OPERATNG SYSTEMS

LIST OF PROGRAMS(1-10)

1.Create a new process by invoking the appropriate system call. Get the process identifier of the currently running process and its respective parent using system calls and display the same using a C program.

Aim : To write a c program for currently running process and its respective parent using system calls

Algorithm :

Step 1 : Include heading files like include<stdio.h> and include<unistd.h>

Step 2 : print process id %d\n, using get pid

Step 3 : print parent process id %d\n, using get pid

Step 4 : Return 0

Code :

```
#include<stdio.h>
#include<unistd.h>
int main()
{
  printf("Process ID: %d\n", getpid() );
  printf("Parent Process ID: %d\n", getpid() );
  return 0;
}
```

Output :

```
Process ID: 4376
Parent Process ID: 4376

--------------------------------
Process exited after 0.06683 seconds with return value 0
Press any key to continue . . .
```

Result: By running the code output has been verified successfully

2. Identify the system calls to copy the content of one file to another and illustrate the same using a C program.

Aim: To write a c program to copy the content of one file to another file

Algorithm :

Step 1: Include heading files like include<stdio.h> and include <stdlib.h>

Step 2: Give file, “fptr1”,”fptr2” and char as [100] and share the data open for reading.

Step 3: Store data as “connect” open file % in the file name.

Step 4: Store data enter the file name to open for writing.

Step 5: By using while loop data as fptr 1 and fptr 2.

Step 6: Contents are copied to %s file name and close (fptr 1) close (fptr 2).

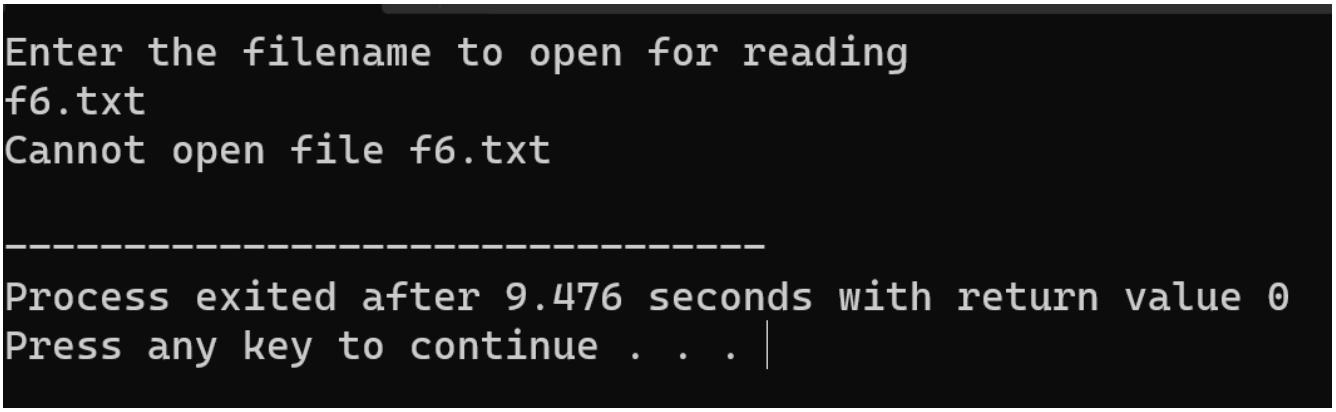
Step 7: The file 1 data has been copied to file 2.

Code:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fptr1, *fptr2;
    char filename[100], c;
    printf("Enter the filename to open for reading \n");
    scanf("%s", filename);
    fptr1 = fopen(filename, "r");
    if (fptr1 == NULL)
    {
    printf("Cannot open file %s \n", filename);
    exit(0);
    }
    printf("Enter the filename to open for writing \n");
    scanf("%s", filename);
    fptr2 = fopen(filename, "w");
    if (fptr2 == NULL)
    {
    printf("Cannot open file %s \n", filename);
    exit(0);
    }
    c = fgetc(fptr1);
    while (c != EOF)
```

```
    {
    fputc(c, fptr2);
    c = fgetc(fptr1);
    }
    printf("\nContents copied to %s", filename);
    fclose(fptr1);
    fclose(fptr2);
    return 0;
}
```

Output :



Result: By running the code output has been verified successfully

3. Design a CPU scheduling program with C using First Come First Served technique with the following considerations.

a. All processes are activated at time 0.

b. Assume that no process waits on I/O devices.

Aim: To design a CPU scheduling program using c code for the first come first served technique.

Algorithm:

Step 1: Include heading files like include<stdio.h> and include <stdlib.h>

Step 2: giving int values, Burst time, waiting time, total arriving time, float, waiting average, and total average time.

Step 3: Enter the number of processes to store the data using a for loop.

Step 4: enter the burst time for the process %d…..”

Step 5: “t process \t burst time \t waiting time \t turn around time \n”.

Step 6: store the data (\n average waiting time__%8”, waiting average(n); print(“\n average turn around time _ _%8”) as total average time.

Code:

```
#include <stdio.h>
int main()
{
int A[100][4];
int i, j, n, total = 0, index, temp;
float avg_wt, avg_tat;
printf("Enter number of process: ");
scanf("%d", &n);
printf("Enter Burst Time:\n");
for (i = 0; i < n; i++) {
        printf("P%d: ", i + 1);
        scanf("%d", &A[i][1]);
        A[i][0] = i + 1;
}
for (i = 0; i < n; i++) {
        index = i;
        for (j = i + 1; j < n; j++)
                if (A[j][1] < A[index][1])
                        index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;

        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
}
A[0][2] = 0;
for (i = 1; i < n; i++) {
        A[i][2] = 0;
        for (j = 0; j < i; j++)
                A[i][2] += A[j][1];
        total += A[i][2];
}
avg_wt = (float)total / n;
```

```
total = 0;
printf("P	BT	WT	TAT\n");

for (i = 0; i < n; i++) {
	A[i][3] = A[i][1] + A[i][2];
	total += A[i][3];
	printf("P%d	%d	%d	%d\n", A[i][0],A[i][1], A[i][2], A[i][3]);
}
avg_tat = (float)total / n;
printf("Average Waiting Time= %f", avg_wt);
printf("\nAverage Turnaround Time= %f", avg_tat);
}
```

Output:

```
Enter number of process: 4
Enter Burst Time:
P1: 12
P2: 14
P3: 15
P4: 16
P	BT	WT	TAT
P1	12	0	12
P2	14	12	26
P3	15	26	41
P4	16	41	57
Average Waiting Time= 19.750000
Average Turnaround Time= 34.000000
--------------------------------
Process exited after 17.9 seconds with return value 0
Press any key to continue . . .
```

Result: Hence the program is executed successfully.

4. Construct a scheduling program with C that selects the waiting process with the smallest execution time to execute next.

Aim: To write a scheduling program using c code that selects the waiting process with the smallest execution time.

Algorithm:

Step 1: Include heading files like include<stdio.h> and give burst time waiting time and total arriving time.

Step 2: Enter the number of processes and burst time using for loop and giving iteration values.

Step 3: Initialize the waiting time 0 and give an increment as it is in the for loop.

Step 4: Average waiting time in float total /n. total =0

Step 5: store the data n process burst time t waiting time, Turn-Around time, giving for (i=0;i<n;i++).

Step 6: average TAT(float) total /n.

Step 7: print(average waiting time)

Print (average turnaround time)

Code:

```
#include<stdio.h>
 int 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;
   }
   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;
    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("nProcesst    Burst Time    tWaiting TimetTurnaround Time");
    for(i=0;i<n;i++)
    {
        tat[i]=bt[i]+wt[i];
        total+=tat[i];
        printf("np%dtt  %dtt    %dttt%d",p[i],bt[i],wt[i],tat[i]);
    }
    avg_tat=(float)total/n;
    printf("nnAverage Waiting Time=%f",avg_wt);
    printf("nAverage Turnaround Time=%fn",avg_tat);
}
```

Output:

```
Enter number of process:3
nEnter Burst Time:
p1:45
p2:32
p3:18
nProcesst    Burst Time    tWaiting TimetTurnaround Time
np3tt  18tt    0ttt18np2tt  32tt    18ttt50np1tt  45tt    50ttt95nnAverage Waiting Time=22.666666nAverage Turnaround Time=54.333332n
--------------------------------
Process exited after 8.49 seconds with return value 0
Press any key to continue . . .
```

Result: Hence the program is executed successfully.

5. Construct a scheduling program with C that selects the waiting process with the highest priority to execute next.

Aim: To write a c code for a scheduling program that selects the waiting process with the highest priority to execute next.

Algorithm:

Step 1: including heading files like #include<stdio.h>

Step 2: giving int values burst time, waiting time, and turnaround time in priority.

Step 3: Number of processes and give ASCII number = 65 floating average waiting time and average turn around time.

Step 4: Enter the total number of processes and store the data.

Step 5: Enter the burst time average waiting time and priority.

Step 6: using for loop initializing the values and increment.

Step 7: print the average waiting time and average turnaround time.

Code:

```
#include<stdio.h>
struct priority_scheduling {
  char process_name;
  int burst_time;
  int waiting_time;
  int turn_around_time;
  int priority;
};
int main() {
  int number_of_process;
  int total = 0;
  struct priority_scheduling temp_process;
  int ASCII_number = 65;
  int position;
```

```
float average_waiting_time;
float average_turnaround_time;
printf("Enter the total number of Processes: ");
scanf("%d", & number_of_process);
struct priority_scheduling process[number_of_process];
printf("\nPlease Enter the  Burst Time and Priority of each process:\n");
for (int i = 0; i < number_of_process; i++) {
  process[i].process_name = (char) ASCII_number;
  printf("\nEnter the details of the process %c \n", process[i].process_name);
  printf("Enter the burst time: ");
  scanf("%d", & process[i].burst_time);
  printf("Enter the priority: ");
  scanf("%d", & process[i].priority);
  ASCII_number++;
}
for (int i = 0; i < number_of_process; i++) {
  position = i;
  for (int j = i + 1; j < number_of_process; j++) {
    if (process[j].priority > process[position].priority)
      position = j;
  }
  temp_process = process[i];
  process[i] = process[position];
  process[position] = temp_process;
}
process[0].waiting_time = 0;
for (int i = 1; i < number_of_process; i++) {
  process[i].waiting_time = 0;
  for (int j = 0; j < i; j++) {
    process[i].waiting_time += process[j].burst_time;
  }
  total += process[i].waiting_time;
}
average_waiting_time = (float) total / (float) number_of_process;
total = 0;
```

```
  printf("\n\nProcess_name \t Burst Time \t Waiting Time \t  Turnaround Time\n");
  printf("-----------------------------------------------------------\n");
  for (int i = 0; i < number_of_process; i++) {
    process[i].turn_around_time = process[i].burst_time + process[i].waiting_time;
    total += process[i].turn_around_time;
    printf("\t    %c  \t\t    %d  \t\t %d  \t\t %d", process[i].process_name, process[i].burst_time,
          process[i].waiting_time, process[i].turn_around_time);
    printf("\n-----------------------------------------------------------\n");
  }
  average_turnaround_time = (float) total / (float) number_of_process;
  printf("\n\n Average Waiting Time : %f", average_waiting_time);
  printf("\n Average Turnaround Time: %f\n", average_turnaround_time);
  return 0;
}
```

Output:

```
Enter the total number of Processes: 3

Please Enter the  Burst Time and Priority of each process:

Enter the details of the process A
Enter the burst time: 2
Enter the priority: 1

Enter the details of the process B
Enter the burst time: 10
Enter the priority: 3

Enter the details of the process C
Enter the burst time: 6
Enter the priority: 2


Process_name     Burst Time      Waiting Time     Turnaround Time
-----------------------------------------------------------
          B               10              0               10
-----------------------------------------------------------
          C               6               10              16
-----------------------------------------------------------
          A               2               16              18
-----------------------------------------------------------


 Average Waiting Time : 8.666667
 Average Turnaround Time: 14.666667
```

6. Construct a c program to implement pre-emptive priority scheduling algorithm.

Aim: To construct a c program to implement pre-emptive priority scheduling algorithm.

Algorithm:

1. Initialize the necessary data structures to store process information,including process ID, burst time, and remaining time.
2. Read the number of processes (N) from the user.
3. Read the time quantum (slice time) from the user.
4. For each process, read the following information:
5. Process ID (PID)
6. Burst Time (time required for execution)
7. Create a queue data structure to store the processes.
8. Enqueue all processes into the queue.
9. Initialize a variable current_time to 0 (representing the current time in thesimulation).
10. Initialize a variable total_waiting_time to 0.
11. While the queue is not empty, repeat the following:
    a. Dequeue a process from the front of the queue.
    b. Calculate the execution time for the process, which is the minimumof the time quantum and the remaining time for the process.
    c. Update the process's remaining time.
    d. Update current_time by adding the execution time.
    e. If the process still has remaining time, enqueue it back into thequeue.
    f. Calculate the waiting time for the process as current_time - arrivaltime, where arrival time is the time when the process was first enqueued.
    g. Add the waiting time to total_waiting_time.
12. Calculate the average waiting time as total_waiting_time / N.
13. Print the average waiting time.

Program: -

```
#include<stdio.h>
#include<conio.h>int
main()
{
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];float
   avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP;
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);printf(" Arrival
time is: \t");
scanf("%d", &at[i]);
```

```
printf(" \nBurst time is: \t");
scanf("%d",  &bt[i]); temp[i] =
bt[i];
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");for(sum=0, i = 0;
y!=0; )
{
if(temp[i] <= quant && temp[i] > 0)
{
    sum = sum + temp[i];
    temp[i] = 0;
    count=1;
    }
    else if(temp[i] > 0)
    {
        temp[i] = temp[i] - quant;sum
        = sum + quant;
    }
    if(temp[i]==0 && count==1)
    {
        y--;
        printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-
at[i]-bt[i]);
        wt = wt+sum-at[i]-bt[i];tat
        = tat+sum-at[i]; count =0;
```

```
    }
    if(i==NOP-1)
    {
        i=0;
    }
    else if(at[i+1]<=sum)
    {
        i++;
    }
    else
    {
        i=0;
    }
}
avg_wt = wt *
1.0/NOP;avg_tat =
tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```

Output:-

```
 Total number of process in the system: 3

 Enter the Arrival and Burst time of the Process[1]
 Arrival time is:       2

Burst time is:  33334

 Enter the Arrival and Burst time of the Process[2]
 Arrival time is:       23

Burst time is:  45

 Enter the Arrival and Burst time of the Process[3]
 Arrival time is:       27

Burst time is:  67
Enter the Time Quantum for the process:         9

 Process No             Burst Time              TAT             Waiting Time
Process No[2]           45                              121                     76
Process No[3]           67                              175                     108
Process No[1]           33334                           33444                   110
 Average Turn Around Time:      98.000000
 Average Waiting Time:  11246.666992
```

Result: By running the code output has been verified successfully.

7. Construct a C program to implement a non-preemptive SJF algorithm.

Aim: To write a c program to implement the non-preemptive SJF algorithm.

Algorithm:

1. Function SortByBurstTime(num, mat):
   - Sort the array 'mat' based on the burst time (mat[2]) in ascending order.
2. Function WaitingTimeTurnaroundTime(num, mat):
   - Initialize waiting time (mat[3]) for the first process to 0.
   - Iterate over processes from the second to the last:
     - Calculate waiting time (mat[3][i]) as the sum of previous waiting time and burst time of the previous process.
     - Calculate turnaround time (mat[5][i]) as the sum of waiting time and burst time.
     - Calculate waiting time for the current process (mat[4][i]) as the difference between turnaround time and burst time.
3. Function main():
   - Initialize num to the number of processes.
   - Initialize mat as a 6x3 array with process details.
   - Display "Before Arrange...".
   - Display "Process ID\tArrival Time\tBurst Time" for each process in mat.
   - Call SortByBurstTime(num, mat).
   - Call WaitingTimeTurnaroundTime(num, mat).
   - Display "Final Result...".
   - Display "Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time" for each process in mat.

Code:

```
#include<iostream>
using namespace std;
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
void arrangeArrival(int num, int mat[][3]) {
  for(int i=0; i<num; i++) {
    for(int j=0; j<num-i-1; j++) {
      if(mat[1][j] > mat[1][j+1]) {
        for(int k=0; k<5; k++) {
          swap(mat[k][j], mat[k][j+1]);
        }
```

```
        }
      }
    }
}
void completionTime(int num, int mat[][3]) {
   int temp, val;
   mat[3][0] = mat[1][0] + mat[2][0];
   mat[5][0] = mat[3][0] - mat[1][0];
   mat[4][0] = mat[5][0] - mat[2][0];
    for(int i=1; i<num; i++) {
      temp = mat[3][i-1];
      int low = mat[2][i];
      for(int j=i; j<num; j++) {
        if(temp >= mat[1][j] && low >= mat[2][j]) {
          low = mat[2][j];
          val = j;
        }
      }
      mat[3][val] = temp + mat[2][val];
      mat[5][val] = mat[3][val] - mat[1][val];
      mat[4][val] = mat[5][val] - mat[2][val];
      for(int k=0; k<6; k++) {
        swap(mat[k][val], mat[k][i]);
      }
   }
}
int main() {
   int num = 3, temp;
   int mat[6][3] = {1, 2, 3, 3, 6, 4, 2, 3, 4};
   cout<<"Before Arrange...\n";
   cout<<"Process ID\tArrival Time\tBurst Time\n";
   for(int i=0; i<num; i++) {
      cout<<mat[0][i]<<"\t\t"<<mat[1][i]<<"\t\t"<<mat[2][i]<<"\n";
   }
   arrangeArrival(num, mat);
   completionTime(num, mat);
   cout<<"Final Result...\n";
   cout<<"Process ID\tArrival Time\tBurst Time\tWaiting Time\tTurnaround Time\n";
   for(int i=0; i<num; i++) {
      cout<<mat[0][i]<<"\t\t"<<mat[1][i]<<"\t\t"<<mat[2][i]<<"\t\t"<<mat[4][i]<<"\t\t"<<mat[
5][i]<<"\n";
   }
}
```

Output:

```
Before Arrange...
Process ID      Arrival Time    Burst Time
1               3               2
2               6               3
3               4               4
Final Result...
Process ID      Arrival Time    Burst Time      Waiting Time    Turnaround Time
1               3               2               0               2
3               4               4               1               5
2               6               3               3               6

--------------------------------
Process exited after 0.1109 seconds with return value 0
Press any key to continue . . .
```

Result: By running the code output has been verified successfully.

8. Construct a C program to simulate the Round-Robin scheduling algorithm with C.

Aim: To stimulate a Round-Robin scheduling algorithm using c code.

Algorithm:

1. Input the number of processes (n), burst time for each process, and the time quantum.
2. Initialize arrays for process IDs, burst times, and remaining times.
3. Initialize variables for waiting time, turnaround time, and time.
4. Use a while loop to simulate until all processes are completed:
   a. Iterate through each process:
      i. If the process has remaining time:
         - Execute the process for the minimum of quantum or remaining time.
         - Update remaining time, waiting time, turnaround time, and current time.
5. Calculate average waiting time and average turnaround time.
6. Display the results.
7. In the main function:
   a. Input process details.
   b. Call the simulation function with the provided parameters.

8. End the program.

Code:

```
#include<stdio.h>
#include<conio.h>
int main()
```

```
{
   int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0, at[10], bt[10], temp[10];
   float avg_wt, avg_tat;
   printf(" Total number of process in the system: ");
   scanf("%d", &NOP);
   y = NOP;
for(i=0; i<NOP; i++)
{
printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i+1);
printf(" Arrival time is: \t");
scanf("%d", &at[i]);
printf(" \nBurst time is: \t");
scanf("%d", &bt[i]);
temp[i] = bt[i];
}
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
for(sum=0, i = 0; y!=0; )
{
if(temp[i] <= quant && temp[i] > 0)
{
   sum = sum + temp[i];
   temp[i] = 0;
   count=1;
   }
   else if(temp[i] > 0)
   {
      temp[i] = temp[i] - quant;
      sum = sum + quant;
   }
   if(temp[i]==0 && count==1)
```

```
    {
      y--;
      printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i], sum-at[i], sum-at[i]-
bt[i]);
      wt = wt+sum-at[i]-bt[i];
      tat = tat+sum-at[i];
      count =0;
    }
    if(i==NOP-1)
    {
      i=0;
    }
    else if(at[i+1]<=sum)
    {
      i++;
    }
    else
    {
      i=0;
    }
}
avg_wt = wt * 1.0/NOP;
avg_tat = tat * 1.0/NOP;
printf("\n Average Turn Around Time: \t%f", avg_wt);
printf("\n Average Waiting Time: \t%f", avg_tat);
getch();
}
```

Result: By running code output has verified successfully.

Output:

```
Total number of process in the system: 4

Enter the Arrival and Burst time of the Process[1]
Arrival time is:       1

Burst time is:  23

Enter the Arrival and Burst time of the Process[2]
Arrival time is:       2

Burst time is:  32

Enter the Arrival and Burst time of the Process[3]
Arrival time is:       3

Burst time is:  2

Enter the Arrival and Burst time of the Process[4]
Arrival time is:       4

Burst time is:  45
Enter the Time Quantum for the process:         5

 Process No              Burst Time              TAT             Waiting Time
Process No[3]            2                               9                       7
Process No[1]            23                              64                      41
Process No[2]            32                              85                      53
Process No[4]            45                              98                      53
 Average Turn Around Time:      38.500000
 Average Waiting Time:  64.000000
```

9. Illustrate the concept of inter-process communication using shared memory with a C program.

Aim: To illustrate the inter-process communication using shared memory with c code.

Algorithm:

Algorithm:

1. Include necessary headers.
2. Define a structure for shared data.
3. Create shared memory key using ftok().
4. Create or access the shared memory segment using shmget().
5. Attach shared memory to process using shmat().
6. Perform Inter-Process Communication using shared memory:
   a. Write/Read data to/from the shared memory.
7. Detach shared memory using shmdt().

8. Optionally, remove shared memory segment using shmctl().

9. Implement producer and consumer processes with synchronization if needed.

10. Compile and run the program.

11. End the program.

Code:

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

Output:

```vbnet
Data written to shared memory: Hello, shared memory!
```

Result: By running the code , output has verified successfully.

10. Illustrate the concept of inter-process communication using message queue with a c program

Aim: To illustrate the concept of inter-process communication using message queue with a c program.

Algorithm:

1.Import necessary libraries for message queues in C.

2.Create a message queue using msgget() system call.

3.Fork the process to create a child process for communication.

4.In the child process, send a message to the message queue using msgsnd() system call.

5.In the parent process, receive the message from the message queue using msgrcv() system call.

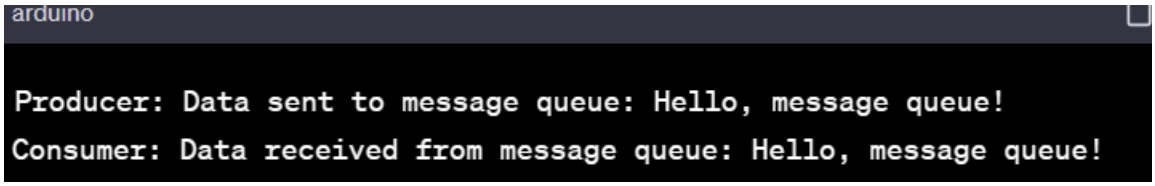
6.Display the received message.

Code:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int main() {
    int msgid = msgget(IPC_PRIVATE, 0666 | IPC_CREAT);
    msgsnd(msgid, "Hello", sizeof("Hello"), 0);
    msgrcv(msgid, "Hello", sizeof("Hello"), 0, 0);
    msgctl(msgid, IPC_RMID, NULL);
    return 0;
}
```

Output:



Result: By running the code output has verified successfully.