Lab8

Socket Programming&
Final project

Computer Communication Networks

Roey Salah 206115438
Liel Sinn 209195155
Pair-num:8

Part 1:

1.13

For the TCP receiver:

The functions used are socket() and connect ().With socket() a new socket is created ,for TCP the function will be with the arguments: socket(AF_INET, SOCK_STREAM, 0). The connect() function is establishing a connection to the server(the 3 way handshake of TCP):

connect(server_sock, (struct sockaddr*) &server_addr, sizeof(server_addr))

For the TCP sender:

The functions used are socket(),bind(),listen() and accept(). With socket() a new TCP socket is created: socket(AF_INET, SOCK_STREAM, 0). With bind() the server address is bind to the socket opened. With listen() a queue of the socket for incoming connection requests is created and with accept(), incoming connection requests are transferred from the queue to the socket of the sender.

1.14

The listen() function is used to create a queue for the socket of the sender for incoming connection requests: listen(lisen_sock, SOMAXCONN)

The lisen_sock is the description file of the socket of the sender and the SOMAXCONN is the length of the queue created.

1.15

Packets sent by the sender can be lost, however it is not so likely. The reason is that the TCP protocol is more "sympathic", this means that the rate of sending will be according to the flow control (how many packages the receiver can still receive according to the buffer) and the congestion control (how much traffic there is in the network on the way between the receiver and the sender). Yes, there are steps the program can be execute to verify that all packets arrived.For example an Ack packet can be send with the sequence num of the last packet arrived. According to the sequence number the buffer at the sender will be retransmitted.

1.16

I could used an defined ip address. For example INADDR_ANY is an defined address in the libraries :
<sys/socket.h>
<netinet/in.h>
<netinet/ip.h>

In the code the change would be:

```
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
```

(htonl is converting the host to network order of big/little endian and INADDR_ANY is the address of define.)

instead of:

```
server_addr.sin_addr.s_addr = inet_addr(atoi(argv[1]));
```

(inet_addr interprets character strings representing host addresses and returns host addresses tha are suitable for use as an server address and atoi is converting the given string representation to an integer .)

שאלה 1.17

הצד המקבל יודע מתי הצד השולח סיים לשלוח הודעה הפרוטוקולים TCP מחזיר ערך שם O (הודעה מגיעה לקליינט מידע מהצד השולח לתוך בופר מסויים), הודעה מקבלים O פשוט נדע שהצד השולח סיים לשלוח את כל שרשרת הפקטות והם סיגר את הקישוריות.

שאלה 1.18

1.1 כן הי אינטרנט יהיה כי נשלחו מספר התבלות שונים (seq_num) והמקבל לשלוח מאחור אומר ב- TCP אל יש הקשורה אמינה, מה יש איבוד של שולחים שוב.

1.2 הצד המקבל משרת בפרוטוקולים TCP שמקבלת פרמטרים, אחד מהפרמטרים זה הפקסר אלין הם רוצה לשמור את המידע שהקבל ולכן השימוש בפרוטוקולים הם לא בהכרח במספר התבלות (seq_num) אלא עד שהפקסר שהגדרנו יקבל את המידע שהם בקש.

1.3 יכול להיות מצב שיש איבודים בדרך ולכן מאחר ואמר ב- TCP יש צורך לשלוח בחזרה התבלות שבדקו מאחר והצד המקבל דיות שמצפה למשל יחבולה לפבית מסויים שאז האיבר מבין עקב תבלה בדרך.

שאלה 1.19

כפי שציינו בשאלה הקודמת, הצד השולח מקבל חיווי ACK אם לא הלכן שהוא שולח מהצד המקבל, פעמים אים נבדק שימוש בפרוטוקולים לחשל ו- TCP נהל לדעת את בעת המידע הגיע בלבן או שהיתה תקלה בדרך, וכך בעצם הצד השולח יודע אים לשלוח שוב מאחר ו- TCP אומרים בהקשר של שליחת המידע.

Part 2:

2.13

For the UDP sender:

The functions used are socket() and bind ().With socket() a new socket is created ,for UDP the function will be with the arguments: socket(AF_INET, SOCK_DGRAM, 0). The bind() function is binding server address to the server:

bind(lisen_sock, (struct sockaddr*)&listen_addr,

sizeof(listen_addr).

For the UDP receiver:

The functions used are socket() and bind (), while here the bind function is no must and the address is of the receiver.

2.14

The differences between :

the receivers:

In both a socket is created. The difference is in the creation is one specific argument: for UDP its SOCK_DGRAM and for TCP its SOCK_STREAM. Moreover the TCP receiver calls connect() to establish the connection.

the senders:

As with the receivers also here, both are creating a socket . The difference is in the creation is one specific

argument: for UDP its SOCK_DGRAM and for TCP its SOCK_STREAM. For the TCP connection the listen() function is required to establish a queue for the incoming connection requests that are getting to the sender and in order to transfer an incoming connection requests from the queue to the socket of sender accept is used. In UDP both of this functions are not used.

This differences attend are there since the UDP protocol is unreliable while the TCP is. The UDP needs no queue, if a package doesn't arrives at the destination nothing is done about it. Moreover the TCP is establishing its connections with a 3 way handshake, that’s another reason for the need of a queue and the accepting process.

2.15

By Chens mail-> this question is canceled

2.16

As in the question given Alice's program is working only the first time running and in the second not. The possible reason of the problem is that to each port only one socket can be bind to. As a result, if a new socket is opened and the socket is bind to the same port it causes the error given.

A possible solution is to attach the new port to a port that is not in use yet with help of a while loop and an if statement.

The Drawback is that no port can be used in the same time for two sockets and the runtime of the program is getting loger as a result of the additional while loop .

Another possibility is that after the first program ended, the socket was not closed and this means that by the time Alice is running the program again, the socket is still open. In the program a new socket is opened and bind to the given address. Now in this situation two sockets are trying to be bind to the same address which causes the error.

A simple solution could be to close the socket in the end of the program.

שאלה 6 3 - הפרמטרים השונים שהפונקציה select מקבלת הם:
nfds, readfds, writefds, exceptfds, timeout.
כאשר נפרט על כל אחד:
nfds - ארגומנט שבו עובר ה- descriptor file הגבוה ביותר (במספרו) בכל אחד משלושת הסטים (ועוד 1 שמוסיפים).
ה- descriptor file בכל סט נבדק עד למספר שארגומנט זה.
readfds - ה- descriptors file בסט זה נבדקים כדי לדעת אם הם מוכנים לקריאה.
writefds - ה- descriptors file בסט זה נבדקים כדי לדעת אם מוכנים לכתיבה.
exceptfds - ה- descriptors file בסט זה נבדקים עבור מצבים חריגים שמתרחשים.
timeout - ארגומנט זה ובעזרתו אין רוצים זמן שבו הפונקציה select תחסום/תבצע חסימה עבור זמן המתנה ל- descriptor file שיהפוך זמין.

בדרך שלנו השתמשנו בשני השדות הראשונים שהם nfds ו- readfds, כל שאר השדות הדבירנו NULL (שפה timeout וכל מה אפס) מאחר ולא עשינו בהם שימוש בקוד שלנו.

שאלה 37 - הפונקציה select מחזירה ערכים: 0, -1, -1
נסביר כי ערך החזרה בפורק. נניח שערך החזרה אי
* עבור ערך החזרה שהוא 0, הפונקציה מופעלת בעקבות
שפקע ה- timeout לפני שכל descriptors file הפכו
למוכנים (מוכן לעבוד איתו בעתיד).
* עבור ערך החזרה -1, צב סימן לשגיאה, ומה שמעניין זה
רק צב האלה של errno כדי לחדוע זה השגיאה הזו,
שצב בעתיד אומר כי ההשמה ל- descriptor file לא נשמרה
כמו שנדרש.
* עבור ערך החזרה שגדול מ- 1, הפונקציה בעתיד
מחזירה את מספר ה- descriptors file שמוכנים
בשלושת הסטים של ה- descriptor שמוחזר (מוגדר סדר
הבטים שלו ב- readfds, writefds ו- exceptfds)

שאלה 3.8 - כן ניתן להשתמש בטבלאות, ובכל פעם לבדוק בפיקה עם FD_ISSET שבודקת האם ב- socket מסויים הייתה ניסיון להתחבר (נמסיון לשלוח מידע), וכך בעצם אנו עוברים בלולאה ובודקות עם יש ניסיונות שמתנשים להתחבר. וכך בעצם בודקים אם כן socket את יש מידע שנשלח בו או לא.

שאלה 3.9 - שימוש מסוג לפונקציות select הם למשל בלינוקס בעזרת הפונקציה מתאימה/משנה את פסק הזמן את השיחה מופרעת ע"י מטפל/מחזיק סיגנלים.
שימוש נוסף לפונקציה select הוא לבדוק סינכרון I/O (שגם קלט פלט).