



# **VELLORE INSTITUTE OF** **TECHNOLOGY-AP**



*CSE-2008 OPERATING*  
*SYSTEMS*

**ASSIGNMENT-1**

**OPERATING SYSTEM LAB MANUAL**

**TATHAGAT BANERJEE**

**17BCE7100**

**SEMESTER 4**

## **List of Experiments:**

S. No.	Experiment Name	Page Number
1.	Write a program in C that creates a child process, waits for the termination of the child and lists its PID, together with the state in which the process was terminated.	3
2.	Write a program in C to implement different scheduling algorithms.	4-10
3.	Write a program to multiply two matrices that uses threads to divide up the work necessary to compute the product of two matrices.	11-12
4.	Write a C program to implement the following game given	13-18
5.	Write a program in C to implement producer consumer problem.	19-20
6.	Write a program in C to perform fork() system call to create child processes.	21
7	Write a program in C to implement inter process communication.	22-23

# EXPERIMENT 1

**AIM: Write a program in C that creates a child process, waits for the termination of the child and lists its PID, together with the state in which the process was terminated.**

## CODE:

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

int main()
{
    int stat;
    pid_t cpid ;
    if (fork()== 0)
        {printf("HC: hello from child\n");
        printf("PID child = %d terminated\n", getpid());
        }

    else
    {
        printf("HP: hello from parent\n");
        wait(NULL);
        printf("CT: child has terminated\n");
        printf("PID parent = %d terminated\n", getpid());
        printf("Exit status of CHILD: %d\n", WIFEXITED(stat));

    }

    printf("Bye\n");

    return 0;
}
```

## OUTPUT:

```
fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q3 Q3.c
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q3
HP: hello from parent
HC: hello from child
PID child = 14363 terminated
Bye
CT: child has terminated
PID parent = 14362 terminated
Exit status of CHILD: 1
Bye
fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

# EXPERIMENT 2

**AIM: Write a program in C to implement different scheduling algorithms.**

**CODE:**

## **a) First Come First Serve**

```
#include<stdio.h>
int main()
{
    int n,bt[20],wt[20],tat[20],avwt=0,avtat=0,i,j;
    printf("Enter total number of processes(maximum 20):");
    scanf("%d",&n);
    printf("\nEnter Process Burst Time\n");
    for(i=0;i<n;i++)
    {
        printf("P[%d]:",i+1);
        scanf("%d",&bt[i]);
    }
    wt[0]=0; //waiting time for first process is 0
    //calculating waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
    }
    printf("\nProcess\t\tBurst Time\tWaiting Time\tTurnaround Time");
    //calculating turnaround time
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        avwt+=wt[i];
        avtat+=tat[i];
        printf("\nP[%d]\t\t%d\t\t%d\t\t%d",i+1,bt[i],wt[i],tat[i]);
    }
    avwt/=i;
    avtat/=i;
    printf("\n\nAverage Waiting Time:%d",avwt);
    printf("\nAverage Turnaround Time:%d",avtat);
    return 0;
}
```

## OUTPUT:

```
fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q4_FCFS Q4_FCFS.c
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q4_FCFS
Enter total number of processes(maximum 20):4

Enter Process Burst Time
P[1]:14
P[2]:3
P[3]:13
P[4]:2

Process          Burst Time    Waiting Time    Turnaround Time
P[1]              14            0               14
P[2]               3            14              17
P[3]              13            17              30
P[4]               2            30              32

Average Waiting Time:15
Average Turnaround Time:23fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

## b) Round Robin

```
#include<stdio.h>

int main()
{
    int count,j,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;
    for(count=0;count<n;count++)
    {
        printf("Enter Arrival Time and Burst Time for Process Process Number %d
        :",&count+1);
        scanf("%d",&at[count]);
        scanf("%d",&bt[count]);
        rt[count]=bt[count];
    }
    printf("Enter Time Quantum:\t");
    scanf("%d",&time_quantum);
    printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\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|\t%d\t|\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("\nAverage Waiting Time= %f\n",wait_time*1.0/n);
printf("Avg Turnaround Time = %f",turnaround_time*1.0/n);
return 0;
}

```

## OUTPUT:

```
fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q4_RR Q4_RR.c
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q4_RR
Enter Total Process:      4
Enter Arrival Time and Burst Time for Process Process Number 1 :0
3
Enter Arrival Time and Burst Time for Process Process Number 2 :1
2
Enter Arrival Time and Burst Time for Process Process Number 3 :4
3
Enter Arrival Time and Burst Time for Process Process Number 4 :2
6
Enter Time Quantum:      3

Process |Turnaround Time|Waiting Time
P[1]    |      3      |      0
P[2]    |      4      |      2
P[3]    |      4      |      1
P[4]    |     12      |      6

Average Waiting Time= 2.250000
Avg Turnaround Time = 5.750000fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

### c) Shortest Job First

```
#include<stdio.h>
void main()
{
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("\nEnter Burst Time:\n");
for(i=0;i<n;i++)
{
printf("p%d:",i+1);
scanf("%d",&bt[i]);
p[i]=i+1; //contains process number
}
//sorting burst time in ascending order using selection sort
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;
}
wt[0]=0; //waiting time for first process will be zero
//calculate waiting time
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; //average waiting time
total=0;
printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
for(i=0;i<n;i++)

{
tat[i]=bt[i]+wt[i]; //calculate turnaround time
total+=tat[i];
printf("\np%d\t\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]);
}
avg_tat=(float)total/n; //average turnaround time
printf("\n\nAverage Waiting Time=%f",avg_wt);
printf("\nAverage Turnaround Time=%f\n",avg_tat);
}

```

## OUTPUT:

```

fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q4_SJF
Enter number of process:5

Enter Burst Time:
p1:6
p2:5
p3:2
p4:2
p5:8

Process  Burst Time      Waiting Time      Turnaround Time
p3         2              0              2
p4         2              2              4
p2         5              4              9
p1         6              9             15
p5         8             15             23

Average Waiting Time=6.000000
Average Turnaround Time=10.600000
fsrt@fsrt-VirtualBox:~/Desktop/OS$ █

```



## d) Priority Schedule

```
#include<stdio.h>
int main()
{
    int bt[20],p[20],wt[20],tat[20],pr[20],i,j,n,total=0,pos,temp,avg_wt,avg_tat;
    printf("Enter Total Number of Process:");
    scanf("%d",&n);

    printf("\nEnter Burst Time and Priority\n");
    for(i=0;i<n;i++)
    {
        printf("\nP[%d]\n",i+1);
        printf("Burst Time:");
        scanf("%d",&bt[i]);
        printf("Priority:");
        scanf("%d",&pr[i]);
        p[i]=i+1; //contains process number
    }
    //sorting burst time, priority and process number in ascending order using
    selection sort
    for(i=0;i<n;i++)
    {
        pos=i;
        for(j=i+1;j<n;j++)
        {
            if(pr[j]<pr[pos])
                pos=j;
        }
        temp=pr[i];
        pr[i]=pr[pos];
        pr[pos]=temp;
        temp=bt[i];
        bt[i]=bt[pos];
        bt[pos]=temp;
        temp=p[i];
        p[i]=p[pos];
        p[pos]=temp;
    }
    wt[0]=0; //waiting time for first process is zero

    //calculate waiting time
    for(i=1;i<n;i++)
    {
        wt[i]=0;
        for(j=0;j<i;j++)
            wt[i]+=bt[j];
        total+=wt[i];
    }
    avg_wt=total/n; //average waiting time
    total=0;
    printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i]; //calculate turnaround time
        total+=tat[i];
        printf("\nP[%d]\t\t %d\t\t %d\t\t %d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=total/n; //average turnaround time
    printf("\n\nAverage Waiting Time=%d",avg_wt);
    printf("\n\nAverage Turnaround Time=%d\n",avg_tat);
    return 0;
}
```

## OUTPUT:

```
fsrt@fsrt-VirtualBox: ~/Desktop/OS
File Edit View Search Terminal Help
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q4_PS
Enter Total Number of Process:4

Enter Burst Time and Priority

P[1]
Burst Time:2
Priority:2

P[2]
Burst Time:3
Priority:3

P[3]
Burst Time:4
Priority:1

P[4]
Burst Time:3
Priority:4

Process  Burst Time    Waiting Time    Turnaround Time
P[3]          4             0                4
P[1]          2             4                6
P[2]          3             6                9
P[4]          3             9               12

Average Waiting Time=4
Average Turnaround Time=7
fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

# EXPERIMENT 3

**AIM:** Write a program to multiply two matrices that uses threads to divide up the work necessary to compute the product of two matrices. There are several ways to improve the performance using threads. You need to divide the product in row dimension among multiple threads in the computation. That is, if you are computing  $A * B$  and  $A$  is a  $10 \times 10$  matrix and  $B$  is a  $10 \times 10$  matrix, your code should use threads to divide up the computation of the 10 rows of the product which is a  $10 \times 10$  matrix. If you were to use 5 threads, then rows 0 and 1 of the product would be computed by thread 0, rows 2 and 3 would be computed by thread 1,...and rows 8 and 9 would be computed by thread 4.

## CODE:

```
// CPP Program to multiply two matrix using pthreads
// #include <stdc++.h>
#include <bits/stdc++.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
using namespace std;

// maximum size of matrix
#define MAX 4

// maximum number of threads
#define MAX_THREAD 4

int matA[MAX][MAX];
int matB[MAX][MAX];
int matC[MAX][MAX];
int step_i = 0;

void* multi(void* arg)
{
    int core = step_i++;

    // Each thread computes 1/4th of matrix multiplication
    for (int i = core * MAX / 4; i < (core + 1) * MAX / 4; i++)
        for (int j = 0; j < MAX; j++)
            for (int k = 0; k < MAX; k++)
                matC[i][j] += matA[i][k] * matB[k][j];
}

// Driver Code
int main()
{
    // Generating random values in matA and matB
    for (int i = 0; i < MAX; i++) {
        for (int j = 0; j < MAX; j++) {
            matA[i][j] = rand() % 10;
            matB[i][j] = rand() % 10;
        }
    }
}
```

```

// Displaying matA
cout << endl
    << "Matrix A" << endl;
for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++)
        cout << matA[i][j] << " ";
    cout << endl;
}

// Displaying matB
cout << endl
    << "Matrix B" << endl;
for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++)
        cout << matB[i][j] << " ";
    cout << endl;
}

// declaring four threads
pthread_t threads[MAX_THREAD];

// Creating four threads, each evaluating its own part
for (int i = 0; i < MAX_THREAD; i++) {
    int* p;
    pthread_create(&threads[i], NULL, multi, (void*)(p));
}

// joining and waiting for all threads to complete
for (int i = 0; i < MAX_THREAD; i++)
    pthread_join(threads[i], NULL);

// Displaying the result matrix
cout << endl
    << "Multiplication of A and B" << endl;
for (int i = 0; i < MAX; i++) {
    for (int j = 0; j < MAX; j++)
        cout << matC[i][j] << " ";
    cout << endl;
}
return 0;
}

```

## OUTPUT:

```

/Desktop/OS$ g++ -pthread Q5.cpp
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./a.out

Matrix A
3 7 3 6
9 2 0 3
0 2 1 7
2 2 7 9

Matrix B
6 5 5 2
1 7 9 6
6 6 8 9
0 3 5 2

Multiplication of A and B
43 100 132 87
56 68 78 36
8 41 61 35
56 93 129 97
fsrt@fsrt-VirtualBox:~/Desktop/OS$

```

# EXPERIMENT 4

**AIM:** Write a C program to implement the following game. The parent program P first creates two pipes, and then spawns two child processes C and D. One of the two pipes is meant for communications between P and C, and the other for communications between P and D. Now, a loop runs as follows. In each iteration (also called round), P first randomly chooses one of the two flags: MIN and MAX (the choice randomly varies from one iteration to another). Each of the two child processes C and D generates a random positive integer and sends that to P via its pipe. P reads the two integers; let these be c and d. If P has chosen MIN, then the child who sent the smaller of c and d gets one point. If P has chosen MAX, then the sender of the larger of c and d gets one point. If c = d, then this round is ignored. The child process who first obtains ten points wins the game. When the game ends, P sends a user-defined signal to both C and D, and the child processes exit after handling the signal (in order to know who was the winner). After C and D exit, the parent process P exits. During each iteration of the game, P should print appropriate messages (like P's choice of the flag, the integers received from C and D, which child gets the point, the current scores of C and D) in order to let the user know how the game is going on. Name your program childsgame.c

## CODE:

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

#define MIN 0
#define MAX 1
#define BUFSIZE 10

/* The signal handler for the child process C, signal passes message of its victory
or defeat */
void CSigHandler (int sig)
{
    if (sig == SIGUSR1)
    {
        printf("C says I have won\n");
    }
    else if (sig == SIGUSR2)
    {
        printf("C says I have lost\n");
    }
    exit(0);
}
```

```

/* The signal handler for the child process D, signal passes message of its victory
or defeat */
void DSigHandler (int sig)
{
    if (sig == SIGUSR1)
    {
        printf("D says I have won\n");
    }
    else if (sig == SIGUSR2)
    {
        printf("D says I have lost\n");
    }
    exit(0);
}

/* Main function */
int main ()
{
    srand((unsigned int)time(NULL)); //seeding time
    int pidC, pidD; //pid for child processes
    int fd1[2], fd2[2]; //for pipes

    char number1[10], number2[10]; //to pass numbers in string format

    if(pipe(fd1)==-1) //if pipe could not be created
    {
        printf("Could not make pipe between P and C\n");
        exit(0);
    }
    if(pipe(fd2)==-1) //if pipe could not be created
    {
        printf("Could not make pipe between P and D\n");
        exit(0);
    }

    if ((pidC=fork()) == 0)//Child C
    {
        srand(pidC); //seeding
        //block fd2
        close(fd2[0]);
        close(fd2[1]);

        close(fd1[0]); //close read end

        signal(SIGUSR1, CSigHandler); // Register SIGUSR1 handler */
        signal(SIGUSR2, CSigHandler); // Register SIGUSR2 handler */

        while(1)
        {
            int n = rand()%100; //random number generated to be passed

            sprintf(number1, "%d", n); //writing to string
            write(fd1[1], number1, BUFSIZE); //writing to pipe
            sleep(1);
        }
    }
    else if((pidD = fork()) == 0)//Child D
    {
        //block fd1
        close(fd1[0]);
        close(fd1[1]);
    }
}

```

```

close(fd2[0]); //close read end

signal(SIGUSR1, DSigHandler);          /* Register SIGUSR1 handler */
signal(SIGUSR2, DSigHandler);          /* Register SIGUSR2 handler */

while(1)
{
    int n = rand()%100;                  //random number generated to be passed

    sprintf(number2, "%d", n);           //writing to string
    write(fd2[1], number2, BUFSIZE);     //writing to pipe
    sleep(1);
}
}
else
{
    close(fd1[1]); //close write end
    close(fd2[1]); //close write end

    int scoreC = 0, scoreD = 0, round = 1;

    //Loop for the matches between C and D, while one does not reach a score of
10 while(scoreC!=10 && scoreD != 10)
    {
        int flag = rand()%2;             //random flag is selected
        printf("\nRound %d : \nChoice of flag is ", round);
        round++;
        if (flag==MIN)
        {
            printf("MIN\n");
        }
        else
        {
            printf("MAX\n");
        }

        read(fd1[0], number1, BUFSIZE);   //number from child C is read
        int numberC = atoi(number1);
        read(fd2[0], number2, BUFSIZE);   //number from child D is read
        int numberD = atoi(number2);

        printf("Number received from C is %d\n", numberC);
        printf("Number received from D is %d\n", numberD);

        if(flag == MIN)
        {
            if(numberC < numberD)
            {
                ++scoreC;
                printf("C gets the point\n");
            }
            else if(numberC > numberD)
            {
                ++scoreD;
                printf("D gets the point\n");
            }
        }
        else
        {
            if(numberC > numberD)
            {
                ++scoreC;
                printf("C gets the point\n");
            }
        }
    }
}

```

```

        }
        else if(numberC < numberD)
        {
            ++scoreD;
            printf("D gets the point\n");
        }
    }

    printf("Score of C : %d\n", scoreC);
    printf("Score of D : %d\n", scoreD);
}
if(scoreC==10)
{
    printf("\nP says C has won\n");
    kill(pidC, SIGUSR1);           //process C killed
    kill(pidD, SIGUSR2);           //process D killed
}
else
{
    printf("\nP says D has won\n");
    kill(pidD, SIGUSR1);           //process D killed
    kill(pidC, SIGUSR2);           //process C killed
}

exit(0);                          //exiting
}
}

```

## OUTPUT: (part-1)

```

fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q6 Q6.c
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q6

```

```

Round 1 :
Choice of flag is MAX
Number received from C is 83
Number received from D is 67
C gets the point
Score of C : 1
Score of D : 0

```

```

Round 2 :
Choice of flag is MAX
Number received from C is 86
Number received from D is 23
C gets the point
Score of C : 2
Score of D : 0

```

```

Round 3 :
Choice of flag is MAX
Number received from C is 77
Number received from D is 69
C gets the point
Score of C : 3
Score of D : 0

```

```

Round 4 :
Choice of flag is MAX
Number received from C is 15
Number received from D is 83
D gets the point
Score of C : 3
Score of D : 1

```

```

Round 5 :
Choice of flag is MAX
Number received from C is 93
Number received from D is 23

```



## (part-2)

```
fsrt@fsrt-VirtualBox: ~/Desktop/OS
File Edit View Search Terminal Help
Choice of flag is MAX
Number received from C is 93
Number received from D is 23
C gets the point
Score of C : 4
Score of D : 1

Round 6 :
Choice of flag is MAX
Number received from C is 35
Number received from D is 75
D gets the point
Score of C : 4
Score of D : 2

Round 7 :
Choice of flag is MAX
Number received from C is 86
Number received from D is 51
C gets the point
Score of C : 5
Score of D : 2

Round 8 :
Choice of flag is MIN
Number received from C is 92
Number received from D is 30
D gets the point
Score of C : 5
Score of D : 3

Round 9 :
Choice of flag is MAX
Number received from C is 49
Number received from D is 83
D gets the point
Score of C : 5
Score of D : 4
```

Activate Windows  
Go to Settings to activate Windows.

## (part-3)

```
fsrt@fsrt-VirtualBox: ~/Desktop/OS
File Edit View Search Terminal Help
Score of C : 5
Score of D : 4

Round 10 :
Choice of flag is MIN
Number received from C is 21
Number received from D is 74
C gets the point
Score of C : 6
Score of D : 4

Round 11 :
Choice of flag is MAX
Number received from C is 62
Number received from D is 79
D gets the point
Score of C : 6
Score of D : 5

Round 12 :
Choice of flag is MIN
Number received from C is 27
Number received from D is 2
D gets the point
Score of C : 6
Score of D : 6

Round 13 :
Choice of flag is MAX
Number received from C is 90
Number received from D is 19
C gets the point
Score of C : 7
Score of D : 6

Round 14 :
Choice of flag is MAX
Number received from C is 59
Number received from D is 9
```

Activate Windows  
Go to Settings to activate Windows.

## (part-4)

```
fsrt@fsrt-VirtualBox: ~/Desktop/OS
File Edit View Search Terminal Help
Score of C : 6
Score of D : 6

Round 13 :
Choice of flag is MAX
Number received from C is 90
Number received from D is 19
C gets the point
Score of C : 7
Score of D : 6

Round 14 :
Choice of flag is MAX
Number received from C is 59
Number received from D is 9
C gets the point
Score of C : 8
Score of D : 6

Round 15 :
Choice of flag is MAX
Number received from C is 63
Number received from D is 29
C gets the point
Score of C : 9
Score of D : 6

Round 16 :
Choice of flag is MIN
Number received from C is 26
Number received from D is 36
C gets the point
Score of C : 10
Score of D : 6

P says C has won
D says I have lost
C says I have won
fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

Activate Windows  
Go to Settings to activate Windows.

# EXPERIMENT 5

**AIM: Write a program in C to implement producer consumer problem.**

## CODE:

```
#include<stdio.h>
#include<stdlib.h>
int mutex=1,full=0,empty=3,x=0;
int main()
{
    int n;
    void producer();
    void consumer();
    int wait(int);
    int signal(int);
    printf("\n1.Producer\n2.Consumer\n3.Exit");
    while(1)
    {
        printf("\nEnter your choice:");
        scanf("%d",&n);
        switch(n)
        {
            case 1: if((mutex==1)&&(empty!=0))
                producer();
            else
                printf("Buffer is full!!");
                break;
            case 2: if((mutex==1)&&(full!=0))
                consumer();
            else
                printf("Buffer is empty!!");
                break;
            case 3:exit(0);
            break;
        }
    }
    return 0;
}

int wait(int s)
{
    return (--s);
}

int signal(int s)
{
    return(++s);
}

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

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

## OUTPUT:

```
fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q8_consumer Q8_consumer.c  
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q8_consumer  
  
1.Producer  
2.Consumer  
3.Exit  
Enter your choice:1  
  
Producer produces the item 1  
Enter your choice:2  
  
Consumer consumes item 1  
Enter your choice:2  
Buffer is empty!!  
Enter your choice:1  
  
Producer produces the item 1  
Enter your choice:2  
  
Consumer consumes item 1  
Enter your choice:2  
Buffer is empty!!  
Enter your choice:3  
fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

# EXPERIMENT 6

**AIM:** Write a program in C to perform fork() system call to create child processes.

## CODE:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{
    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

## OUTPUT:

```
fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q10 Q10.c
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q10
Hello world!
Hello world!
fsrt@fsrt-VirtualBox:~/Desktop/OS$
```

# EXPERIMENT 7

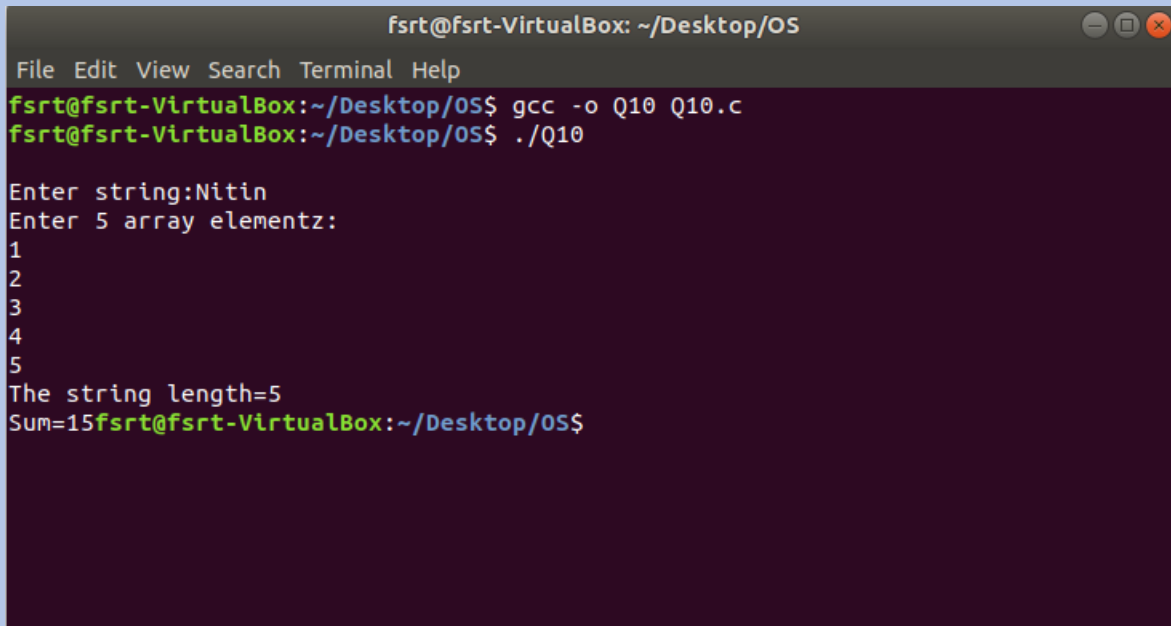
**AIM: Write a program in C to perform fork() system call to create child processes.**

## CODE:

```
#include<stdio.h>
#include<unistd.h>
#include<string.h>
void main()
{
    int pid[2],pid1[2],pid2[2],pid3[2],pid4[2];
    int a[20],i,l,s=0;
    char str[20];
    pipe(pid);
    pipe(pid1);
    pipe(pid2);
    pipe(pid3);
    pipe(pid4);
    if(fork()==0)
    {
        sleep(5);
        close(pid[1]);
        read(pid[0],str,sizeof(str));
        for(i=0,l=0;str[i]!='\0';i++)
            l=l+1;
        close(pid3[0]);
        write(pid3[1],&l,sizeof(l));
        sleep(6);
        printf("Enter %d array elementz:",l);
        for(i=0;i<l;i++)
            scanf("%d",&a[i]);
        close(pid1[0]);
        write(pid1[1],a,sizeof(a));
        close(pid4[0]);
        write(pid4[1],&l,sizeof(l));
    }
    else if(fork()==0)
    {
        sleep(2);
        close(pid1[1]);
        close(pid4[1]);
        read(pid4[0],&l,sizeof(l));
        read(pid1[0],a,sizeof(a));
        for(i=0;i<l;i++)
            s=s+a[i];
        close(pid2[0]);
        write(pid2[1],&s,sizeof(s));
    }
    else
    {
        printf("\nEnter string:");
        scanf("%s",str);
        close(pid[0]);
        write(pid[1],str,sizeof(str));
        sleep(7);
        close(pid3[1]);
    }
}
```

```
read(pid3[0], &l, sizeof(l));  
printf("\nThe string length=%d", l);  
sleep(8);  
close(pid2[1]);  
read(pid2[0], &s, sizeof(s));  
printf("\nSum=%d", s);  
}  
}
```

## OUTPUT:



```
fsrt@fsrt-VirtualBox: ~/Desktop/OS  
File Edit View Search Terminal Help  
fsrt@fsrt-VirtualBox:~/Desktop/OS$ gcc -o Q10 Q10.c  
fsrt@fsrt-VirtualBox:~/Desktop/OS$ ./Q10  
  
Enter string:Nitin  
Enter 5 array elementz:  
1  
2  
3  
4  
5  
The string length=5  
Sum=15fsrt@fsrt-VirtualBox:~/Desktop/OS$
```