

Vorlesung Rechnernetze

Laborübung

Einstieg in die Socketprogrammierung

Prof. Dr. Dirk Staehle

Die Abgabe erfolgt durch Hochladen der bearbeiteten Word-Datei in Moodle.

Bearbeitung in Zweier-Teams

Team-Mitglied 1:

Team-Mitglied 2:

1 Einleitung

In dieser Übung lernen Sie die Programmierung mit Sockets kennen. Die Funktionen zum Umgang mit Sockets sind in jeder Programmiersprache ähnlich, da Sockets vom Betriebssystem angeboten werden. In der Laborübung wird als Programmiersprache Python verwendet. Python ist eine moderne Interpreter-Sprache, die als einfach zu erlernen gilt. Python hat sich in den letzten Jahren zu einer der populärsten Sprache für die Entwicklung von Netzwerk- und Webanwendungen entwickelt.

Dieser Laborversuch besteht aus drei Aufgaben, die in zwei Laborstunden durchzuführen sind. Im ersten Versuch lernen Sie „telnet“ kennen. Telnet ist ein Tool mit dem Sie über die Kommandozeile in einen Socket schreiben bzw. aus einem Socket lesen können. Als Beispiel für telnet dient die tägliche Mail, die Sie dann auch mit einem Python-Skript schreiben dürfen. Der zweite Versuch dient zum Einstieg in die Socket-Programmierung mit Python. Sie implementieren hier einen einfachen Rechenserver. Das dritte Beispiel ist das Erstellen eines Port Scanners, um den Umgang mit Sockets, Threads und Timeouts weiter zu üben.

2 Vorbereitung

Wenn Sie mit Python noch nicht vertraut sind, arbeiten Sie sich in Python ein, indem Sie ein Python-Tutorial durchgehen. Python ist auf den Laborrechnern bereits installiert. Sie können beispielsweise Spyder oder IDLE als IDE nutzen.

Sockets werden in Python von der Bibliothek „socket“ unterstützt. Beschreibungen der Socket-Programmierung in Python finden Sie unter

1. <https://docs.python.org/3/howto/sockets.html>
2. <https://docs.python.org/3/library/socket.html>

3 Mail

In diesem Versuch verstehen Sie am Beispiel eines Mail-Clients, wie Anwendungen über Sockets kommunizieren. Sie lernen das Tool „telnet“ kennen, mit dem ein Socket geöffnet werden kann, um über die Kommandozeile mit einem Server zu kommunizieren. Im ersten Versuch, bauen Sie eine Verbindung zu einem Mail-Server, um über SMTP eine Mail zu schreiben und über IMAP Mails zu lesen. In einem zweiten Versuch implementieren Sie einen Mail-Client in Python, um Mails zu versenden oder abzurufen.

3.1 SMTP über telnet

Mit telnet können Sie einen Socket zu einem Server auf einem bestimmten Port öffnen und dann über Kommandozeilen-Eingabe mit dem Server kommunizieren. In diesem Beispiel sollen Sie über telnet mit dem SMTP Protokoll Mails verschicken. Öffnen Sie dazu mit telnet einen Socket zum (A)SMTP-Port (587) des Mail-Servers asmtpt.htwg-konstanz.de und melden Sie sich mit dem Account (Login: rnetin, Password: Ueben8fuer8RN) an. Schreiben Sie eine Email an einen ihrer Mail-Accounts und prüfen Sie, ob die Mail angekommen ist.

Hinweise:

1. Achten Sie darauf, dass Sie sich im VPN befinden.
2. Zeichnen Sie ihre SMTP Session mit WireShark auf.
3. Sie können „telnet“ entweder in der Windowskommandozeile oder über „Putty“ öffnen.
4. Gehen Sie vor wie beispielsweise auf Wikipedia beschrieben:

https://de.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol

<https://de.wikipedia.org/wiki/SMTP-Auth>

5. Login und Passwort müssen als Base64-kodierte Zeichenketten übertragen werden. Sie können die Konvertierung mit Hilfe eines Online-Tools durchführen oder die Python-Bibliothek base64 verwenden. Importieren Sie dazu die base64-Bibliothek in der Python Shell (`import base64`) und verwenden Sie folgende Befehle zum Konvertieren zwischen ASCII- und Base64-Zeichenketten:

```
(base64.b64encode('ASCII-String'.encode('utf-8'))).decode('utf-8')
```

```
(base64.b64decode('Base64-String')).decode('utf-8')
```

3. Die Antwort-Mail können alle Labor-Teilnehmer lesen, da alle den gleichen Mail-Account für ping asmtip die Laborübung nutzen.
4. Achten Sie bei der Eingabe von Text in telnet darauf, dass Sie sich nicht vertippen. Eine Korrektur ist nicht mehr möglich. Ein praktikabler Umgang mit diesem „Problem“ besteht darin, die Kommandos in einem Editor zu entwerfen und dann in das „telnet-Fenster“ zu kopieren. Das ist auch hilfreich für die nächste Aufgabe, wenn Sie den Mail-Client in Python implementieren.
5. Betrachten Sie die Aufzeichnung in WireShark. Was fällt Ihnen auf.
6. Schreiben Sie noch eine Mail an ihren Email-Account. Verwenden Sie willkürliche Email-Adressen für MAIL-FROM sowie für das „from:“-Feld in der Mail. Siehe auch Wikipedia Beispiel. Lesen Sie die Mail in ihrem Postfach. Was fällt Ihnen auf?

Hinweis: Die Erkenntnisse, die Sie hier erlangen dienen LEDIGLICH dazu, Sie darauf hinzuweisen, wie einfach es ist, Fake-Mails zu versenden. Bitte schreiben Sie keine Fake-Mails, auch nicht zum Spaß.

3.2 SMTP in Python

Implementieren sie einen SMTP-Client in Python. Benutzen Sie aber nicht die in Python vorhandene Mail-Bibliotheken (smtplib) sondern programmieren Sie direkt auf Sockets.

Hinweise:

1. Verwenden Sie die Python Bibliothek base64, um Login und Passwort in Base64-Code zu konvertieren.

2. Alle Daten werden im UTF-8-Format übertragen.
3. Alle gesendeten Zeilen müssen mit "\r\n" enden, damit der Mail-Server das Zeilenende erkennt. Auch diese Zeichen müssen „UTF-8“-codiert gesendet werden.

4 Rechen-Server

Implementieren Sie einen einfachen Rechen-Server in Python. Der Rechen-Server soll in der Lage sein, die Summe, das Produkt, das Minimum und das Maximum von n Zahlen zu bestimmen. Die Kommunikation mit dem Server erfolgt über Sockets. Eine Anfrage hat das folgende Format:

<ID><Rechenoperation><N><z1><z2>...<zN>

ID ist ein unsigned Integer (4 Bytes) und dient als Identifikator der Aufgabe. Rechenoperation ist eine der Zeichenketten „Summe“, „Produkt“, „Minimum“, „Maximum“ und wird „UTF-8 kodiert“ übertragen. Die Zahl N wird als Unsigned-Char-Wert (1 Byte) übertragen und gibt an, wie viele Zahlen folgen. Die Zahlen z1 bis zN werden als signed Integer (4 Bytes) übertragen. Verwenden Sie die Funktionen `pack` und `unpack` aus dem Modul `struct`, um die Nachrichten zu erzeugen. Sowohl `struct.pack` als auch `.encode()` liefern Bytes-Objekte, die einfach konkateniert werden können.

Das Ergebnis der Rechnung soll dem Client in folgendem Format zurückgeliefert werden:

<ID><Ergebnis>

Dabei ist ID definiert wie oben und Ergebnis soll ein signed Integer-Wert sein.

Implementieren Sie die Aufgabe mit Stream- und Datagram Sockets.

4.1 Lokale Kommunikation

Testen Sie die Skripte zunächst lokal, indem Sie für Client und Server die IP-Adresse 127.0.0.1 (localhost) verwenden. Starten sie dazu erst das Server-Skript und dann das Client-Skript jeweils in einem Windows-Cmd-Fenster. Nutzen Sie außerdem das Tool CurrPorts (<https://www.nirsoft.net/utils/cports.html>), um die aktiven Sockets zu überwachen.

Erklären Sie den Zusammenhang von ausgetauschten Paketen und Python-Code, indem Sie

1. für jedes gesendete Paket bestimmen, welcher Befehl in welchem Skript (Client/Server) dafür verantwortlich ist, dass das Paket gesendet wird
2. für jeden blockierenden Befehl bestimmen, die Ankunft welches Pakets dafür verantwortlich ist, dass die Ausführung des Befehls vervollständigt wird

4.2 Netzwerk-Kommunikation

Gehen Sie mit ihren Rechnern ins VPN. Konfigurieren Sie die IP-Adressen jetzt so, dass die Kommunikation im lokalen Netzwerk (im VPN) möglich ist.

Beantworten Sie die folgenden Fragen durch Experimente und unter Verwendung der Python-Hilfe:

1. Wie können Sie im Client Python-Skript die IP-Adresse und Port-Nummer des verwendeten lokalen Sockets bestimmen („bestimmen“ im Sinne von herausfinden)?
2. Wann (in welcher Code-Zeile) und woher erhält ein Client seine IP-Adresse und Port-Nummer?
3. Wie können Sie im Client-Skript die IP-Adresse und Port-Nummer des Sockets setzen?
4. Warum müssen Sie Timeouts verwenden und wie funktioniert try ... except? Mit welchem Befehl können Sie einen gemeinsamen Timeout für alle Sockets setzen?
5. Finden Sie experimentell heraus, ob Sie einen Server betreiben können, der ECHO-Anfragen auf dem gleichen Port für UDP und TCP beantwortet?

4.3 Unterstützung für mehrere Clients

Erweitern sie den Server mit Threads, so dass dieser mit mehreren Clients gleichzeitig verbunden sein kann.

5 Port Scan

5.1 Beschreibung

In diesem Versuch führen wir einen Port-Scan durch, um herauszufinden, welche Ports auf einem Server geöffnet sind. Wir scannen zum einen die standardisierten Ports von 1-50 nach offenen TCP Ports. Wenn wir einen offenen Port finden, versuchen wir, eine Nachricht an diesen Port zu schicken. Weiterhin vermuten wir, dass auf dem Server ECHO-Dienste für die Übertragung mit TCP und UDP laufen.

Ist ein TCP-Port auf einem Server geöffnet, dann wird ein Verbindungsaufbau auf diesem Port akzeptiert. Ist der TCP-Port nicht offen, so antwortet der Server entweder nicht (Windows Fehler-Code 10060) oder mit einem RST+ACK, in dem der Verbindungsaufbau zurückgewiesen wird (Windows Fehler-Code 10061).

Ist ein UDP Port auf einem Server geöffnet, so antwortet der Server entweder mit einer Nachricht oder überhaupt nicht. Die Reaktion hängt sowohl vom empfangenden Dienst als auch von der Nachricht selbst ab. Ist der UDP Port nicht geöffnet, so antwortet der Server entweder nicht oder mit einem ICMP Paket vom Typ 3, mit dem er mitteilt, dass das Ziel nicht erreichbar ist (Windows Fehler-Code 10054).

5.2 Versuch

5.2.1 TCP Port Scanner

Implementieren Sie ein Skript, das eine gegebene Anzahl von TCP-Ports auf einem gegebenen Server scannt und die offenen Ports zurückliefert. Führen Sie das Script für den Labor-Server 141.37.168.26 und Ports zwischen 1 und 50 durch. Zeichnen Sie die Kommunikation mit WireShark auf.

Starten Sie die einzelnen Port-Anfragen als Threads, um den Scan-Vorgang zu beschleunigen. Sie können einfach eine Funktion mit

```
t=Thread(target=<function>,args=(<arg>,,))
```

starten. Achten Sie darauf, dass der Thread beendet werden kann. Einfach geht dies mittels eines global Flags `Continue`, das der Thread kontinuierlich abfragt und sich bei `Continue==False` beendet.

Hinweis: <https://docs.python.org/3.4/library/threading.html>

5.2.2 UDP Port Scanner

Erweitern Sie das Script, um auch UDP Ports zu scannen. Führen Sie das Script für den Labor-Server 141.37.168.26 und Ports zwischen 1 und 50 durch. Zeichnen Sie die Kommunikation mit WireShark auf. Unterscheiden Sie Ports, auf denen Sie keine Antwort bekommen und Ports, auf denen Sie Fehlermeldung 10054 erhalten.

5.3 Fragen

1. Geben Sie die Liste der offenen TCP und UDP Ports an.
2. Wählen Sie für TCP und UDP jeweils einen offenen und einen geschlossenen Port und erklären Sie die entsprechende Paketsequenz, die Sie in WireShark aufgezeichnet haben.
3. Auf Port 7 des Servers läuft ein ECHO-Dienst. Testen Sie ihr Client-Script mit dem ECHO-Server. Versuchen Sie das TCP und das UDP Script.