Ex No :4

INTER PROCESS COMMUNICATION

ALGORITHM

Step 1: Start the program.
Step 2: Define the key.
Step 3: Attach the client to the shared memory created by the server.
Step 4: Read the content from the shared memory.
Step 5: Display the content on the screen.
Step 6: Stop

PROGRAM:

MESSAGE QUEUE FOR WRITER PROCESS

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#define MAX 10
struct mesg_buffer {
        long mesg_type;
        char mesg_text[100];
} message;
int main()
{
        key_t key;
        int msgid;
        key = ftok("progfile", 65);
        msgid = msgget(key, 0666 | IPC_CREAT);
        message.mesg_type = 1;
        printf("Write Data : ");
        fgets(message.mesg_text,MAX,stdin);
        msgsnd(msgid, &message, sizeof(message), 0);
        printf("Data send is : %s \n", message.mesg_text);
        return 0;
}
```

MESSAGE QUEUE FOR READER PROCESS

```
#include <stdio.h>
#include <sys/ipc.h>
#include <sys/msg.h>
struct mesg_buffer {
        long mesg_type;
        char mesg_text[100];
} message;
int main()
{
        key_t key;
        int msgid;
```

```
        key = ftok("progfile", 65);
        msgid = msgget(key, 0666 | IPC_CREAT);
        msgrcv(msgid, &message, sizeof(message), 1, 0);
        printf("Data Received is : %s \n", message.mesg_text);
        msgctl(msgid, IPC_RMID, NULL);
        return 0;
    }
```

OUTPUT:

RESULT:

Ex No :5

MUTUAL EXCLUSION

AIM:

To write a C-program to implement the producer – consumer problem using semaphores (Mutual Exclusion).

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the required variables.
Step 3: Initialize the buffer size and get maximum item you want to produce.
Step 4: Get the option, which you want to do either producer, consumer or exit from the operation.
Step 5: If you select the producer, check the buffer size if it is full the producer should not produce the item or otherwise produce the item and increase the value buffer size.
Step 6: If you select the consumer, check the buffer size if it is empty the consumer should not consume the item or otherwise consume the item and decrease the value of buffer size.
Step 7: If you select exit come out of the program.
Step 8: Stop the program.

PROGRAM:

```
#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 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:

RESULT:

Ex No :6

BANKERS ALGORITHM FOR DEADLOCK AVOIDANCE

AIM:

To write a C program to implement banker‟s algorithm for deadlock avoidance.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the memory for the process.
Step 3: Read the number of process, resources, allocation matrix and available matrix.
Step 4: Compare each and every process using the banker‟s algorithm.
Step 5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process
Step 6: Produce the result of state of process
Step 7: Stop the program

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int p, r, i, j, process, count;
  int completed[20], Max[20][20], alloc[20][20], need[20][20], safeSequence[20],
   avail[20];
 count = 0;
printf("Enter the no of Processes : ");
scanf("%d", &p);
for(i = 0; i< p; i++)
        completed[i] = 0;
printf("\n\nEnter the no of Resources : ");
scanf("%d", &r);
printf("\n\nEnter the Max Matrix for each Process : ");
for(i = 0; i < p; i++)
{
         printf("\nFor Process %d : ", i + 1);
         for(j = 0; j < r; j++)
         scanf("%d", &Max[i][j]);
}
printf("\n\nEnter the allocation for each Process : ");
        for(i = 0; i < p; i++)
 {
        printf("\nFor Process %d : ",i + 1);
        for(j = 0; j < r; j++)
        scanf("%d", &alloc[i][j]);
 }
printf("\n\nEnter the Available Resources : ");
```

```
for(i = 0; i < r; i++)
    scanf("%d", &avail[i]);
for(i = 0; i < p; i++)
    for(j = 0; j < r; j++)
        need[i][j] = Max[i][j] - alloc[i][j];
    do
    {
        printf("\n Max Matrix:\tAllocation Matrix:\n");
        for(i = 0; i < p; i++)
        {
            for( j = 0; j < r; j++)
                printf("%d ", Max[i][j]);
            printf("\t\t");
            for( j = 0; j < r; j++)
                printf("%d ", alloc[i][j]);
            printf("\n");
        }
        process = -1;
        for(i = 0; i < p; i++)
        {
            if(completed[i] == 0)
            {
                process = i ;
                for(j = 0; j < r; j++)
                {
                    if(avail[j] < need[i][j])
                    {
                        process = -1;
                        break;
                    }
                }
            }
            if(process != -1)
                break;
        }
        if(process != -1)
        {
            printf("\nProcess %d runs to Completion!", process + 1);
            safeSequence[count] = process + 1;
            count++;
            for(j = 0; j < r; j++)
            {
                avail[j] += alloc[process][j];
                alloc[process][j] = 0;
                Max[process][j] = 0;
                completed[process] = 1;
            }
        }
```

```
        }
        while(count != p && process != -1);
        if(count == p)
        {
            printf("\nThe system is in a Safe State!!\n");
            printf("Safe Sequence : < ");
            for( i = 0; i < p; i++)
                printf("%d ", safeSequence[i]);
            printf(">\n");
        }
        else
            printf("\nThe system is in an Unsafe State!!");
}
```

OUTPUT:

RESULT:

Ex No :7

ALGORITHM FOR DEADLOCK DETECTION

AIM:

To write a C program to implement algorithm for deadlock detection.

ALGORITHM:

Step 1: Start the program.
Step 2: Declare the memory for the process.
Step 3: Read the number of process, resources, allocation matrix and available matrix.
Step 4: Compare each and every process using the banker‟s algorithm.
Step 5: If the process is in safe state then it is a not a deadlock process otherwise it is a deadlock process
Step 6: Produce the result of state of process
Step 7: Stop the program

PROGRAM:

```
#include<stdio.h>
static int mark[20];
int i,j,np,nr;
int main()
{
        alloc[10][10],request[10][10],avail[10],r[10],w[10];
        printf("\nEnter the no of Process: ");
        scanf("%d",&np);
        printf("\nEnter the no of Resources: ");
        scanf("%d",&nr);
        for(i=0;i<nr;i++)
       {
              printf("\nTotal Amount of the Resource R%d: ",i+1);
              scanf("%d",&r[i]);
       }
       printf("\nEnter the Request Matrix:");
       for(i=0;i<np;i++)
       for(j=0;j<nr;j++)
       scanf("%d",&request[i][j]);
       printf("\nEnter the Allocation Matrix:");
       for(i=0;i<np;i++)
       for(j=0;j<nr;j++)
       scanf("%d",&alloc[i][j]);
       for(j=0;j<nr;j++)
       {
              avail[j]=r[j];
              for(i=0;i<np;i++)
       {
              avail[j]-=alloc[i][j];
       }
       }
```

```
for(i=0;i<np;i++)
{
        int count=0;
        for(j=0;j<nr;j++)
 {
         if(alloc[i][j]==0)
          count++;
           else
          break;
}
 if(count==nr)
 mark[i]=1;
}
for(j=0;j<nr;j++)
 w[j]=avail[j];
for(i=0;i<np;i++)
{
        int canbeprocessed=0;
         if(mark[i]!=1)
        {
                 for(j=0;j<nr;j++)
                {
                 if(request[i][j]<=w[j])
                 canbeprocessed=1;
        else
         {
                canbeprocessed=0;
                break;
         }
        }
if(canbeprocessed)
{
        mark[i]=1;
        for(j=0;j<nr;j++)
        w[j]+=alloc[i][j];
}
}
}
int deadlock=0;
for(i=0;i<np;i++)
if(mark[i]!=1)
deadlock=1;
if(deadlock)
printf("\n Deadlock Detected");
else
printf("\n No Deadlock Possible");
}
```

OUTPUT:

RESULT: