NETZWERKKOMMUNIKATION IN LINUX

Betriebssysteme und Rechnernetze

Tony Gutzeit, Ansh Kakkar, Mohammed Al Assaf

Inhaltsverzeichnis

Abstract

Diese Dokumentation befasst sich mit der Netzwerkkommunikation in Linux. In insgesamt sieben verschieden Programmen wurden sowohl ein TCP-Chat als auch ein UDP-Chat und die Netzwerkmitschnitte programmiert. Dabei sind drei Programme für die TCP-Seite und drei Programme für die UDP-Seite, sowie ein Programm für das Start-Menü.

Einleitung

Zunächst wird sich mit den Startschwierigkeiten befasst, darauffolgend sind die Erklärungen für jedes einzelne Programm gegeben. Die Struktur des gesamten Programms sieht vor, dass beide Protokolle eine Server-Seite und eine Client-Seite besitzen. Dazu kommt noch der jeweilige PCAP-Abschnitt, um die Netzpakete einzufangen. Gesteuert wird alles vom Haupt-Menü.

Schwierigkeiten

Das Projekt bestand zunächst aus einer Reihe von Schwierigkeiten. Alle Gruppenmitglieder sind zwar mit Java vertraut, jedoch brachte C viele neues und vor allem große Umgewöhnungen mit sich. Demnach musste zuerst ein großer Teil der Recherche-Arbeit auf C fokussiert werden.

Außerdem waren die Protokolle TCP und UDP zum Startzeitpunkt des Projektes noch Neuland für alle Teammitglieder, genauso auch die richtige Funktion von Sockets oder die Möglichkeiten die PCAP mit sich bringt.

Nach mehreren Wochen Recherche und Test-Programmen konnte jedoch die eigentliche Aufgabe angegangen werden. Dabei fiel auf, dass die Methoden „getservbyname“ und „gethostbyname“ in der Aufgabenstellung schon etwas unklar definiert wurden. Darüber hinaus ließ sich erkennen, dass diese beiden Methoden

auch keinen wirklichen Sinn darstellen. Mittlerweile kann man zwar einen Service-Namen eingeben und dazu einen Port und das Protokoll erhalten, jedoch muss schon im Methodenaufruf selbst das Protokoll angegeben werden. Das heißt eigentlich findet „getservbyname“ nur den Port heraus, da die Aufgabe aber nicht spezifiziert hatte, ob dies genug sei, herrschte eine Unsicherheit unter den Teammitgliedern. Gleiches Spiel war es mit „gethostbyname“.

Die meisten Probleme machte jedoch der letzte Teil der Aufgabe, die PCAP-Implementierung. 70% der gefundenen Recherche-Ergebnisse bezogen sich auf Methoden, die seit mehreren Jahren nicht geupdatet wurden oder mittlerweile nicht mal mehr existieren. So dauerte es viel zu lange, um das richtige „device“ zum Abhören zu finden. Wenn einmal bekannt ist, dass das Gerät nicht der Standard „eth0“, sondern „lo“ heißen muss, lösen sich die meisten Probleme von selbst, aber allein das Anzeigen aller verfügbaren Geräte hat mehrere Tage wertvoller Arbeitszeit gekostet.

Letztendlich funktioniert nun doch alles einwandfrei, wobei natürlich noch jede Menge Verbesserungen im Command Line Interface möglich wären, um Abläufe noch schneller zu gestalten und Menü-Positionen eindeutig erkennbarer zu machen. Ein erster Ansatz dazu befindet sich bereits im Menu, nämlich das Wort „M E N U“ generiert aus „*“-Symbolen.

Programmierung von Menu / TCP / UDP

Hier starten nun die Erklärungen zu den einzelnen Programmen.

Menu

In dem Main-Menu interagiert der Nutzer des Programms mit diesem. Dieser Code in der Main ist ein C-Programm, das ein Menü mit verschiedenen Optionen bereitstellt. Die ersten drei #include-Anweisungen sind Header-Dateien, die Funktionen für die Ein- und Ausgabe, die Speicherzuweisung und die Prozesssteuerung bieten.

Die Methode „optionOne“ wird definiert. Diese Methode gibt "Creating a TCP-Connection..." aus und führt dann zwei Systembefehle aus. Der erste Befehl öffnet ein neues Terminal-Fenster und führt den Befehl „./TCP/TCPServer“ aus. Der zweite Befehl führt den Befehl „./TCP/TCPClient“ im aktuellen Terminal aus.

Die Methode „optionTwo“ ist ähnlich wie „optionOne“, jedoch für UDP-Verbindungen. Sie gibt "Creating an UDP-Connection..." aus und führt im Anschluss den UDP-Server und UDP-Client aus.

Die Methode „optionThree“ und „optionFour“ öffnen LOG-Dateien für den Datenverkehr von TCP bzw. UDP. Sie geben "Opening LOG-FILES..." aus, warten zwei Sekunden und rufen dann den Befehl „sudo less <filename>“ auf, um die Log-Datei anzuzeigen. Der Benutzer kann die Anzeige beenden, indem er „Q“ drückt.

Die Main-Methode ist der Einstiegspunkt des Programms. Sie enthält eine Endlosschleife, die das Menü anzeigt und die Benutzereingabe verarbeitet. Die Schleife wird nur durchbrochen, wenn der Benutzer die Option „0“ auswählt, um das Programm zu beenden. Innerhalb der Schleife werden die verschiedenen Optionen mittels des Befehls „printf“ angezeigt, und der Benutzer wird dazu aufgefordert, eine Option auszuwählen. Der Ausdruck „scanf("%d", &option)“ liest die Benutzereingabe als Ganzzahl und speichert sie in der Variablen option. Wenn die Eingabe keine Ganzzahl ist, gibt das Programm eine Fehlermeldung aus und die

Eingabeaufforderung wird erneut angezeigt. Die „while-Schleife“ verwendet „getchar()“ zur Bereinigung des Eingabepuffers, um sicherzustellen, dass der nächste scanf-Aufruf eine gültige Eingabe erhält.

Die „switch-Anweisung“ überprüft den Wert von der Variable „option“ und führt den entsprechenden Codeblock aus, der zu der ausgewählten Option gehört. Wenn eine ungültige Option ausgewählt wird, wird eine Fehlermeldung angezeigt und das Menu erneut aufgerufen

Zusammenfassend bietet dieses Programm dem Benutzer ein Menü, um zwischen verschiedenen Optionen zu wählen. Je nach ausgewählter Option werden verschiedene Aktionen wie das Starten von Servern und Clients oder das Anzeigen von Log-Dateien ausgeführt. Die Endlosschleife ermöglicht es dem Benutzer, wiederholt Optionen auszuwählen, bis er sich entscheidet, das Programm zu beenden.

TCP-Client

Der vorliegende Code ist eine TCP-Client-Anwendung, die eine Endlosschleife enthält, die dem Benutzer ein Menü mit verschiedenen Optionen anzeigt und auf Benutzereingaben wartet. Der Benutzer kann aus den folgenden Optionen wählen:

1. Starten des TCP-Clients: Dies ruft die Methode „creatingConnection“ auf, um eine Verbindung zu einem Server über TCP herzustellen und dann eine Chat-Funktion zu ermöglichen.
2. Starten von PCAP-LOG: Dies öffnet ein neues Terminal-Fenster und führt den Befehl „sudo ./pcapForTCP“ aus, um den PCAP-LOG zu starten.
3. Service-Informationen abrufen: Dies nutzt die Methode „serviceInformation“, um Informationen über einen bestimmten Dienst abzurufen.
4. Host-Namen abrufen: Dies nutzt die Funktion „hostName“, um Informationen über einen bestimmten Host abzurufen.
0. Beenden: Dies beendet das Programm.

Bei der Option „Starten des TCP-Clients“ wird eine TCP-Verbindung zu einem Server hergestellt. Der Benutzer wird aufgefordert, eine IP-Adresse einzugeben, woraufhin

eine Verbindung zum Server hergestellt wird. Anschließend kann der Benutzer einen Text eingeben, der an den Server gesendet wird. Zugleich werden empfangene Nachrichten vom Server angezeigt. Das Ganze läuft in einer Schleife, bis der Benutzer „exit“ eingibt.

Bei der Option „Service-Informationen abrufen“ wird der Benutzer dazu aufgefordert, den Namen eines Dienstes einzugeben. Das Programm sucht daraufhin nach diesem und gibt den Dienstnamen, den Port und das verwendete Protokoll aus.

Bei der Option „Host-Namen abrufen“ wird von dem Benutzer verlangt, einen Hostnamen einzugeben. Das Programm sucht dann nach dem Host und gibt den offiziellen Hostnamen sowie die zugehörigen IP-Adressen aus.

Wenn der Benutzer eine ungültige Option eingibt, wird eine entsprechende Fehlermeldung angezeigt.

Das Programm wird erst beendet, wenn der Benutzer die Option „0“ wählt, um das Menü zu verlassen.

TCP-Server

Der gegebene Code in „TCPServer.c“ implementiert einen einfachen Chat-Server, der es einem Client ermöglicht, Nachrichten an den Server zu senden. Der Server selbst antwortet hierbei mit einer Nachricht.

Der Code funktioniert wie folgt:

Zuerst wird eine Verbindung zwischen dem Server und einem Client hergestellt. Der Server wartet auf einen Client, der sich mit ihm verbinden möchte. Als nächstes wird der Server einen Socket mit „socket(AF_INET, SOCK_STREAM, 0)“ erstellen. Dieses wird für die Netzwerkkommunikation benötigt. Dann bindet er das Socket an eine bestimmte IP-Adresse und einen Port mit „bind()“. Ab dem Punkt startet das Hören auf dem Socket mit „listen()“. Danach akzeptiert der Server eine eingehende Verbindung von einem Client mit „accept()“. Dadurch wird ein neues Socket „connfd“ erstellt, der für die Kommunikation mit diesem Client verwendet wird.

In der „chatFunction“-Methode findet der eigentliche Chat zwischen Client und Server statt. In einer Schleife werden Nachrichten zwischen Client und Server ausgetauscht. Der Server liest eine Nachricht vom Client mit „read()“ und sendet seine Antwort mit „write()“ an den Client, wobei er überprüft, ob die Nachricht „exit“ lautet. Falls dem so ist, wird die Schleife und somit der Server beendet. Am Ende wird das Socket geschlossen.

UDP-Client

Dieser Code ist eine UDP-Client-Anwendung. Er stellt eine Verbindung zu einem Server her, sendet und empfängt Daten über UDP-Sockets und bietet auch einige zusätzliche Funktionen wie die Abfrage von Service-Informationen und Host-Namen.

Die Methode „runningChatFunction()“ stellt die Verbindung zum Server her und ermöglicht die Kommunikation. Der Benutzer gibt Nachrichten ein, die an den Server gesendet werden, und empfängt die Antworten des Servers. Der Austausch wird durch die Eingabe von „exit“ durch den Nutzer beendet.

Die Funktion „serviceInformation()“ ermöglicht die Abfrage von Service-Informationen. Der Benutzer gibt den Namen eines Dienstes ein und die Methode sucht die entsprechenden Informationen mithilfe der „getservbyname()“-Methode. Wenn der Dienst gefunden wird, werden der Name, der Port und das Protokoll des Dienstes ausgegeben.

Die Methode „hostName()“ macht es dem Benutzer möglich, den Host-Namen und die zugehörigen IP-Adressen abzurufen. Der Benutzer gibt einen Namen ein und die Funktion sucht und gibt den offiziellen Host-Namen und eine Liste der IP-Adressen aus.

Die „main()“-Methode stellt ein Menü dar, in dem der Benutzer verschiedene Optionen auswählen kann. Der Benutzer kann die UDP-Client-Anwendung starten, eine PCAP-Log-Anwendung starten, Service-Informationen abfragen, Host-Namen abfragen oder das Programm beenden. Je nach Benutzerwahl wird die entsprechende Methode aufgerufen.

Der Code enthält auch einige Eingabeüberprüfungen, um sicherzustellen, dass der Benutzer gültige Eingaben macht, und einige Fehlerüberprüfungen für die Socket-Operationen.

UDP-Server

Der Code ist für einen UDP-Server. Der Server wartet darauf, dass ein Client eine Nachricht sendet. Sobald eine Nachricht empfangen wird, sendet der Server dieselbe Nachricht als Antwort an den Client zurück. Der Server empfängt und beantwortet weiterhin Nachrichten, bis er die Nachricht „exit“ vom Client erhält. Daraufhin wird die Kommunikation beendet.

Es wird ein UDP-Socket mit der Methode „socket()“ erstellt. Das Socket wird mit „AF_INET“ für das Internetprotokoll „IPv4“ und „SOCK_DGRAM“ für den Datagramm-basierten UDP-Typ erstellt. Dann wird die Methode „bind()“ verwendet, um die Serveradresse an das Socket zu binden. Dadurch wird das Socket an den angegebenen Port und das Netzwerkinterface gebunden.

Eine Endlosschleife wird gestartet, um eingehende Nachrichten von den Clients zu empfangen und darauf zu reagieren. Mit der Methode „recvfrom()“ werden Daten vom Client empfangen, der empfangene Inhalt wird in den Puffer („buffer“) geschrieben. Wenn die empfangene Nachricht den Text „exit“ enthält, wird die Schleife beendet. Die empfangene Nachricht wird dann auf der Konsole ausgegeben. Mit der Methode „sendto()“ wird eine Kopie der empfangenen Nachricht als Antwort an den Client gesendet. Die Schleife läuft weiter und wartet auf weitere eingehende Nachrichten.

Zusammenfassend ist der Code ein einfacher UDP-Server, der auf eingehende Nachrichten von Clients lauscht, die Nachrichten verarbeitet und eine Kopie der Nachricht an den Client zurückschickt.

PCAP- Netzwerkmitschnitte (TCP & UDP)

Um die verschiedenen Netzwerk-Pakete des TCP- und UDP-Verkehrs kümmern sich zwei eigenständige Programme, sie betreiben das sog. „sniffing“, also das Einfangen der Netzwerkpakete. Das Programm „pcapForTCP.c“ teilt die Aufgabe nun in grob drei Teile auf.

Der erste Teil findet dabei in der Main-Methode statt. Hier werden die für PCAP benötigten Methoden aufgerufen. Zunächst speichert das Programm den Wert der „pcap_open_live“ für das Device „lo“ (loopback) in der „handle“ Variable. So kann in der nächsten if-Abfrage festgestellt werden, ob das bereits genannte Gerät überhaupt existiert, beziehungsweise ob PCAP den nötigen Zugang hat, um dem Netzwerkverkehr auf diesem Gerät zuzuhören.

Als nächstes wird der Filter erstellt, schließlich soll in diesem Programm nur dem TCP-Verkehr nachgegangen werden. Dieser Filter muss von der Methode „pcap_compile“ auf seine Gültigkeit überprüft werden, denn ansonsten könnte man auch ein ausgedachtes Protokoll benutzen und würde zu spät eine geeignete Fehlermeldung erhalten.

Nachdem der Filter also überprüft wurde, kann schließlich pcap auf TCP-Pakete angesetzt werden mit der Methode „pcap_setfilter“. Der Schritt danach beinhaltet dann das eigentliche Einfangen der Pakete. Die Methode „pcap_loop“ nimmt nun alle bereits in der „handle“ Variable gespeicherten Informationen und wartet jetzt darauf das ein TCP-Paket im Netzwerk verschickt wird. Falls dies geschieht, startet der zweite Teil des Programms.

Das eingefangene Paket wird dem „packetHandler“ übergeben. Diese Methode hat zwei einfache Aufgaben. Einmal eine Bestätigung auszugeben, dass ein Paket eingefangen wurde und außerdem das eingefangene Paket an die „parseTcpHeader“-Methode weiterzugeben.

Diese Methode verarbeitet das Paket so, dass es für Menschen lesbar ist. Zunächst müssen dafür jedoch ein paar Vorbereitungen seitens des Programms getroffen werden. Da ein Paket aus mehreren Header Schichten besteht, muss das Programm

erst die oberen Header aufschlüsseln, um zum eigentlichen TCP-Teil des Pakets zu gelangen. Als erstes kriegt also der „ipHeader“ einen Verweis auf den „ether_header“ (Ethernet Header). Dadurch können mit der „inet_ntoa“ Methode schonmal die Quellen IP Adresse und die Ziel IP Adresse übersetzt und in Variablen gespeichert werden.

Danach kann durch einen Verweis auf den IP-Header auf den TCP-Header zugegriffen werden. Dieser enthält Informationen zu den Ports, Flags, der Länge des Headers und die Sequenznummer. All das extrahiert das Programm mit den passenden Übersetz-Methoden (wenn nötig) und speichert alle Werte in weiteren Variablen ab. Im folgenden Schritt werden diese Variablen dann erneut aufgerufen und mit einer passenden Beschreibung ausgegeben.

Danach ermittelt das Programm über die „time“ und „localtime“ Methoden den genauen aktuellen Zeitpunkt, um für jedes Paket einen „Timestamp“ speichern zu können, damit die Identifizierung von Paketen etwas leichter wird.

Als letztes speichert das Programm die extrahierten Werte in einer LOG-Datei. Die „stdio.h“ Datei gewährt Zugriff darauf, eigene Dateien durch das Programm zu erstellen. Die Methode „fopen“ öffnet eine Datei namens „tcpheaders.log“ und fügt alle durch die Methode „fprintf“ gekennzeichneten Zeilen dem Dokument hinzu. Diese Zeilen enthalten noch einmal die gewonnenen Variablen von zuvor, sowie die passende Beschreibung.

Und schließlich wird das LOG-File geschlossen (und gleichzeitig gespeichert), was zur Folge hat, dass auch die letzte pcap Methode „pcap_close“ ausgeführt wird, womit das Programm beendet ist.

Für die UDP-Pakete ändert sich vom Grundaufbau nichts. Die nennenswerten Unterschiede in der Datei „pcapForUDP.c“ sind vor allem die Änderung des Filters auf das eben genannte „udp“ Protokoll und die Änderungen in der Header Parse Methode. Hier werden vom Programm nun die entsprechenden Informationen aus dem UDP-Header gezogen und nicht aus dem TCP-Header. Der IP-Header mit seinen Informationen bleibt jedoch unverändert. Schließlich werden auch hier alle extrahierten Informationen zusammen mit einem „Timestamp“ in einem LOG-File gespeichert.

Fazit

Zum Abschluss der Projektarbeit haben wir einige wichtige Erkenntnisse während des Prozesses gewonnen.

Zunächst einmal haben wir gemerkt, dass die Programmiersprache „C“ im Vergleich zu anderen Sprachen etwas veraltet ist. Ein Beispiel dafür ist die fehlende Unterstützung für objektorientierte Programmierung.

Außerdem haben wir über die verschiedenen Eigenschaften von den UDP- und den TCP-Standards erfahren. Die verschiedenen Einsatzgebiete der beiden Protokolle empfanden wir als sehr interessant, jedoch lag unsere Präferenz bei UDP, aufgrund des schnelleren Datenverkehrs.

Abschließend lässt sich sagen, dass das Arbeiten an diesem Projekt nach den anfänglichen Schwierigkeiten doch ziemlich reibungslos verlief. Der Kernaspekt der guten Zusammenarbeit lag an der passenden Aufgabenverteilung, je nach Stärken der einzelnen Gruppenmitglieder. Wir haben Vieles gelernt und uns Methoden angeeignet, damit das gemeinsame Arbeiten erleichtert werden kann.