1. Develop a program in which the parent process sends an array and an element to its child processthrough a pipe; the child searches the array for the element and returns the result through a pipe.The array size should be taken as input from the user.

#include <pthread.h>

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

#define MAX 10

int main()

{

  int fd[2], i = 0;

  pipe(fd);

  pid\_t pid = fork();

   if(pid > 0) {

      wait(NULL);

      // closing the standard input

      close(0);

      // no need to use the write end of pipe here so close it

      close(fd[1]);

       // duplicating fd[0] with standard input 0

      dup(fd[0]);

      int arr[MAX];

      // n stores the total bytes read successfully

      int n = read(fd[0], arr, sizeof(arr));

      for ( i = 0;i < n/4; i++)

         // printing the array received from child process

          printf("%d ", arr[i]);

   }

  else if( pid == 0 ) {

      int arr[] = {1, 2, 3, 4, 5};

      // no need to use the read end of pipe here so close it

      close(fd[0]);

       // closing the standard output

      close(1);

      // duplicating fd[0] with standard output 1

      dup(fd[1]);

      write(1, arr, sizeof(arr));

  }

  else {

      perror("error\n"); //fork()

  }

}

1. Develop a program in which the parent process sends two matrices to its child process through a pipe and the child process returns the sum of the matrices to the parent through a pipe. The parent should print the result.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <wait.h>

#define MSGSIZE 16

char\* msg1 = "hello, world #1";

char\* msg2 = "hello, world #2"; char\* msg3 = "hello, world #3";

int main() {

char inbuf[MSGSIZE];

int pipe1[2], pipe2[2], pid, nbytes;

if (pipe(pipe1) < 0) exit(1);

if (pipe(pipe2) < 0) exit(1);

if (fork() > 0) {

// parent

int mat[3][3];

// input first matrix

printf("Parent Process:\nEnter a 3x3 matrix:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

scanf("%d", &mat[i][j]);

}

}

write(pipe1[1], mat, sizeof(mat));

// write first matrix

// input second matrix

printf("Enter another 3X3 matrix:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

scanf("%d", &mat[i][j]);

}

}

write(pipe1[1], mat, sizeof(mat));

// write second matrix

read(pipe2[0], mat, sizeof(mat));

// read matrix sum

// display the matrix sum

printf("Sum of matrices is:\n");

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

printf("%d ", mat[i][j]);

}

printf("\n");

}

wait(NULL);

// wait for child process to finish

} else {

// child

int mat\_inp[3][3], mat\_sum[3][3];

// read first matrix

read(pipe1[0], mat\_sum, sizeof(mat\_sum));

// read second matrix

read(pipe1[0], mat\_inp, sizeof(mat\_inp));

printf("\nChild process adding the matrices...\n\n");

// add matrix

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

mat\_sum[i][j] += mat\_inp[i][j];

}

}

// write matrix sum

write(pipe2[1], mat\_sum, sizeof(mat\_sum));

}

return 0;

}

1. Suppose there is exist a file and you have to read, write and update the file concurrently. Write a multithreaded program such that, there should be different threads for all different tasks and each thread access the file synchronously.
2. Write a program to implement a deadlock scenario, in which two threads are accessing two resources concurrently.

// Java program to illustrate Deadlock

// in multithreading.

class Util

{

    // Util class to sleep a thread

    static void sleep(long millis)

    {

        try

        {

            Thread.sleep(millis);

        }

        catch (InterruptedException e)

        {

            e.printStackTrace();

        }

    }

}

// This class is shared by both threads

class Shared

{

    // first synchronized method

    synchronized void test1(Shared s2)

    {

        System.out.println("test1-begin");

        Util.sleep(1000);

        // taking object lock of s2 enters

        // into test2 method

        s2.test2();

        System.out.println("test1-end");

    }

    // second synchronized method

    synchronized void test2()

    {

        System.out.println("test2-begin");

        Util.sleep(1000);

        // taking object lock of s1 enters

        // into test1 method

        System.out.println("test2-end");

    }

}

class Thread1 extends Thread

{

    private Shared s1;

    private Shared s2;

    // constructor to initialize fields

    public Thread1(Shared s1, Shared s2)

    {

        this.s1 = s1;

        this.s2 = s2;

    }

    // run method to start a thread

    @Override

    public void run()

    {

        // taking object lock of s1 enters

        // into test1 method

        s1.test1(s2);

    }

}

class Thread2 extends Thread

{

    private Shared s1;

    private Shared s2;

    // constructor to initialize fields

    public Thread2(Shared s1, Shared s2)

    {

        this.s1 = s1;

        this.s2 = s2;

    }

    // run method to start a thread

    @Override

    public void run()

    {

        // taking object lock of s2

        // enters into test2 method

        s2.test1(s1);

    }

}

public class Deadlock

{

    public static void main(String[] args)

    {

        // creating one object

        Shared s1 = new Shared();

        // creating second object

        Shared s2 = new Shared();

        // creating first thread and starting it

        Thread1 t1 = new Thread1(s1, s2);

        t1.start();

        // creating second thread and starting it

        Thread2 t2 = new Thread2(s1, s2);

        t2.start();

        // sleeping main thread

        Util.sleep(2000);

    }

}

1. Write a program to implement for deadlock avoidance using conditional locking in which two threads are accessing two resources concurrently. Note: user*pthread\_mutex\_trylock()* function for conational locking
2. Implement a client-server program in which the server accepts a string from a client and transfer the upper-case string to client using TCP socket in C.

Example: client send “abcdXjhiB” and in reply Server replies   
 “ABCDXJHIB”

TCP Server:

#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);

}

TCP 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);

}

1. Implement a client-server program in which the server accepts a string from a client and transfer the upper-case string to client using UDP socket in C. Example: client send “abcdXjhiB” and in reply Server replies   
    “ABCDXJHIB”

TCP Server:

#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);

}

TCP 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);

}

1. Implement a client-server program in which the server accepts a connection from a client and updates it own Master table by adding the client information and send the updated table to client, so client can update their own table.

Table format:

|  |  |  |
| --- | --- | --- |
| Node No | Ip Address | Port no |
| 1 | 172.31.100.36 | 2345 |
| 2 | 172.31.100.40 | 3128 |
| 3 | 172.31.100.52 | 2323 |

1. Develop a client-server program to implement a date-time server and client. Upon connection establishment, the server should send its current date, time and CPU load information to its clients.

DateServer.java

import java.net.\*;  
import java.io.\*;  
import java.util.\*;  
class DateServer  
{  
    public static void main(String args[]) throws Exception  
    {  
        ServerSocket s=new ServerSocket(5217);  
        while(true)  
        {  
            System.out.println("Waiting For Connection ...");  
            Socket soc=s.accept();  
            DataOutputStream out=new DataOutputStream(soc.getOutputStream());  
            out.writeBytes("Server Date: " + (new Date()).toString() + "\n");  
            out.close();  
            soc.close();  
        }  
    }  
}

#### DateClient.java

import java.io.\*;  
import java.net.\*;  
class DateClient  
{  
    public static void main(String args[]) throws Exception  
    {  
        Socket soc=new Socket(InetAddress.getLocalHost(),5217);          
        BufferedReader in=new BufferedReader(new InputStreamReader(soc.getInputStream()  ));  
        System.out.println(in.readLine());  
    }      
}

1. Suppose you have two TCP servers for converting a lower case string to upper case string. You have to design a load balancer server that accept lower case string from client and check for the CPU utilization of both servers. Load balancer will transfer the string to the server having less CPU utilization. The load balancer will get upper case string from server and return to the clients.
2. Write a C program that generate false load on a Mosix node. Use Mosix commands to check the loads of each node of the Mosix cluster and migrate the program to the node that has minimum load by using Mosix commands.