Netzwerkkommunikation in Linux

Von Mohammed Al Assaf, Tony Gutzeit und Ansh Kakkar

Inhalt

- Programmierung des Menus
- Programmierung von TCP/UDP
  - Client
  - Server
  - PCAP

Menu

```
// Auswahl 1 des Menüs
void optionOne(){
    printf("Creating a TCP-Connection...\n----------\n");

    // Starten des Gegenprogramms auf einem zweiten Terminal
    system("gnome-terminal -- ./TCP/TCPServer");
    // Ausführung des Hauptprogramms auf dem gleichen Terminal
    system("./TCP/TCPClient");
}
```

Menu

```
    /* If-Abfrage und Eingabe miteinander, um sicherzustellen, dass eine Zahl eingegeben wurde,
     * gleichzeitig speichert "option" die Eingabe des Nutzers
     * und falls es keine Zahl war, springt die Schleife zurück and den Anfang */
    if (scanf("%d", &option) != 1) {
        printf("Invalid input. Please enter a number.\n");
        // Leeren des Input-Puffers, um für neue Eingaben bereit zu sein
        while (getchar() != '\n');
        continue;
    }

    // Switch-Case Funktion um die getätigte Auswahl schließlich durchzuführen
    switch (option) {
        case 0:
            // Standard Exit-Methode mit Standard Exit-Code 0
            exit(EXIT_SUCCESS);
        case 1:
            optionOne();
            break;
        case 2:
            optionTwo();
            break;
        case 3:
            optionThree();
            break;
        case 4:
            optionFour();
            break;
        default:
            printf("Invalid option. Please select a valid option (1-4).\n");
            break;
    }
}
```

Menu

```c
/* If-Abfrage und Eingabe miteinander, um sicherzustellen, dass eine Zahl eingegeben wurde,
 * gleichzeitig speichert "option" die Eingabe des Nutzers
 * und falls es keine Zahl war, springt die Schleife zurück and den Anfang */
if (scanf("%d", &option) != 1) {
    printf("Invalid input. Please enter a number.\n");
    // Leeren des Input-Puffers, um für neue Eingaben bereit zu sein
    while (getchar() != '\n');
    continue;
}

// Switch-Case Funktion um die getätigte Auswahl schließlich durchzuführen
switch (option) {
    case 0:
        // Standard Exit-Methode mit Standard Exit-Code 0
        exit(EXIT_SUCCESS);
```

Client

```c
// Unendliche Schleife, damit das Menü nur per Exit-Option verlassen werden kann
while (1) {
    // Prints für die verschiedenen verfügbaren Auswahlmöglichkeiten
    printf("Select an option:\n");
    printf("[1] Start the TCP-Client\n");
    printf("[2] Start PCAP-LOG\n");
    printf("[3] Get Service-Information\n");
    printf("[4] Get the Host-Name\n");
    printf("[0] To EXIT\n");

    /* If-Abfrage und Eingabe miteinander, um sicherzustellen, dass eine Zahl eingegeben wurde,
     * gleichzeitig speichert "option" die Eingabe des Nutzers
     * und falls es keine Zahl war, springt die Schleife zurück and den Anfang */
    if (scanf("%d", &option) != 1) {
        printf("Invalid input. Please enter a number.\n");
        // Leeren des Input-Puffers, um für neue Eingaben bereit zu sein
        while (getchar() != '\n');
        continue;
    }
```

TCP-Client

```c
// Unendliche Schleife, damit das Menü nur per Exit-Option verlassen werden kann
while (1) {
    // Prints für die verschiedenen verfügbaren Auswahlmöglichkeiten
    printf("Select an option:\n");
    printf("[1] Start the UDP-Client\n");
    printf("[2] Start PCAP-LOG\n");
    printf("[3] Get Service-Information\n");
    printf("[4] Get the Host-Name\n");
    printf("[0] To EXIT\n");

    /* If-Abfrage und Eingabe miteinander, um sicherzustellen, dass eine Zahl eingegeben wurde,
     * gleichzeitig speichert "option" die Eingabe des Nutzers
     * und falls es keine Zahl war, springt die Schleife zurück and den Anfang */
    if (scanf("%d", &option) != 1) {
        printf("Invalid input. Please enter a number.\n");
        // Leeren des Input-Puffers, um für neue Eingaben bereit zu sein
        while (getchar() != '\n');
        continue;
    }
```

UDP-Client

Client

```c
// Erstellen eines Sockets und Erstellen einer Verbindung zum Server
void creatingConnection(){
    // Der Socket-Datei-Verarbeiter (Hauptspeicher Variable des Sockets)
    int socketFileHandle;
    // Eine Structure für die Server Adresse
    struct sockaddr_in serverAddress;
    //Leeren der Server Adresse, da selbst eine IP festgelegt wird
    bzero(&serverAddress, sizeof(serverAddress));

    // Erstellen des Sockets, in diesem Fall TCP & Erfolg-/Fehler-Meldung
    socketFileHandle = socket(AF_INET, SOCK_STREAM, 0);
    if (socketFileHandle == -1) {
        printf("socket creation failed...\n");
        exit(0);
    }
    else {
        printf("Socket successfully created..\n");
    }
```

TCP-Client

```c
// Erstellen des Sockets, in diesem Fall UDP & Erfolg-/Fehler-Meldung
socketFileHandle = socket(AF_INET, SOCK_DGRAM, 0);
if (socketFileHandle == -1) {
    printf("socket creation failed...\n");
    exit(0);
}
else
    printf("Socket successfully created..\n");

// Verbindung mit dem Server durch connect, dem socketFileHandle, der Server Adresse und Socket Adresse & Erfolg-/Fehler-Meldung
if(connect(socketFileHandle, (struct sockaddr *)&serverAddress, sizeof(serverAddress)) < 0){
    printf("Error: Connect Failed \n");
    exit(0);
}else
    printf("Connection with Server successful\n");
```

UDP-Client

Client

```
// Methode zum Schreiben an den Server und Speichern im Puffer
write(socketFileHandle, buff, sizeof(buff));
/* Abfrage ob der Puffer nur aus dem Wort "exit" besteht,
 * mit der Methode "strncmp" welche 2 Strings vergleicht,
 * falls ja, dann wird die Schleife unterbrochen und die chatFunction Methode endet */
if ((strncmp(buff, "exit", 4)) == 0) {
    printf("Client Exit...\n");
    break;
}
// Methode zum Entleeren des Puffers
bzero(buff, MAX);

// Methode zum Lesen vom Server und Speichern im Puffer
read(socketFileHandle, buff, sizeof(buff));
// Ausgabe der Nachricht (des Puffers)
printf("From Server : %s\t", buff);
// Abfrage ob exit das Wort im Puffer ist ... (s.o.)
if ((strncmp(buff, "exit", 4)) == 0) {
    printf("Client Exit...\n");
    break;
}
```

TCP-Client

```
// Senden der Nachricht an den Server
sendto(socketFileHandle, buff, MAX, 0, (struct sockaddr *) NULL, sizeof(serverAddress));
/* Abfrage ob der Puffer nur aus dem Wort "exit" besteht,
 * mit der Methode "strncmp" welche 2 Strings vergleicht,
 * falls ja, dann wird die Schleife unterbrochen und die chatFunction Methode endet */
if ((strncmp(buff, "exit", 4)) == 0) {
    printf("Client Exit...\n");
    break;
}

// Leeren des Puffers
bzero(buff, MAX);

// Empfangen der Nachricht des Servers
recvfrom(socketFileHandle, buff, sizeof(buff), 0, (struct sockaddr *) NULL, NULL);
printf("From Server: %s\n", buff);
```

UDP-Client

Server

```
// Festlegung der maximalen Zeichen-Anzahl pro Nachricht
#define MAX 80
// Festlegung des Ports
#define PORT 8080
// Festlegung des Kürzels SA (Socket Adress) für die Structure sockaddr [Nur für Convenience]
#define SA struct sockaddr
```

TCP-Client

```
// Festlegung des Ports
#define PORT 5000
// Festlegung der maximalen Zeichen-Anzahl pro Nachricht
#define MAX 80
```

UDP-Client

Server

```c
// Endlos-Schleife, die nur bei Bedingungen abbricht, damit Chat endlos weitergehen kann
for (;;) {
    // Pro Durchang wird tmp auf 0 gesetzt
    tmp = 0;
    // Methode zum Entleeren des Puffers (für fehlerfreie, dopplungsfreie Übertragung wichtig)
    bzero(buff, MAX);

    // Methode zum Lesen vom Client und Speichern im Puffer
    read(connectionFileHandle, buff, sizeof(buff));

    // Ausgabe der Nachricht vom Client (Puffer print)
    printf("From client: %s\t To client : ", buff);
    // Methode zum Entleeren des Puffers
    bzero(buff, MAX);
    // Eingabe-Schleife, jedes Zeichen wird in den Puffer geschrieben bis 'Enter/Eingabetaste' gedrückt wird
    while ((buff[tmp++] = getchar()) != '\n');

    // Methode zum Schreiben an den Client und Speichern im Puffer
    write(connectionFileHandle, buff, sizeof(buff));

    /* Abfrage ob der Puffer nur aus dem Wort "exit" besteht,
     * mit der Methode "strncmp" welche 2 Strings vergleicht,
     * falls ja, dann wird die Schleife unterbrochen und die chatFunction Methode endet */
    if (strncmp(buff, "exit", 4) == 0) {
        printf("Server Exit...\n");
        break;
    }
```

TCP-Client

```c
// Endlos-Schleife, die nur bei Bedingungen abbricht, damit Chat endlos weitergehen kann
for(;;) {
    // Speichern der Länge der Client-Adresse
    length = sizeof(clientAddress);

    // Leeren des Puffers
    bzero(buff, MAX);

    // Empfangen der Nachricht vom Client, mit allen nötigen Informationen
    recvfrom(socketFileHandle, buff, sizeof(buff), 0, (struct sockaddr *) &clientAddress, &length);
    /* Abfrage ob der Puffer nur aus dem Wort "exit" besteht,
     * mit der Methode "strncmp" welche 2 Strings vergleicht,
     * falls ja, dann wird die Schleife unterbrochen und die chatFunction Methode endet */
    if ((strncmp(buff, "exit", 4)) == 0) {
        printf("Server Exit...\n");
        break;
    }

    // Ausgabe des Puffers
    printf("From Client: %s", buff);

    // Informations-Nachricht
    printf("Sending copy of Message as response...\n");
    // Senden der Nachricht an den Client mit einem unveränderten Puffer
    sendto(socketFileHandle, buff, MAX, 0, (struct sockaddr *) &clientAddress, sizeof(clientAddress));
}
```

UDP-Client

PCAP

```c
// Main Methode zum Erstellen eines PCAP-Geräts, um Netzwerk-Kommunikation mitzuschneiden
int main() {
    // Benötigte Variable zum Speichern und Weitergeben aller Informationen (Das PCAP-Gerät)
    pcap_t* handle;
    // Benötigter Error-Puffer im Falle eines Fehlers
    char errbuf[PCAP_ERRBUF_SIZE];

    // Öffnen des Netzwerk-Interfaces mit Erfolg- und Fehler-Meldung
    handle = pcap_open_live("lo", BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        printf("Error opening device: %s\n", errbuf);
        return 1;
    }else
        printf("Success opening device...\n");
```

TCP-Client

```c
// Main Methode zum Erstellen eines PCAP-Geräts, um Netzwerk-Kommunikation mitzuschneiden
int main() {
    // Benötigte Variable zum Speichern und Weitergeben aller Informationen (Das PCAP-Gerät)
    pcap_t* handle;
    // Benötigter Error-Puffer im Falle eines Fehlers
    char errbuf[PCAP_ERRBUF_SIZE];

    // Öffnen des Netzwerk-Interfaces mit Erfolg- und Fehler-Meldung
    handle = pcap_open_live("lo", BUFSIZ, 1, 1000, errbuf);
    if (handle == NULL) {
        printf("Error opening device: %s\n", errbuf);
        return 1;
    }else
        printf("Success opening device...\n");
```

UDP-Client

PCAP

```
// Finales Einstellen des Filters mit Erfolg- und Fehler-Meldung
if (pcap_setfilter(handle, &fp) == -1) {
    printf("Error setting filter: %s\n", pcap_geterr(handle));
    return 1;
}else
    printf("Success setting filter...\n");
// Finale Aktivierungsnachricht
printf("TCP-Packet capturing is active...\n");

// Methode zum eigentlichen Einfangen der Pakete
pcap_loop(handle, 0, packetHandler, NULL);

// Schließen des PCAP Geräts
pcap_close(handle);
```

TCP-Client

```
// Finales Einstellen des Filters mit Erfolg- und Fehler-Meldung
if (pcap_setfilter(handle, &fp) == -1) {
    printf("Error setting filter: %s\n", pcap_geterr(handle));
    return 1;
}else
    printf("Success setting filter...\n");
// Finale Aktivierungsnachricht
printf("UDP-Packet capturing is active...\n");

// Methode zum eigentlichen Einfangen der Pakete
pcap_loop(handle, 0, packetHandler, NULL);

// Schließen des PCAP Geräts
pcap_close(handle);
```

UDP-Client

PCAP

```
// Methode zum Extrahieren der Informationen aus IP und TCP Header
void parseTcpHeader(const unsigned char* packet) {
    // Verschiebung des Fokus vom "Ethernet Header" auf den "IP Header"
    const struct ip* ipHeader = (struct ip*)(packet + sizeof(struct ether_header));

    // Extrahierung der beiden IP-Adressen für Quelle und Ziel aus dem IP Header
    unsigned char* sourceIP = inet_ntoa(ipHeader->ip_src);
    unsigned char* destinationIP = inet_ntoa(ipHeader->ip_dst);

    // Verschiebung des Fokus vom "IP Header" auf den "TCP Header"
    const struct tcphdr* tcpHeader = (struct tcphdr*)(packet + sizeof(struct ether_header) + ipHeader->ip_hl * 4);
```

```
// Extrahierung der Ports, Sequenznummer, Bestätigungsnummer, Header-Länge und Flags
unsigned int sourcePort = ntohs(tcpHeader->th_sport);
unsigned int destinationPort = ntohs(tcpHeader->th_dport);
unsigned int sequenceNumber = ntohl(tcpHeader->th_seq);
unsigned int acknowledgmentNumber = ntohl(tcpHeader->th_ack);
unsigned int headerLength = tcpHeader->th_off * 4;
unsigned int flags = tcpHeader->th_flags;
```