**7. WAP to demonstrate the use of semaphore – IPC**

**Ans.**

**AIM:** A program to demonstrate the use of semaphore - IPC

**DESCRIPTION:**

Semaphore is an integer variable which is accessed or modified by using two atomic operations: *wait()* and *signal()*. In C program the corresponding operations are *sem\_wait()* and *sem\_post()*. Here, we write a Program for Process Synchronization using Semaphores to understand the implementation of *sem\_wait()* and *sem\_signal()* to avoid a [race condition](https://dextutor.com/race-condition/).

**CODE:**

#include<pthread.h>

#include<stdio.h>

#include<semaphore.h>

#include<unistd.h>

void \*fun1();

 void \*fun2();

 int shared=1; //shared variable

 sem\_t s; //semaphore variable

 int main()

 {

 sem\_init(&s,0,1); //initialize semaphore variable - 1st argument is address of variable, 2nd is number of processes sharing semaphore, 3rd argument is the initial value of semaphore variable

 pthread\_t thread1, thread2;

 pthread\_create(&thread1, NULL, fun1, NULL);

 pthread\_create(&thread2, NULL, fun2, NULL);

 pthread\_join(thread1, NULL);

 pthread\_join(thread2,NULL);

 printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable

 }

void \*fun1()

{

    int x;

    sem\_wait(&s); //executes wait operation on s

    x=shared;//thread1 reads value of shared variable

    printf("Thread1 reads the value as %d\n",x);

    x++;  //thread1 increments its value

    printf("Local updation by Thread1: %d\n",x);

    sleep(1); //thread1 is preempted by thread 2

    shared=x; //thread one updates the value of shared variable

    printf("Value of shared variable updated by Thread1 is: %d\n",shared);

    sem\_post(&s);

}

void \*fun2()

{

    int y;

    sem\_wait(&s);

    y=shared;//thread2 reads value of shared variable

    printf("Thread2 reads the value as %d\n",y);

    y--;  //thread2 increments its value

    printf("Local updation by Thread2: %d\n",y);

    sleep(1); //thread2 is preempted by thread 1

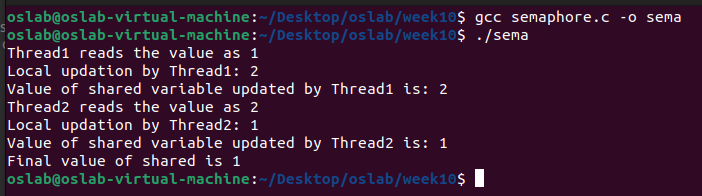
    shared=y; //thread2 updates the value of shared variable

    printf("Value of shared variable updated by Thread2 is: %d\n",shared);

    sem\_post(&s);

}

**OUTPUT:**



**8. WAP to demonstrate the use of Mutex – IPC**

**Ans.**

**AIM:** A program to demonstrate the use of mutex - IPC

**DESCRIPTION:**

A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place is called a [race condition](https://dextutor.com/critical-section-problem/). This Program uses threads to simulate race condition.

**CODE:**

#include<pthread.h>

#include<stdio.h>

#include<unistd.h>

void \*fun1();

void \*fun2();

int shared=1; //shared variable

int main()

 {

 pthread\_t thread1, thread2;

 pthread\_create(&thread1, NULL, fun1, NULL);

 pthread\_create(&thread2, NULL, fun2, NULL);

 pthread\_join(thread1, NULL);

 pthread\_join(thread2,NULL);

 printf("Final value of shared is %d\n",shared); //prints the last updated value of shared variable

 }

void \*fun1()

{

    int x;

    x=shared;//thread one reads value of shared variable

    printf("Thread1 reads the value of shared variable as %d\n",x);

    x++;  //thread one increments its value

    printf("Local updation by Thread1: %d\n",x);

    sleep(1);  //thread one is preempted by thread 2

    shared=x; //thread one updates the value of shared variable

    printf("Value of shared variable updated by Thread1 is: %d\n",shared);

}

void \*fun2()

 {

     int y;

     y=shared;//thread two reads value of shared variable

     printf("Thread2 reads the value as %d\n",y);

     y--;  //thread two increments its value

     printf("Local updation by Thread2: %d\n",y);

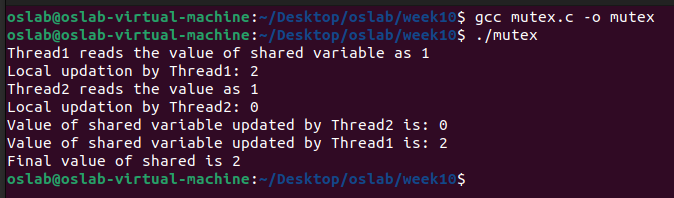
     sleep(1); //thread two is preempted by thread 1

     shared=y; //thread one updates the value of shared variable

     printf("Value of shared variable updated by Thread2 is: %d\n",shared);

 }

**OUTPUT:**



**9. WAP to demonstrate the use of message queue – IPC**

**Ans.**

**AIM:**  A program to demonstrate the use of message queue - IPC

**DESCRIPTION:**

Program for IPC using Message queues are almost similar to named pipes with the exception that they do not require the opening and closing of pipes. But, they face one similar problem like named pipes; blocking on full pipes. Message queues send blocks of data from one process to another. Each block of data is considered to have a type. There is an upper limit on the maximum size of each block and also a limit on the maximum total size of all blocks on all queues in the system.

**CODE:**

#include<stdlib.h>

 #include<stdio.h>

 #include<string.h>

 #include<unistd.h>

 #include<sys/types.h>

 #include<sys/ipc.h>

 #include<sys/msg.h>

 #define MAX\_TEXT 512   //maximum length of the message that can be sent allowed

 struct my\_msg{

    long int msg\_type;

    char some\_text[MAX\_TEXT];

 };

 int main()

 {

    int running=1;

    int msgid;

    struct my\_msg some\_data;

    char buffer[50]; //array to store user input

    msgid=msgget((key\_t)14534,0666|IPC\_CREAT);

    if (msgid == -1) // -1 means the message queue is not created

    {

        printf("Error in creating queue\n");

        exit(0);

    }

    while(running)

    {

        printf("Enter some text:\n");

        fgets(buffer,50,stdin);

        some\_data.msg\_type=1;

        strcpy(some\_data.some\_text,buffer);

        if(msgsnd(msgid,(void \*)&some\_data, MAX\_TEXT,0)==-1) // msgsnd returns -1 if the message is not sent

        {

            printf("Msg not sent\n");

        }

        if(strncmp(buffer,"end",3)==0)

        {

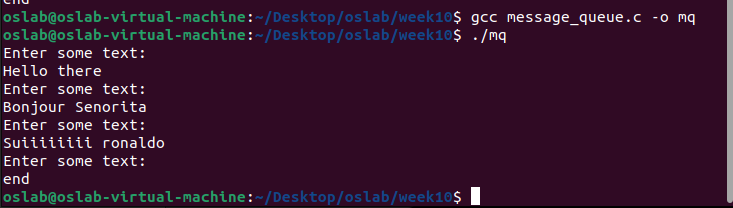
            running=0;

        }

    }

 }

**OUTPUT:**



**10. WAP to demonstrate the use of socket – IPC**

**Ans.**

**AIM:** A program to demonstrate the use of socket - IPC

**DESCRIPTION:**

A socket is **a bi-directional data transfer mechanism**. They are used to transfer data between two processes. The two processes can be running on the same system as Unix-domain or loopback sockets, or on different systems as network sockets. Sockets allow communication between two different processes on the same or different machines. To be more precise, it's a way to talk to other computers using standard Unix file descriptors. In Unix, every I/O action is done by writing or reading a file descriptor. A file descriptor is just an integer associated with an open file and it can be a network connection, a text file, a terminal, or something else. A Unix Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. Most of the application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

**CODE:**

**SOCKET:**

#include <stdio.h>

#include <netdb.h>

#include <netinet/in.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#define MAX 80

#define PORT 8080

#define SA struct sockaddr

// Function designed for chat between client and server.

void func(int connfd)

{

    char buff[MAX];

    int n;

    // infinite loop for chat

    for (;;) {

        bzero(buff, MAX);

        // read the message from client and copy it in buffer

        read(connfd, buff, sizeof(buff));

        // print buffer which contains the client contents

        printf("From client: %s\t To client : ", buff);

        bzero(buff, MAX);

        n = 0;

        // copy server message in the buffer

        while ((buff[n++] = getchar()) != '\n')

            ;

        // and send that buffer to client

        write(connfd, buff, sizeof(buff));

        // if msg contains "Exit" then server exit and chat ended.

        if (strncmp("exit", buff, 4) == 0) {

            printf("Server Exit...\n");

            break;

        }

    }

}

// Driver function

int main()

{

    int sockfd, connfd, len;

    struct sockaddr\_in servaddr, cli;

    // socket create and verification

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sockfd == -1) {

        printf("socket creation failed...\n");

        exit(0);

    }

    else

        printf("Socket successfully created..\n");

    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

    servaddr.sin\_port = htons(PORT);

    // Binding newly created socket to given IP and verification

    if ((bind(sockfd, (SA\*)&servaddr, sizeof(servaddr))) != 0) {

        printf("socket bind failed...\n");

        exit(0);

    }

    else

        printf("Socket successfully binded..\n");

    // Now server is ready to listen and verification

    if ((listen(sockfd, 5)) != 0) {

        printf("Listen failed...\n");

        exit(0);

    }

    else

        printf("Server listening..\n");

    len = sizeof(cli);

    // Accept the data packet from client and verification

    connfd = accept(sockfd, (SA\*)&cli, &len);

    if (connfd < 0) {

        printf("server accept failed...\n");

        exit(0);

    }

    else

        printf("server accept the client...\n");

    // Function for chatting between client and server

    func(connfd);

    // After chatting close the socket

    close(sockfd);

}

**CLIENT:**

#include <arpa/inet.h> // inet\_addr()

#include <netdb.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <strings.h> // bzero()

#include <sys/socket.h>

#include <unistd.h> // read(), write(), close()

#define MAX 80

#define PORT 8080

#define SA struct sockaddr

void func(int sockfd)

{

    char buff[MAX];

    int n;

    for (;;) {

        bzero(buff, sizeof(buff));

        printf("Enter the string : ");

        n = 0;

        while ((buff[n++] = getchar()) != '\n')

            ;

        write(sockfd, buff, sizeof(buff));

        bzero(buff, sizeof(buff));

        read(sockfd, buff, sizeof(buff));

        printf("From Server : %s", buff);

        if ((strncmp(buff, "exit", 4)) == 0) {

            printf("Client Exit...\n");

            break;

        }

    }

}

int main()

{

    int sockfd, connfd;

    struct sockaddr\_in servaddr, cli;

    // socket create and verification

    sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

    if (sockfd == -1) {

        printf("socket creation failed...\n");

        exit(0);

    }

    else

        printf("Socket successfully created..\n");

    bzero(&servaddr, sizeof(servaddr));

    // assign IP, PORT

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

    servaddr.sin\_port = htons(PORT);

    // connect the client socket to server socket

    if (connect(sockfd, (SA\*)&servaddr, sizeof(servaddr))

        != 0) {

        printf("connection with the server failed...\n");

        exit(0);

    }

    else

        printf("connected to the server..\n");

    // function for chat

    func(sockfd);

    // close the socket

    close(sockfd);

}

**OUTPUT:**

