandatory parameter 'retval' which is an pointer to an integer that stores return status of the thread terminated. The scope of this variable must be global so that any thread waiting to join may read the join status.
Used to terminate the thread.

2) int pthread_join(pthread_t th, void **thread return);

- Used to wait for termination of the thread.
- th - This is the ID for which the thread waits.
- thread_return → It is a pointer to a location where exit status of thread mentioned in ptr is stored.

3) int pthread_t pthread_self(void);

- used to get thread ID of unread thread.

4) pthread_equal(pthread_t t1, pthread_t t2);

- It compares whether threads are same or not. If 2 threads are equal the func. return non zero value otherwise 0.
t1 and t2 are thread IDs of 2 threads.

5) int pthread_cancel(pthread_t thread);

- sends cancellation reqd. to thread
- accepts mandatory parameter which is the thread ID of thread of which cancellation request is sent.

6) int pthread_detach(pthread_t thread);

- used to detach a thread. A detached thread doesn't require a thread to join on terminating. The resources of the thread are automatically released after terminating if thread is detached.

- thread returns the thread ID of the thread that must be detached.

Anirban Hazra
CSE 16, 2005643

Q1 Write a program to achieve two-way comm. using pipes

Anirban Hazra
CSE 16, 2005643

PROGRAM CODE:

```
#include<stdio.h>
#include<unistd.h>
int main()
{ int pipefds1[1], pipefds1[2];
  int returnstatus1, returnstatus2, pid;
  char pipe1writemessage[20]="Hi";
  char pipe2writemessage[20]="Hello";
  char readmessage[20];
  returnstatus1 = pipe (pipefds1);
  returnstatus2 = pipe (pipefds2);
  if(returnstatus1==-1)
  { printf("Unable to create pipe1 \n");
    return 1;
  }
  if(returnstatus2==-1)
  { printf("Unable to create pipe2 \n");
    return 1;
  }
  pid = fork();
  if(pid!=0) // Parent process
  { close(pipefds1[0]);
    close(pipefds2[1]);
    printf("Parent: Writing: Pipe1: Message is %s\n"
                        pipe1writemessage);
    write(pipefds1[1], pipe1writemessage, sizeof
                        (pipe1writemessage));
    read(pipefds[0], readmessage, sizeof
                        (readmessage));
    printf("Parent: Reading: Pipe2: Message is %s\n",
                        readmessage);
```


}

```
    else // Child Process
    {
    close(pipefds1[1]);
    close(pipefds2[0]);
    read(pipefds1[0], readmessage, sizeof(readmessage));
    printf("Child : Reading : Pipe1 : Message is %s \n",
                                        readmessage);
    printf("Child : Writing : Pipe2 : Message is %s\n",
                                    pipe2writemessage);
    }
    return 0;
}
```

OUTPUT :

```
Parent : Writing : Pipe1 : Message is Hi
Child : Reading : Pipe1 : Message is Hi
Child : Writing : Pipe2 : Message is Hello
Parent : Reading : Pipe2 : Message is Hello
```

Q 2 Write a program in C to write 2 numbers from parent and read by child and add those numbers and print the sum

PROGRAM CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <time.h>
#include <sys/wait.h>
int main()
{ int p1[2], p2[2];
  if (pipe(p1) == -1) return 1;
  if (pipe(p2) == -1) return 1;
  int t = fork();
```


Anirban Hazra -
CSE 16, 2005643

Anirban Hazra -
CSE 16, 2005643

```
    if (t==0)
    {  close(p1[0]);
       close(p2[1]);
       int a, b;
       if(read(p2[0], &a, sizeof(a)) == -1) return 3;
       printf("1st number in child is %d\n", a);
       if(read(p2[0], &b, sizeof(b)) == -1) return 3;
       printf("2nd number in child is %d\n", b);
       int sum = a+b;
       printf("Sum in Parent Process is %d\n", sum);
       if(write(p1[1], &sum, sizeof(sum)) == -1) return 4;
       printf("Wrote from Child Process %d\n", sum);
       close(p1[1]);
       close(p2[0]);
    }
    else
    {  close(p1[1]);
       close(p2[0]);
       srand(time(NULL));
       int x = rand() % 100;
       int y = rand() % 100;
       if(write(p2[1], &y, sizeof(y)) == -1) return 5;
       printf("Wrote from parent process %d\n", y);
       if(write(p2[1], &z, sizeof(z)) == -1) return 6;
       printf("Wrote from parent process %d\n", z);
       int sump;
       if(read(p1[0], &sump, sizeof(sump)) == -1) return 7;
       printf("Result received in parent process: %d\n",
                                                  sump);
       wait(NULL);
       close(p1[0]);
       close(p2[1]);
    }
    return 0;
}
```

OUTPUT:

```
Wrote from Parent Process 24
Wrote from Parent Process 26
1st number in child is 24
2nd number in child is 26
Sum in parent process is 50
Wrote from Child Process 50
Result received in parent process is 50.
```

THANK YOU

Anirban Hazra
CSE 16, 2005643