Experiment 5:

Subject: CSL403 Operating System Lab

NAME : GINI CHACKO

ROLL : 8942

CLASS : SE COMPS B

Aim: Study Inter process Communication

Objectives: Implement and Analyse concepts of Process Synchronization and Deadlocks.

Theory:
In computer science, inter-process communication or Inter Process Communication (IPC) refers specifically to the mechanisms an operating system provides to allow processes it manages to share data. Typically, applications can use IPC categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing.

These are the methods in IPC:

1. Pipes (Same Process) –
This allows flow of data in one direction only. Analogous to simplex systems(Keyboard). Data from the output is usually buffered until input process receives it which must have a common origin.

2. Names Pipes (Different Processes) –
This is a pipe with a specific name it can be used in processes that don't have a shared common process origin. E.g. is FIFO where the details written to a pipe is first named.

3. Message Queuing –
This allows messages to be passed between processes using either a single queue or several message queue. This is managed by system kernel these messages are coordinated using an API.

4. Semaphores –
This is used in solving problems associated with synchronization and to avoid race condition. These are integer values which are greater than or equal to 0.

5. Shared memory –
This allows the interchange of data through a defined area of memory. Semaphore values have to be obtained before data can get access to shared memory.

6. Sockets –
This method is mostly used to communicate over a network between a client and a server. It allows for a standard connection which is computer and OS independent.

Problem Statement:

Write the program that connects client to server through sockets.
a. The server passes the string (two words) to client.
b. Client process splits the string.
c. Client spawns 2 child processes and passes one word to each child using pipe.
d. First child check the input is palindrome or not.
e. Second child converts the input to upper case.

Program Section:

➢ client.c

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>
#include <string.h>
void palin(char str[])
{
        int cou= 0;
        int len = strlen(str) - 1;
        while (len>cou)
        {
        if (str[cou++] != str[len--])
        {
        printf("%s is Not Palindrome\n", str);
        printf("-----------------------------------------------\n");
        return;
        }
        }
        printf("%s is palindrome\n", str);
        printf("-----------------------------------------------\n");
}
void uper(char sa[]){
  for (int i = 0; sa[i]!='\0'; i++) {
        if(sa[i] >= 'a' && sa[i] <= 'z') {
        sa[i] = sa[i] -32;
        }
  }
printf("\nString in Upper Case = %s", sa);
printf("-----------------------------------------------\n");
}
void getdat(char *s){
  char * token = strtok(s, " ");
  char *a[10];
char * mm;
 char * ll;
  int c=0;
```

```
   //printf( " %s\n", token );
printf("1");
   for(int i =0;token != NULL;i++){
          a[i] = token;
          token = strtok(NULL, " ");
   }
palin(a[0]);
uper(a[1]);
          //printf( " %s\n",a[0] );
}
   void cre(char *q){
int ass[2];
          if(pipe(ass)==-1){
          printf("error pipe");
}
printf("creating fork\n");
printf("2");
int f = fork();
  if(f == 0){
printf("child process\n");
   write(ass[1],&q,sizeof(q));
printf("\nchild process completed \t%s\n",q);
printf("-----------------------------------------------\n");
   close(ass[1]);
   }
  else{
  char hm[20];
printf("parent process");
  read(ass[0],hm,sizeof(q));
printf(".....%s",hm);

getdat(hm);
      }
}
int main(int argc, char const *argv[])
{
    int sock = 0, val;
struct sockaddr_in serv_addr;
    char buffer[512] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
          printf("\n Socket creation error \n");
           return -1;
    }
printf("-----------------------------------------------\n");
printf("Socket created\n");
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(8080);
     if(inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)<=0)
    {
```

```
        printf("\nInvalid address/ Address not supported \n");
         return -1;
    }
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
         return -1;
    }
printf("Connected Sucessfully\n");
val = read( sock , buffer, 512);
printf("Received from server :%s\n",buffer );
printf("-----------------------------------------------\n");
    //cre(buffer);
    //char bu[] = buffer;
int ass[2];
        if(pipe(ass)==-1){
        printf("error pipe");
}
printf("creating fork\n");
  int f = fork();
printf("2");
  if(f == 0){
printf("child process\n");
   write(ass[1],&buffer,sizeof(buffer));
printf("\nchild process completed %s\n",buffer);
   close(ass[1]);
   }
  else{
  char hm[20];
printf("parent process");
  read(ass[0],hm,sizeof(buffer));
printf(".....%s",hm);
printf("-----------------------------------------------\n");
getdat(hm);
     }
printf("\n");printf("\n");printf("\n");
printf("-----------------------------------------------\n");
    return 0;
}
```

➢ server.c

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
```

```
int main(int argc, char const *argv[])
{    int ser_soc,soc,val;
struct sockaddr_in address;
   int opt = 1;
int addrlen = sizeof(address);
   char buffer[512] = {0};
ser_soc = socket(AF_INET, SOCK_STREAM, 0);

   if (ser_soc == 0)
   {      printf("ERROR creating socket\n");
     }
printf("socket created\n");
   if (setsockopt(ser_soc, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,&opt,
sizeof(opt)))
   {
perror("setsockopt");
           exit(EXIT_FAILURE);
   }
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons( 8080 );
printf("port added\n");
       if (bind(ser_soc, (struct sockaddr *)&address,sizeof(address))<0)
   {
          perror("bind failed");
           exit(EXIT_FAILURE);
   }
   if (listen(ser_soc, 3) < 0)
   {
          printf("ERROR lisiting socket\n");
   }
printf("Listing to socket\n");
   if ((soc = accept(ser_soc, (struct sockaddr *)&address,
                                    (socklen_t*)&addrlen))<0)
   {
          printf("ERROR accepting connection\n");
   }
printf("connection success\n");
   char *mes = "malayalam gini\n";

   send(soc , mes , strlen(mes) , 0 );
   //send(soc , mess , strlen(mess) , 0 );

   return 0;
}
```

Output Section :