Computer Networks Assignment: 02

Submitted by:

Rishika Hazarika
Roll No: 210710007037
Date: 23.05.2024

1. **Briefly explain the basic idea of Spread Spectrum and its uses. Also mention the different types of spread spectrum techniques available.**

Spread spectrum is a technique used for wireless communications in telecommunication and radio communication where the frequency of the transmitted signal is dispersed. This results in a much greater bandwidth than the original one. This dispersion is usually accomplished by modulating the signal with a pseudo-random sequence, referred to as the spreading code. The primary objective is to make the transmitted signal resemble noise to unintended receivers, thereby increasing its resistance to interference, hamming and eavesdropping attacks, including Man-in-the-Middle attacks.

Uses of Spread Spectrum:

- **Global Positioning System (GPS):** GPS uses Direct Sequence Spread Spectrum (DSSS) to transmit signals that are resistant to interference and multipath effects. The spreading codes help distinguish signals from different satellites, thus providing reliable and accurate positioning data.
- **Wireless Communications:**
  - **Wi-Fi**: Wi-Fi standards such as IEEE 802.11b use DSSS to enhance data transmission reliability and resistance to interference in crowded frequency bands.
  - **Bluetooth**: Bluetooth technology uses Frequency Hopping Spread Spectrum (FHSS) to minimize interference with other devices operating in the same frequency band.
- **Cellular Networks**: Spread Spectrum is extensively used in CDMA (Code Division Multiple Access) to allow multiple users to share the same frequency band without significant interference, by assigning unique spreading codes to each user. This is used in 3G networks and some aspects of 4G.
- **Satellite Communications**: Spread spectrum helps maintain reliable communication links between satellites and ground stations, even in the presence of interference and noise.
- **Military Communications**: Spread spectrum is used to secure military communication links, making them resistant to eavesdropping and jamming. Its resilience to interference and jamming makes it ideal for military operations in hostile environments.
- **Cordless Phones**: Cordless phones use spread spectrum to reduce interference from other devices and to improve the clarity and security of voice communications.
- **RFID (Radio Frequency Identification**: RFID systems use spread spectrum to ensure secure and reliable data transmission for tracking and identification purposes.

The different types of spread spectrum techniques available are:

(a) **Direct Sequence Spread Spectrum (DSSS)**: DSSS spreads the signal across a wider bandwidth by modulating the data signal with a high-rate pseudo-random bit sequence (also known as a chipping code). DSSS is commonly used in Wi-Fi and CDMA cellular systems.

(b) **Frequency Hopping Spread Spectrum (FHSS)**: FHSS involves rapidly switching carrier frequencies according to a pseudo-random sequence known to both the transmitter and receiver. Each transmitter hops from one frequency to another in a pseudorandom pattern in intervals called 'hop periods'. FHSS is used in bluetooth technology and military communication systems.

(c) **Time Hopping Spread Spectrum (THSS)**: THSS involves dividing the transmission time into short, pseudo-random intervals and transmitting data in short bursts at these intervals. Each burst is transmitted at a different time slot, spreading the signal over time. THSS is less common but is used in some ultra-wideband (UWB) systems.

(d) **Chirp Spread Spectrum (CSS)**: This method uses frequency-modulated continuous waveforms, where the frequency of the carrier signal varies continously ver time. The spread signal is generated by modulating the data signal with a chirp waveform. CSS is resilient to multipath fading and is used in systems like LoRa (Long Range communication) for IoT applications

2. Briefly explain the working principle of CSMA/CA.

Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) is a protocol used in wireless networks to regulate access of devices to the shared transmission medium and avoid collisions.

The working principle of CSMA/CD are:

(a) **Carrier Sense (CS)**: Before transmitting data, a device using CSMA/CD listens to the channel to detect if it's currently in use by other nodes. If the channel is idle, the device can proceed with transmission. Otherwise, the node waits for a random amount of time before retrying.

(b) **Multiple Access**: Once the channel is sensed as idle, the device initiates transmission of its data. Multiple devices can access the channel, but they must contend for access if they detect the channel is idle simultaneously. There is a possibility of collisions if multiple devices attempt to transmit simultaneously.

(c) **Collision Avoidance**: CSMA/CA aims to avoid collisions altogether. To achieve this, before transmitting, the device sends out a short request to send (RTS) frame or acknowledgment (ACK) to the intended recipient. If the recipient is available and responds with a clear to send (CTS) frame, indicating it's ready to receive data, the transmitting device proceeds with the data transmission. If no response is received or if the channel is busy, the device retries after a random waiting period. The random waiting period is chosen to minimize the likelihood of multiple nodes choosing the same time slot for transmission.

3. Briefly explain the different features of SCTP and its uses.

Stream Control Transmission Protocol (SCTP) is a transport layer protocol that provides reliable, connection-oriented communication between two endpoints. Here's a brief explanation of its different features and uses:

(a) **Message-Oriented Communication**: SCTP supports both message-oriented and stream-oriented communication. It allows the transmission of discrete messages, which preserves message boundaries and enables message-level acknowledgment. Message-oriented communication makes SCTP suitable for applications that require precise message delineation.

(b) **Multi-Homing**: SCTP supports multi-homing, allowing endpoints to have multiple IP addresses. If one path fails, the protocol can seamlessly switch to another, enhancing reliability and fault tolerance.

(c) **Multi-Streaming**:SCTP enables the transmission of multiple independent streams of data within a single connection. Each stream operates independently, allowing for concurrent transmission and reception of data. This means that the loss of a packet in one stream does not block the delivery of packets in other streams, reducing the problem of head-of-line blocking.

(d) **Ordered and Unordered Delivery**: SCTP supports both ordered and unordered delivery of messages. Ordered delivery ensures that messages are delivered in the same order they were sent, while unordered delivery allows messages to be delivered independently of their transmission order. This flexibility can optimize performance depending on the application's requirement.

(e) **Congestion Control**: SCTP includes congestion control mechanisms to regulate the flow of data and prevent network congestion. It dynamically adjusts the transmission rate based on network conditions to optimize performance

(f) **Path Management**: SCTP continuously monitors the status of each network path using heartbeat messages. This helps in detecting path failures and switching to alternate paths as necessary.

(g) **Error Detection and Correction**: SCTP includes robust error detection and correction mechanisms, using checksums to ensure data integrity during transmission.

Uses of SCTP:

(a) **Telecommunications**: SCTP is used in telecommunications networks for signaling and control messages in IP-based networks, such as in the SIGTRAN protocol suite for transporting SS7 signaling messages over IP networks.

(b) **Voice over IP (VoIP)**: SCTP is employed in VoIP applications for its reliability, multi-streaming, and congestion control features, where different types of data (voice, signaling) can be sent over separate streams within the same association.

(c) **Web Services**: SCTP can be utilized in web services for reliable data transfer, especially in scenarios where ordered delivery, multi-homing, and fault tolerance are crucial.

(d) **Real-Time Communication**: SCTP is suitable for real-time communication applications, such as online gaming and video streaming, where low latency, resilience to network failures, multi-streaming and reduced head-of-line blocking are necessary.

4. Briefly explain the working idea behind the following attacks.

I. Port Scanning:

Port scanning is a technique used by attackers to discover which ports on a target system are open and listening for incoming connections and what services are running on those ports. The basic idea behind port scanning is to systematically check each port on a target system to determine its status (open, closed, or filtered).

The working principle of port scanning is mentioned below:

i. **Identifying Target Systems:** The attacker selects a target system or range of systems to scan for open ports. This could be a single host, a range of IP addresses, or even an entire network.

ii. **Choosing Port Ranges**: The attacker selects which ports to scan using specialized software tools called port scanners. This can range from scanning a specific port (e.g., port 80 for HTTP) to scanning a wide range of ports (e.g., all ports from 1 to 65535). Port scanners automate the process of scanning and can provide detailed information about the target system's network services.

iii. **Initiating Connection Attempts:** The attacker sends connection attempts to the target system on the chosen ports. Different types of port scans may use different methods to establish connections, such as:

- TCP Connect Scan: This method attempts to establish a full TCP connection with each port on the target system. If a connection attempt is successful and the target responds, the port is considered open. This indicates that a service is actively listening for incoming connections on that port.
- SYN Scan (Half-open Scan): The scanner sends SYN (synchronization) packets to the target ports. If the port responds with a SYN-ACK (synchronization acknowledgment) packet, it indicates that the port is open. However, the connection is not completed, leaving the port in a half-open state, which can be less detectable by intrusion detection systems.
- UDP Scan: This technique involves sending UDP (User Datagram Protocol) packets to various UDP ports on the target system. If the system

responds with an ICMP (Internet Control Message Protocol) port unreachable message, it indicates that the port is closed. However, if no response is received, it suggests that the port might be open or filtered.

- Stealth Scan (FIN, Xmas, Null): These scans attempt to determine open ports by sending specially crafted packets that do not adhere to the usual TCP protocol conventions. For example, in a FIN scan, the scanner sends packets with the FIN (finish) flag set, expecting different responses from open and closed ports.

iv. **Analyzing Results:** Once the port scanning process is complete, the attacker analyzes the results to identify potential vulnerabilities or services running on the target system. This information can be used for further attacks, such as exploiting known vulnerabilities or launching targeted attacks against specific services.

II. ARP spoofing or ARP cache poisoning:

ARP (Address Resolution Protocol) spoofing, also known as ARP cache poisoning, is a type of cyber attack where an attacker sends falsified ARP messages over a local area network (LAN), tricking devices into associating the attacker's MAC address with the IP address of a legitimate network entity, such as a router or another device. Here's how it works:

i. **ARP Protocol:** ARP is used to map IP addresses to MAC (Media Access Control) addresses on a local network. When one device needs to communicate with another device on the same network, it sends out an ARP request to discover the MAC address associated with the target IP address.

ii. **Spoofing ARP Replies:** The attacker initiaies the ARP spoofing attack by sending out ARP messages to the target network with falsified source IP and MAC addresses. These forged ARP messages typically contain the attacker's own MAC address mapped to the target's IP address.

iii. **ARP Cache Poisoning:** When devices on the network receive these falsified ARP messages, they update their ARP cache tables with the attacker's MAC address for the corresponding IP address. As a result, traffic intended for the legitimate network gateway or other devices is now sent to the attacker's MAC address.

iv. **Packet Interception:** With the attacker now positioned between the victim and other devices on the network, they can intercept, modify, or block the communication between the victim and other hosts. This interception can lead to various malicious activities, such as eavesdropping on sensitive data, altering the contents of data packets, or launching further attacks.

III. DNS spoofing or DNS cache poisoning:

DNS spoofing, also known as DNS cache poisoning, is a technique used by attackers to manipulate the DNS (Domain Name System) resolution process. The basic idea behind DNS spoofing is to corrupt the DNS cache of a DNS resolver with false information, redirecting users to malicious websites or intercepting their traffic. Here's how DNS spoofing typically works:

i. **DNS Resolution Process** When a user wants to access a website by its domain name (e.g., www.example.com), their device sends a DNS query to a DNS resolver (such as their ISP's DNS server or a public DNS server like Google DNS). The resolver then looks up the domain name in its cache or forwards the query to other DNS servers until it finds the corresponding IP address for the domain name.

ii. **Manipulating DNS responses:**An attacker intercepts DNS queries or sends forged DNS responses to the DNS resolver. The attacker's goal is to trick the resolver into caching false DNS records that map legitimate domain names to malicious IP addresses controlled by the attacker.

iii. **Cache Poisoning:**The attacker sends a DNS response with spoofed information, such as mapping a legitimate domain name to the attacker's IP address. If the DNS resolver accepts the forged response and caches the false DNS record, subsequent DNS queries for that domain name will be resolved to the attacker's IP address instead of the legitimate one.

iv. **Redirecting Traffic:**Once the DNS cache is poisoned, users who attempt to access the legitimate website are redirected to the attacker-controlled server. The attacker can then intercept, manipulate, or monitor the traffic passing through their server for malicious purposes, such as phishing, malware distribution, or data theft.

IV. DoS and DDos:

Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks are cyber attacks aimed at disrupting the normal functioning and availability of a target system or network by overwhelming it with a large volume of traffic.

The basic working principle behind such attacks are mentioned below:

i. DoS Attack:

- In a DoS attack, an attacker uses a single source, typically a compromised computer or a botnet (a network of compromised computers), to flood a target system or network with a large volume of traffic or requests.
- The overwhelming volume of traffic exhausts the target system's resources, such as bandwidth, CPU, memory, or network connections, causing it to become slow, unresponsive, or completely unavailable to legitimate users.
- Examples of DoS attack methods include:
  - ICMP Flood: Sending a large number of ICMP (Internet Control Message Protocol) packets to overwhelm network bandwidth.
  - SYN Flood: Sending a large number of TCP SYN (synchronization) packets to consume server resources and prevent legitimate connections from being established.
  - UDP Flood: Sending a large number of UDP (User Datagram Protocol) packets to flood a target server with unwanted traffic.

ii. DDos Attack:

- A DDoS attack amplifies the impact of a DoS attack by involving multiple sources, often thousands or even millions of compromised devices distributed across the internet.
- The attacker controls a botnet, which consists of compromised computers, servers, IoT (Internet of Things) devices, or other internet-connected devices infected with malware.
- The attacker orchestrates these devices to simultaneously launch coordinated attacks against the target system or network, overwhelming its resources with a massive volume of traffic.
- DDoS attacks are typically more difficult to mitigate than DoS attacks due to the distributed nature of the attack sources and the sheer scale of the attack traffic.
- DDoS attacks can be further categorized based on the techniques used, such as volumetric attacks, which flood the target with high volumes of traffic, or application layer attacks, which exploit vulnerabilities in the application layer of the target system.

V. ICMP flood attack or ping flood attack:

An ICMP flood, also known as a ping flood attack, is a type of denail-of-service (DoS) attack that works by overwhelming a target system with a high volume of ICMP (Internet Control Message Protocol) echo request packets, commonly known as "pings."

The working principle behind such attacks is mentioned below:

i. **Sending ICMP Echo requests:** The attacker sends a large number of ICMP echo request packets to the target's IP address, to check if it is reachable and responsive.

ii. **Target's response:** Upon receiving these packets, the target tries to respond to each one with an ICMP echo reply.

iii. **Target Overwhemled:** However, due to the sheer volume of of ICMP echo request packets, the target's system becomes overwhelmed trying to process and respond to them all.

iv. **Resource Exhaustion:** As a result, the target's system can run out of resources such as bandwidth, CPU, memory or network connections, causing it to become slow, unresponsive, or completely unavailable to the legitimate users.

VI. Ping of death:

The Ping of death attack is a type of denial-of-service (DoS) attack that exploits vulnerabilities in the way some operating systems handle large, oversized or malformed ICMP packets.

The basic working principle behind such attacks are mentioned below:

i. **Oversized or Malformed Packets:** The Ping of death attack involes sending ICMP echo request packets that are intentionally crafted to be larger than

the maximum size allowed by the Internet Protocol (IP) standards. This could involve manipulating the packet headers or including extra data to exceed the size limit.

ii. **Packet Fragmentation:** When a packet exceeds the maximum size allowed for transmission, it needs to be fragmented into smaller packets by routers along the network path. However, some operating systems and network devices may have vulnerabilities in their handling of fragmented packets.

iii. **Attack Initiation:** The attacker sends oversized ICMP echo request packets to the target system. These packets are designed to trigger buffer overflow or other vulnerabilities in the target system's network stack when it attempts to reassemble the fragmented packets.

iv. **Buffer Overflow**: If the target system's network stack is vulnerable to the oversized ICMP packets, it may crash, become unstable, or enter into an unresponsive state due to buffer overflow or other memory corruption issues caused by the malformed packets.

VII. Smurf Attack:

Smurf attack is a type of distributed denial-of-service (DDoS) attack that relies on spoofing and amplification to overwhelm a target network or server with a flood of ICMP (Internet Control Message Protocol) packets and IP broadcast addresses.

The basic working principle behind such attacks is mentioned below:

i. **Spoofing:** The attacker spoofs the source IP address of the ICMP packets to appear as if they originated from the target's IP address.

ii. **Amplification:** The attacker sends a large number of ICMP echo request packets (ping requests) to a broadcast address on a network that supports IP broadcast. The broadcast address ensures that all devices on the network receive the packet.Each device on the network that receives the spoofed ICMP packet will generate an ICMP echo reply (ping response) and send it back to the source IP address specified in the ICMP packet.

iii. **Network Amplification:** Since the attacker sends a large number of ICP echo request packets with the target's IP address as the source address, the ictim is inundated with ICMP echo reply packets from all devices within the broadcast domain, resulting in a significant amplication of network traffic directed towards the target

iv. **Denial of Service:** The volume of ICMP echo reply packets overwhelms the victim's network bandwidth, CPU, and other resources, causing it to become slow, unresponsive, or completely unavailable to legitimate users. This denial-of-service condition disrupts the target's ability to communicate over the network and access network services.

VIII. SYN flood attack: A SYN flood attack is a type of denial-of-service (DoS) attack that exploits the TCP three-way handshake process to overwhelm a target server with a flood of connection requests.

The SYN flood attack works in the following way:

i. **TCP Three-Way Handshake:** When a client wants to establish a TCP connection with a server, it initiates a three-way handshake:
   - SYN (Synchtonize): The client sends a SYN packet to the server to initiate a connection request.
   - SYN-ACK (Synchronize-Acknowledgement): if the server is available and willing to establish a connection, it responds with a SYN-ACK packet, indicating acknowledgment of the client's request to establish a connection.
   - ACK (Acknowledgement): The client then sends an ACK packet back to the server, acknowledging the receipt of the SYN-ACK packet. At this point, the connection is established, and data transfer can begin.

ii. **Exploiting the Handshake:** In a SYN flood attack, the attacker sends a large number of SYN packets to the target server, pretending to initiate TCP connections but does not respond to the SYN-ACK packets from the server. This causes the server to keep the half-open connections in a pending state, waiting for the final ACK packet from the client to complete the handshake and establish the connection.

iii. **Resource Exhaustion:** Since the attacker does not respond to the SYN-ACK packets, the server keeps allocating resources (such as memory and CPU time) to maintain the half-open connections. As a result, the server's resources become exhausted, and it is unable to handle legitimate connection requests from other clients.

iv. **Denial of Service:** The SYN flood attack overwhelms the target server's resources, causing it to become slow, unresponsive, or completely unavailable to legitimate users. This denial-of-service condition disrupts the server's ability to handle legitimate traffic and provide services to its intended users.

IX. Man-in-the-middle attack:

A Man-in-the-Middle (MITM) attack is a form of cyberattack where the attacker intercepts and potentially alters communication between two parties without their knowledge or consent.

The basic working idea behind such attacks is mentioned below:

i. **Interception:** The attacker positions themselves between the two parties (such as a client and a server) who are communicating with each other. This could be achieved by various means, often through exploiting vulnerabilities in network protocols, compromising network devices or using techniques like ARP spoofing.

ii. **Spoofing:** The attacker typically impersonates both parties to the other, making each party believe they are communicating directly with the intended recipient. This can involve spoofing IP addresses, MAC addresses, or even SSL certificates to appear legitimate.

iii. **Monitoring:** With the communication passing through the attacker's system, they can intercept and monitor the data being transmitted between the two parties. This includes capturing sensitive information like login credentials, financial data, or personal information.

iv. **Packet Manipulation:** In some cases, the attacker may alter the intercepted data, insert malicious code or modify the communication flow to suit their objectives.

v. **Relay:** The attacker may also act as a relay between the two parties, forwarding legitimate messages while still intercepting and potentially modifying the content.

X. Session hijacking attack:

Session hijacking is a type of cyber attack where an attacker takes control of a valid, active session between a client and a server, allowing them to impersonate the user and perform unauthorized actions. Here's how it typically works:

i. **Session Establishment:** When a user logs into a web application, a session is established between the user's client device (such as a web browser) and the server hosting the application. The server assigns a session ID (e.g., a session cookie) to the user's session, which is used to authenticate and authorize subsequent requests from the user.

ii. **Interception:** The attacker intercepts the session ID while it is being transmitted communication between the client and the server. This can be done through various means, such as eavesdropping on network traffic, packet sniffing on unsecured networks, compromising the user's device with malware, or exploiting vulnerabilities in the client or server software.

iii. **Session Theft:** Once the attacker has intercepted the session ID, they can use it to gain access to the user's active session and impersonate the user's session. This allows them to perform actions on the user's behalf, such as accessing sensitive information, modifying account settings, or making unauthorized transactions.

iv. **Detection Avoidance:** To avoid detection, the attacker may try to maintain the appearance of a legitimate session by mimicking the user's behavior and avoiding actions that would raise suspicion, such as logging out or changing sensitive account settings.

5. Write a simple Chatting Program in C, using sockets.

client.c

```
#include <stdio.h>
#include <unistd.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<netinet/in.h>
```

```
#include<sys/socket.h>
#include<string.h>
int main(){
  int my_socket;
  socklen_t sock_addr_len;
  struct sockaddr_in sock_addr;
  char ch[100];
  my_socket = socket(AF_INET, SOCK_STREAM, 0);
  sock_addr.sin_family = AF_INET;
  sock_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
  sock_addr.sin_port = htons(44332);
  sock_addr_len = sizeof(sock_addr);
  if (connect(my_socket, (struct sockaddr *)&sock_addr, sock_addr_len)==-1)
   {
     perror("Client failure");
     exit(EXIT_FAILURE);
   }
  scanf(" %99[^\n]", ch);
  write(my_socket, ch, 100);
  while(1)
   {
     //listen(my_socket, 1);
     read(my_socket, &ch, 100);
     printf("server: %s\n", ch);
     scanf(" %99[^\n]", ch);
     write(my_socket, ch, 100);
   }
   close(my_socket);
   return 0;
}
```

server.c

```
#include<stdio.h>
#include<arpa/inet.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<unistd.h>
#include<stdlib.h>
#include<string.h>
int main(){
 int serv_sock, client_sock;
 socklen_t serv_addr_len, client_addr_len;
 struct sockaddr_in server_address;
 struct sockaddr_in client_address;
```

```
 serv_sock = socket(AF_INET, SOCK_STREAM, 0);
 server_address.sin_family = AF_INET;
 server_address.sin_addr.s_addr = inet_addr("127.0.0.1");
 server_address.sin_port = htons(44332);
 serv_addr_len = sizeof(server_address);
 if(bind(serv_sock, (struct sockaddr *)&server_address, serv_addr_len)==-1)
  {
    perror("Bind error");
    exit(EXIT_FAILURE);
  }
 listen(serv_sock, 1);
 char ch[100];
 printf("Server waiting...\n");
 client_addr_len = sizeof(client_address);
 client_sock = accept(serv_sock, (struct sockaddr *)&client_address,
 &client_addr_len);
 read(client_sock, ch, 100);
 while(1)
  {
    printf("client: %s\nreply: ", ch);
    scanf(" %99[^\n]", ch);
    write(client_sock, ch, 100);
    //listen(serv_sock, 1);
    read(client_sock, ch, 100);
  }
  close(client_sock);
  close(serv_sock);
  return 0;
}
```

6. **Write the necessary socket programs in C to demonstrate the use of Multi-threaded client and server.**

   client.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>
void* clienthread(void* args){
 int client_request = *((int*)args);
```

```
 int network_socket;
 network_socket = socket(AF_INET,SOCK_STREAM, 0);
 struct sockaddr_in server_address;
 server_address.sin_family = AF_INET;
 server_address.sin_addr.s_addr = INADDR_ANY;
 server_address.sin_port = htons(8989);
 int connection_status = connect(network_socket, (struct sockaddr*)&
 server_address, sizeof(server_address));
 if (connection_status < 0)
  {
    puts("Error\n");
    return 0;
  }
 printf("Connection established\n");
 send(network_socket, &client_request, sizeof(client_request), 0);
 close(network_socket);
 pthread_exit(NULL);
 return 0;
}

int main(){
 printf("1. Read\n");
 printf("2. Write\n");
 int choice;
 scanf("%d", &choice);
 pthread_t tid;
 switch (choice) {
  case 1: {
            int client_request = 1;
            pthread_create(&tid, NULL,clienthread, &client_request);
            sleep(20);
            break;
          }
  case 2: {
            int client_request = 2;
            pthread_create(&tid, NULL, clienthread, &client_request);
            sleep(20);
            break;
          }
  default:
            printf("Invalid Input\n");
            break;
  }
 pthread_join(tid, NULL);
 }
```

server.c

```
#include <arpa/inet.h>
#include <pthread.h>
#include <semaphore.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
sem_t x, y;
pthread_t tid;
pthread_t writerthreads[100];
pthread_t readerthreads[100];
int readercount = 0;
void* reader(void* param){
 sem_wait(&x);
 readercount++;
 if (readercount == 1)
   sem_wait(&y);
 sem_post(&x);
 printf("\n%d reader is inside", readercount);
 sleep(5);
 sem_wait(&x);
 readercount--;
 if (readercount == 0) {
   sem_post(&y);
 }
 sem_post(&x);
 printf("\n%d Reader is leaving",readercount + 1);
 pthread_exit(NULL);
}

void* writer(void* param){
 printf("\nWriter is trying to enter");
 sem_wait(&y);
 printf("\nWriter has entered");
 sem_post(&y);
 printf("\nWriter is leaving");
 pthread_exit(NULL);
}

int main(){
 int serverSocket, newSocket;
 struct sockaddr_in serverAddr;
```

```
 struct sockaddr_storage serverStorage;
 socklen_t addr_size;
 sem_init(&x, 0, 1);
 sem_init(&y, 0, 1);
 serverSocket = socket(AF_INET, SOCK_STREAM, 0);
 serverAddr.sin_addr.s_addr = INADDR_ANY;
 serverAddr.sin_family = AF_INET;
 serverAddr.sin_port = htons(8989);
 bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr));
 if (listen(serverSocket, 50) == 0)
   printf("Listening\n");
 else
   printf("Error\n");
 pthread_t tid[60];
 int i = 0;
 while (1) {
   addr_size = sizeof(serverStorage);
   newSocket = accept(serverSocket,(struct sockaddr*)&serverStorage,&addr_size);
   int choice = 0;
   recv(newSocket,&choice, sizeof(choice), 0);
   if (choice == 1) {
     if (pthread_create(&readerthreads[i++], NULL, reader, &newSocket)!= 0)
       printf("Failed to create thread\n");
     }
     else if (choice == 2) {
         if (pthread_create(&writerthreads[i++], NULL, writer, &newSocket)!= 0)
           printf("Failed to create thread\n");
     }
   if (i >= 50) {
     i = 0;
     while (i < 50) {
         pthread_join(writerthreads[i++],NULL);
         pthread_join(readerthreads[i++],NULL);
     }
     i = 0;
   }
 }
 return 0;
}
```

: