**DAY-25(2-12-2024)**

* **Semaphore:**

1. Semget()🡺 to create
2. Semctl🡺 to control operations
3. Semop()🡺 perform semaphore operations(lock/unlock)

* Syntax:

int semop(int semid,struct sembuf\*sops,size\_t nsops)

parameters:

* Semid🡪 semaphore id returned by the semget()
* Sops🡪 pointer to an array of struct sembuf operations to perform
* Operation -1 for lock and 1 for unlock
* Nops🡪 number of operations in an array
* On success it returns 0
* On failure it returns 1
* Steps:
* Create the shared memory id using shmget()
* Then use pointer to point that location which is returned by the shmid
* Create semaphore id using semget()
* Create a union(for single value or for the structure0 and structure(for the locking and unlocking)
* If we write sleep operation then it performs content switching

Program:

#include <stdio.h>

#include <stdlib.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <unistd.h>

#include <string.h>

#define SHM\_KEY 0x1234 // Key for shared memory

#define SEM\_KEY 0x5678 // Key for semaphore

union semun {

int val;

};

void producer() {

int shmid = shmget(SHM\_KEY, sizeof(int), IPC\_CREAT | 0666);

if (shmid < 0) {

perror("shmget");

exit(1);

}

int \*ptr = (int \*)malloc(sizeof(int));

\*ptr = 10;

\*ptr = 20;

int \*shared\_data = (int \*)shmat(shmid, NULL, 0);

if (shared\_data == (void \*)-1) {

perror("shmat");

exit(1);

}

int semid = semget(SEM\_KEY, 1, IPC\_CREAT | 0666);

if (semid < 0) {

perror("semget");

exit(1);

}

union semun sem\_union;

sem\_union.val = 1; // Initialize semaphore to 1

semctl(semid, 0, SETVAL, sem\_union);

struct sembuf sem\_lock = {0, -1, 0}; // Lock semaphore

struct sembuf sem\_unlock = {0, 1, 0}; // Unlock semaphore

for (int i = 0; i < 5; i++) {

semop(semid, &sem\_lock, 1); // Lock semaphore

\*shared\_data = i; // Write data to shared memory

printf("Producer: Wrote %d to shared memory\n", i);

semop(semid, &sem\_unlock, 1); // Unlock semaphore

sleep(3); // Simulate work

}

shmdt(shared\_data);

}

void consumer() {

int shmid = shmget(SHM\_KEY, sizeof(int), 0666);

if (shmid < 0) {

perror("shmget");

exit(1);

}

int \*shared\_data = (int \*)shmat(shmid, NULL, 0);

if (shared\_data == (void \*)-1) {

perror("shmat");

exit(1);

}

int semid = semget(SEM\_KEY, 1, 0666);

if (semid < 0) {

perror("semget");

exit(1);

}

struct sembuf sem\_lock = {0, -1, 0}; // Lock semaphore

struct sembuf sem\_unlock = {0, 1, 0}; // Unlock semaphore

for (int i = 0; i < 5; i++) {

semop(semid, &sem\_lock, 1); // Lock semaphore

printf("Consumer: Read %d from shared memory\n", \*shared\_data);

semop(semid, &sem\_unlock, 1); // Unlock semaphore

sleep(3); // Simulate work

}

shmdt(shared\_data);

}

int main(int argc, char \*argv[]) {

if (argc != 2) {

fprintf(stderr, "Usage: %s <producer|consumer>\n", argv[0]);

exit(1);

}

if (strcmp(argv[1], "producer") == 0) {

producer();

} else if (strcmp(argv[1], "consumer") == 0) {

consumer();

} else {

fprintf(stderr, "Invalid argument: %s\n", argv[1]);

exit(1);

}

return 0;

}

**SOCKET PROGRAMMING:**

* OSI MODEL(open system interconnect):

1. Physical layer
2. Data link layer 🡺 mac address
3. Network layer 🡺 IP
4. Transport layer 🡺 TCP(connection-oriented) /UDP
5. Session layer
6. presentation layer
7. application layer 🡺 https/http
8. **physical layer:**

* raw bytes of data is transmitted
* Socket programming:
* It is a way of connecting two nodes on a network to communicate with each other
* One socket is listening and other socket(node) is sender
* The server forms the listener socket while the client reaches out the server
* In socket it describes port number, Ip address and protocols

A diagram of a computer flowchart

Description automatically generated

* + Socket contains IP address ,port number and family or protocol(TCP/UDP/FTP)
  + The bind will bind the data present in the socket
  + Stages of server:

1. socket creation:

* int sockfd=socket(domain,type,protocol)
* sockfd🡺 socket descriptor(like a file handler)
* domain🡺 use AF\_LOCAL for the same host, for different host connected by IPV4 ,we use AF\_INET(address family) and AF\_INET6 for process connected by IPV6
* Type 🡺 communication type

SOCK\_STREAM:TCP(reliable and connection oriented)

SOCK\_DGRAM: UDP( unreliable and connectionless)

* Protocol: TCP🡪0

1. Socket options
2. Bind:

* Int bind(int sockfd,const struct sockaddr \*addr, socklen\_t addrlen)
* The bind function binds the socket to the address and port number specified in addr(custeom data structure)

1. **Listen(very very important) :**

* Int listen(int sockfd,int backlog)
* listen() marks the socket referred to by sockfd as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept(2).
* The sockfd argument is a file descriptor that refers to a socket of type SOCK\_STREAM or SOCK\_SEQPACKET.
* The backlog argument defines the maximum length to which the queue of pending connections for sockfd may grow. If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED or, if the underlying protocol supports retransmission, the request may be ignored so that a later reattempt at connection succeeds.
* Example if the server have the capable to establish connection with 1000 clients and if backlog is 5( the length of the queue after the queue is full) and if you want to establish for 1010 then the server can accommodate only 1000 clients and from 1001 to 1005 are there in backlog and remaining 1006 to 1010 cannot be established if the clients are going out from the server then the client present in the backlog is going to establish and the remaining client 1006 to the backlog and process is repeats

1. Accept:

* Int new\_socket=accept(int sockfd,struct sockaddr \*addr,socklen\_t \*addrelen)
* It creates the new socket id for the client from the server
* The accept() system call is used with connection-based socket types (SOCK\_STREAM, SOCK\_SEQPACKET). It extracts the first connection request on the queue of pending connections for the listening socket, sockfd, creates a new connected socket, and returns a new file descriptor referring to

that socket. The newly created socket is not in the listening state. The original socket sockfd is unaffected by this call.

* Accept knowns the clients IP address,clients family and all the things of clients not the server
* The connection is established between server and client and transfer the data

Memset is used to setting all the address values to the null

Htons🡪 host to network short 🡪 it is 16 bits

Netstat -ant in putty used to give the port number , **netstat`** stands for network statistics. It allows users to display network-related information and diagnose various networking issues.

* Stages of client:

1. Socket
2. Connect:

* Int connect(int sockfd,const struct sockddr \* addr,socklen\_t addrlen)

Program For server:

/\*

Program for tcp server

socket()

bind()

listen()

accept()

send/recv/read/write/sento/recvfrom

close()

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define MAXBUFF 1024

#define PORTNO 4003

int main()

{

int sfd = 0, retValue=0, csfd=0;

int clientAddlen = 0;

struct sockaddr\_in serv\_address, client\_address;

char msg[MAXBUFF] = {0,};

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

if(sfd < 0)

{

perror("socket() ");

exit(EXIT\_FAILURE);

}

printf("\nSocket created with sockfd : %d\n",sfd);

//reset/set address of server

memset(&serv\_address,'\0',sizeof(serv\_address));

serv\_address.sin\_family = AF\_INET;

serv\_address.sin\_port = htons(PORTNO);

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

retValue = bind(sfd, (struct sockaddr \*)&serv\_address,sizeof(serv\_address));

if(retValue < 0)

{

perror("bind()");

exit(EXIT\_FAILURE);

}

printf("\nBinded the server to the ipaddress with portno\n");

retValue = listen(sfd, 5);

if(retValue < 0)

{

perror("listen() ");

exit(EXIT\_FAILURE);

}

printf("\nListening to the clients now\n");

// while(1){

csfd = accept(sfd,(struct sockaddr \*)&client\_address,&clientAddlen);

read(csfd,msg,MAXBUFF);

printf("\nClient sent Msg: %s\n\n",msg);

memset(msg,'\0',MAXBUFF);

strcpy(msg,"====================================================\n");

strcat(msg,"\n=============Welcome to Capgemini Server=============\n");

strcat(msg,"====================================================\n");

write(csfd,msg,strlen(msg));

/\*if(csfd < 0)

{

perror("accept() ");

exit(EXIT\_FAILURE);

}

printf("\nServer: Client got a connection\n");

strcpy(msg,"====================================================\n");

strcat(msg,"\n=============Welcome to Capgemini Server=============\n");

strcat(msg,"====================================================\n");

write(csfd,msg,strlen(msg));

while(1)

{

memset(msg,'\0',MAXBUFF);

read(csfd,msg,MAXBUFF);

printf("Server: Client sent a msg: %s\n",msg);

if(strcmp(msg,"bye\n")==0)

{

close(csfd);

break;

}

}

\*/

// }

close(csfd);

close(sfd);

return 0;

}

Program for client:

/\*

Program for tcp Client

socket()

connect()

send/recv/read/write/sento/recvfrom

close()

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORTNO 4003

#define MAXBUFF 1024

int main()

{

int sfd = 0, retValue=0;

struct sockaddr\_in serv\_address;

char msg[MAXBUFF] = {0,};

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

if(sfd < 0)

{

perror("socket() ");

exit(EXIT\_FAILURE);

}

printf("\nSocket created with sockfd : %d",sfd);

//reset/set address of client

memset(&serv\_address,'\0',sizeof(serv\_address));

// setting of Server side ipaddress and port no

serv\_address.sin\_family = AF\_INET;

serv\_address.sin\_port = htons(PORTNO);

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

retValue = connect(sfd,(struct sockaddr \*)&serv\_address, sizeof(serv\_address));

if(retValue < 0)

{

perror("connect()");

exit(EXIT\_FAILURE);

}

printf("\nClient: Connected to the server\n");

strcpy(msg,"\nHi This is from Client\n");

write(sfd,msg,MAXBUFF);

memset(msg,'\0',MAXBUFF);

read(sfd,msg,MAXBUFF);

puts(msg);

/\*

printf("%s",msg);

while(1){

memset(msg,'\0',MAXBUFF);

fgets(msg,MAXBUFF,stdin);

write(sfd,msg, strlen(msg));

if(strcmp(msg,"bye\n")==0)

{

break;

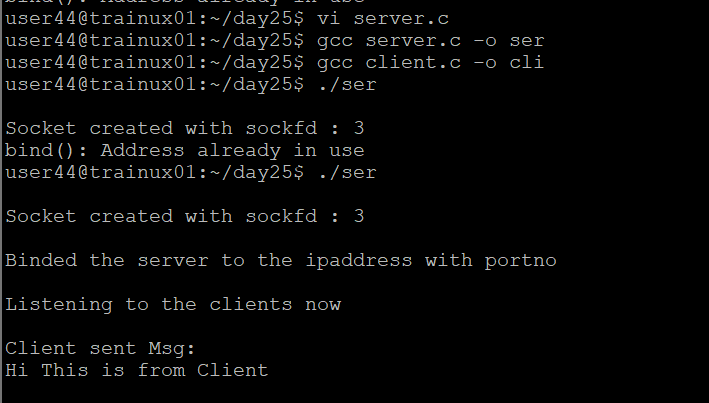
}

}\*/

close(sfd);

return 0;

}



A screenshot of a computer program

Description automatically generated

The server can establish the connection between more than one client between the data is transfer to only one client

If the server is closed we have to close the client as well

Program client version1:

/\*

Program for tcp Client

socket()

connect()

send/recv/read/write/sento/recvfrom

close()

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORTNO 4003

#define MAXBUFF 1024

int main()

{

int sfd = 0, retValue=0;

struct sockaddr\_in serv\_address;

char msg[MAXBUFF] = {0,};

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

if(sfd < 0)

{

perror("socket() ");

exit(EXIT\_FAILURE);

}

printf("\nSocket created with sockfd : %d",sfd);

//reset/set address of client

memset(&serv\_address,'\0',sizeof(serv\_address));

// setting of Server side ipaddress and port no

serv\_address.sin\_family = AF\_INET;

serv\_address.sin\_port = htons(PORTNO);

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

retValue = connect(sfd,(struct sockaddr \*)&serv\_address, sizeof(serv\_address));

if(retValue < 0)

{

perror("connect()");

exit(EXIT\_FAILURE);

}

printf("\nClient: Connected to the server\n");

strcpy(msg,"\nHi This is from Client\n");

write(sfd,msg,MAXBUFF);

memset(msg,'\0',MAXBUFF);

read(sfd,msg,MAXBUFF);

puts(msg);

close(sfd);

return 0;

}

Program for client version 2:

/\*

Program for tcp Client

socket()

connect()

send/recv/read/write/sento/recvfrom

close()

\*/

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORTNO 4003

#define MAXBUFF 1024

int main()

{

int sfd = 0, retValue=0;

struct sockaddr\_in serv\_address;

char msg[MAXBUFF] = {0,};

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

if(sfd < 0)

{

perror("socket() ");

exit(EXIT\_FAILURE);

}

printf("\nSocket created with sockfd : %d",sfd);

//reset/set address of client

memset(&serv\_address,'\0',sizeof(serv\_address));

// setting of Server side ipaddress and port no

serv\_address.sin\_family = AF\_INET;

serv\_address.sin\_port = htons(PORTNO);

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

retValue = connect(sfd,(struct sockaddr \*)&serv\_address, sizeof(serv\_address));

if(retValue < 0)

{

perror("connect()");

exit(EXIT\_FAILURE);

}

printf("\nClient: Connected to the server\n");

// strcpy(msg,"\nHi This is from Client\n");

while(1){

fgets(msg,MAXBUFF,stdin);

write(sfd,msg,MAXBUFF);

memset(msg,'\0',MAXBUFF);

}

close(sfd);

return 0;

}

Question )Send a structure from the client receive it at the server and display it in the server

Answer: program for server:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define MAXBUFF 1024

#define PORTNO 4003

// Define a structure to send

struct Data {

int id;

char name[50];

};

int main()

{

int sfd = 0, retValue=0, csfd=0;

int clientAddlen = 0;

struct sockaddr\_in serv\_address, client\_address;

struct Data receivedData;

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

if(sfd < 0)

{

perror("socket() ");

exit(EXIT\_FAILURE);

}

printf("\nSocket created with sockfd : %d\n",sfd);

// Reset server address

memset(&serv\_address, '\0', sizeof(serv\_address));

serv\_address.sin\_family = AF\_INET;

serv\_address.sin\_port = htons(PORTNO);

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

retValue = bind(sfd, (struct sockaddr \*)&serv\_address, sizeof(serv\_address));

if(retValue < 0)

{

perror("bind() ");

exit(EXIT\_FAILURE);

}

printf("\nBinded the server to the ip address with portno\n");

retValue = listen(sfd, 5);

if(retValue < 0)

{

perror("listen() ");

exit(EXIT\_FAILURE);

}

printf("\nListening to the clients now\n");

while(1)

{

csfd = accept(sfd, (struct sockaddr \*)&client\_address, &clientAddlen);

if(csfd < 0)

{

perror("accept() ");

continue;

}

// Receive structure data from the client

read(csfd, &receivedData, sizeof(struct Data));

// Display the received data

printf("\nReceived structure from client:\n");

printf("ID: %d\n", receivedData.id);

printf("Name: %s\n", receivedData.name);

// Close client socket if done

close(csfd);

}

close(sfd);

return 0;

}

Program for client:

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define MAXBUFF 1024

#define PORTNO 4003

// Define a structure to send

struct Data {

int id;

char name[50];

};

int main()

{

int sfd = 0;

struct sockaddr\_in serv\_address;

struct Data dataToSend;

// Create socket

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

if(sfd < 0)

{

perror("socket() ");

exit(EXIT\_FAILURE);

}

printf("\nSocket created with sockfd : %d\n", sfd);

// Set server address

memset(&serv\_address, '\0', sizeof(serv\_address));

serv\_address.sin\_family = AF\_INET;

serv\_address.sin\_port = htons(PORTNO);

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

// Connect to the server

if(connect(sfd, (struct sockaddr \*)&serv\_address, sizeof(serv\_address)) < 0)

{

perror("connect() ");

exit(EXIT\_FAILURE);

}

printf("\nConnected to the server\n");

// Populate the structure with data

dataToSend.id = 1;

strcpy(dataToSend.name, "John Doe");

// Send the structure to the server

write(sfd, &dataToSend, sizeof(struct Data));

printf("\nStructure sent to server:\n");

printf("ID: %d\n", dataToSend.id);

printf("Name: %s\n", dataToSend.name);

// Close the socket

close(sfd);

return 0;

}

Server should not be printing anything in the screen and it creates logs

* UDP:
* In this accept is not there so it cannot have the ip address of the client so in UDP first we have to sendto() from the client to the server(recvfrom())
* In sendto() include the libraries🡺 #include <sys/socket.h>
* In socket() 🡺 #include <sys/socket.h>
* In inet 🡺 #include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

* Write and sendto is same between flag is there in falg in sendto()
* In udp we use sendto() and recvfrom()
* Sendto() syntax:

ssize\_t sendto(int sockfd, const void \*buf, size\_t len, int flags,

const struct sockaddr \*dest\_addr, socklen\_t addrlen);

* Syntax for recvfrom():

ssize\_t recvfrom(int sockfd, void \*buf, size\_t len, int flags,

struct sockaddr \*src\_addr, socklen\_t \*addrlen);