**Day 25**

**Semaphore**

1. Semget() : create or access a semaphore set

Syntax:

* Int semget(key\_t key.int nsems, int semflg);

Parameters:

1. Semctl() : Control operation on a semaphore

Syntax:

Int semctl(int semid, int semnum, int cmd, union semnum arg);

Parameters:

semid

semnum

cmd

arg

1. Semop: perform semaphore operation(lock/unlock)

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

gcc -o prod sem\_

**OSI MODEL:**

* The **OSI (Open Systems Interconnection)** Model is a set of rules that explains how different computer systems communicate over a network.
* OSI Model was developed by the **International Organization for Standardization (ISO)**.
* The OSI Model consists of 7 layers and each layer has specific functions and responsibilities.

A diagram of a computer

Description automatically generated

* **Socket Programming:**
* Socket programming is a way of connecting two nodes on a network to communicate with each other.
* One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection.
* The server forms the listener socket while the client reaches out to the server.

A diagram of a computer program

Description automatically generated

**Stages for Server:**

1. **Socket Creation**

Syntax: int sockfd = socket (domain, type, protocol)

* **sockfd**: socket descriptor, an integer (like a file handle)
* **domain**: integer, specifies communication domain. We use AF\_ LOCAL as defined in the POSIX standard for communication between processes on the same host.
* **type:** communication type  
  SOCK\_STREAM: TCP(reliable, connection-oriented)  
  SOCK\_DGRAM: UDP(unreliable, connectionless)
* **protocol:**Protocol value for Internet Protocol(IP), which is 0. This is the same number that appears on the protocol field in the IP header of a packet.(man protocols for more details)

TCP-0

UDP-

socket - create an endpoint for communication

1. **Bind:**

int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

* After the creation of the socket, the bind function binds the socket to the address and port number specified in addr(custom data structure).

1. **Listen:**

int listen(int sockfd, int backlog);

* listen - listen for connections on a socket.
* The behavior of the backlog argument on TCP sockets changed with Linux 2.2. Now it specifies the queue length for completely established sockets waiting to be accepted, instead of the number of incomplete connection requests. The maximum length of the queue for incomplete sockets can be set using /proc/sys/net/ipv4/tcp\_max\_syn\_backlog.
* The backlog 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.

1. **Accept:**

* int new\_socket= accept(int sockfd, struct sockaddr \*addr, sckle**n\_t \*addlen):**

**Stacket connection:** Exactly the same as that of server’s socket creation

* **2. Connect:**The connect() system call connects the socket referred to by the file descriptor sockfd to the address specified by addr. Server’s address and port is specified in addr.
* int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

**memset:** setting all the values to NULL.

**Man htonos:**

convert values between host and network byte order.

The **htonl() function** converts the unsigned integer hostlong from host byte order to network byte order.

The **htons() function** converts the unsigned short integer hostshort from host byte order to network byte order.

The **ntohl() function** converts the unsigned integer netlong from network byte order to host byte order.

The ntohs**() function** converts the unsigned short integer netshort from network byte order to host byte order.

* If the server is closed you need to close client as well.

**Socket:**

socket - create an endpoint for communication.

SYNOPSIS

#include <sys/types.h> /\* See NOTES \*/

#include <sys/socket.h>

int socket(int domain, int type, int protocol);

DESCRIPTION

socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

The domain argument specifies a communication domain; this selects the protocol family which will be used for communication. These families are defined in <sys/socket.h>.

send:

NAME

send, sendto, sendmsg - send a message on a socket

SYNOPSIS

#include <sys/types.h>

#include <sys/socket.h>

ssize\_t send(int sockfd, const void \*buf, size\_t len, int flags);

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

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

ssize\_t sendmsg(int sockfd, const struct msghdr \*msg, int flags);

DESCRIPTION

The system calls send(), sendto(), and sendmsg() are used to trans‐

mit a message to another socket.

The send() call may be used only when the socket is in a connected

state (so that the intended recipient is known). The only differ‐

ence between send() and write(2) is the presence of flags. With a

zero flags argument, send() is equivalent to write(2). Also, the

following call

send(sockfd, buf, len, flags);

is equivalent to

sendto(sockfd, buf, len, flags, NULL, 0);

**The flags argument:**

The flags argument is the bitwise OR of zero or more of the following flags.

* Length is nothing but size of socket.