פרויקט גמר: דף שעררררררר סומך עלייך בקטע הזה נירוס

**הסבר על הקוד:**

ראשית, נצפה ונסביר את ההקלטת wireshark. רצף הפעולות שקורות בהקלטה הן:

1. חיבור של המשתמש לסרבר
2. המשתמש מכניס שם משתמש
3. הסרבר בודק האם השם אינו תפוס, במידה ולא מחזיר username accepted
4. לאחר מכן, המשתמש מכניס את הפקודה <download><bigfile.txt>, פקודה זו מבקשת מהשרת להוריד את הקובץ bigfile.txt
5. ההורדה מתחילה, לאחר מעבר של 50% מהקובץ, הסרבר מחכה לתשובה ממשתמש כדי להמשיך באמצעות הפקודה <proceed>, לאחר פקודה זו ההורדה ממשיכה.

בגדול, מה שנצפה לראות בהקלטה, בלוק של TCP שמהווה את הפעולות שקרו בסעיפים 1,2,3 ולאחר מכן בלוק של UDP שמהווה את הפעולות שקרו בסעיפים 4,5

**נסביר את מהלך ההקלטה:**

ראשית, לאחר חיבור של המשתמש לסרבר, המשתמש בוחר את השם שלו, במידה וניתן להשתמש בשם זה הסרבר מחזיר username accepted וכך בעצם הקליינט יודע האם הוא יכול להשתמש בשם זה. במידה ולא, תתקבל הודעת שגיאה, והשרת יתן למשתמש ניסיון חוזר.

Table

Description automatically generated

לאחר מכן, השרת מחזיר חיווי למשתמש שהוא התחבר בהצלחה לשרת, <connected>

Table

Description automatically generated

לאחר שהמשתמש מחובר לשרת, נרצה להוריד קובץ מהשרת, שם הקובץ הוא bigfile.txt, הודעה זו נשלחת מעל UDP.

Table

Description automatically generated

לאחר מכן, השרת מחזיר שהוא מתחיל לשלוח את ההודעה, כאשר המשתמש מקבל את הודעה זו, הוא מפעיל את הפונק' שמטפלת בקבלת הקובץ מהשרת.

Table

Description automatically generated

בתחילה, שולח השרת את מס' הפאקטות שהוא עתיד לשלוח, במקרה הנ"ל 26, ורק לאחר מכן מתחיל בשליחת הקבצים.

Table

Description automatically generated with medium confidence

לאחר שליחה של מס' הפקאטות, מתחילה בעצם שליחת המידע, תוך החזרה של ACK's מהקליינט

Table

Description automatically generated

ACK:

Graphical user interface, application, Word

Description automatically generated

מהלך זה מתבצע למעשה עד לשליחה של 50% מהקובץ, כשאר נגיע לשליחה של 50% השרת יחכה להודעה מהמשתמש על מנת להמשיך בשליחה הקבצים. <proceed>

Graphical user interface, text, application

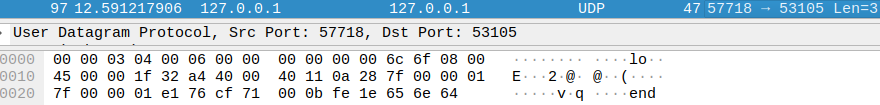
Description automatically generated

לאחר שהמשתמש ביקש מהשרת להמשיך, השרת מחזיר לו <proceeding> על מנת שהמשתמש ידע שהשרת קיבל את ההודעה והוא ממשיך, לאחר שהמשתמש מקבל את ההודעה הזו, הוא יודע להמשיך את פעולת קבלת הקובץ.

Table

Description automatically generated with medium confidence

בשלב זה, ממשיכה ההורדה – שליחת הקבצים מצד השרת והחזרת ה ACK's מצד המשתמש. בסוף קבלת כל המידע הדרוש, המשתמש מחזיר הודעת end לשרת, וכך השרת יודע שכל הקובץ התקבל במלואו.



**מבנה הקוד:**

לאחר שעברנו על ההקלטה, נרצה להסביר באופן מפורט על האלגוריתם שלנו ומבנה המחלקות.

**MyServer.py**

מחלקה זו, כפי שניתן להבין משמה, מאגדת בתוכה את כל פעולות השרת.

השרת מחזיק יומן של המשתמשים עם הסוקטים שלהם, רשימה של הקבצים שניתן להוריד מהשרת, וסוקטים מסוג UDP ו TCP.

כדי להפעיל את השרת, יש לייצר אובייקט מסוג MyServer ולהפעיל את הפונק' accept client באמצעות אובייקט מסוג MyServer.

**def accept\_clients:**

למעשה, פונק' מאפשרת קבלת משתמשים אל השרת. עבור כל משתמש שמתחבר היא בודקת האם השם שבחר פנוי ומחזירה שגיאה במידה והשם שגוי לאחר שהתקבל שם שאינו תפוס הפונק' מחזירה חיווי על התחברות בהצלחה. בנוסף, הפונק' יוצרת שני תהליכים שמאזינים לפקטות מסוג TCP ו UDP.

**def handle\_udp:**

פונק' זו מופעלת ע"י תהליך מהפונק' accept\_clients. היא למעשה מאזינה לפקטות מסוג UDP בתהליך נפרד ופועלת בהתאם למידע שקיבלה מהמשתמש. כאשר היא מקבלת הודעה לבקשה להורדה היא יוצרת סוקט חדש מסוג UDP ואותו שולחת לפונק' send\_file. היא עושה זאת על מנת לאפשר הורדה של קבצים במקביל, כי על אותו פורט ניתן לעבור רק מול משתמש אחד בו זמנית. לאחר סיום ההורדה השרת סוגר את הסוקט.

**def handle\_client:**

פונק' זו מופעלת ע"י תהליך מהפונק' accept\_clients. היא למעשה מאזינה לפקטות מסוג TCP בתהליך נפרד ופועלת בהתאם למידע שקיבלה מהמשתמש. אחראית על כל העברת ההודעות הרגילה, בין משתמשים ובין השרת ללקוח. הודעות כמו בקשה לקבל את כל המשתמשים, הודעות, קבצים וכו'.

**def send\_file:**

פונק' זו שולחת את הקובץ אל המשתמש שביקש את הקובץ הנ"ל. לאחר שהפונק' handle\_udp קיבלה בקשה להורדה, היא מפעילה את הפונק send\_file, פונק' זו בתחילה עושה split, כלומר לוקחת את הקובץ שהיא צריכה לשלוח ומפצלת אותו לפקטות באורך ה buffer. כשאר משורשר לכל פקטה גם מס', שבאמצעותו המשתמש ידע מה מס' הפקטה שהתקבלה וכך יוכל להחזיר את ה ACK המתאים. לאחר מכן, שולחת חיווי למשתמש שהיא מתחילה לשלוח את הקובץ ושולחת את גודל הקובץ. לאחר מכן היא מתחילה בשליחת המידע. לאחר שנשלח 50% מהקובץ ע"י קבלת הACK's מהמשתמש, השרת מחכה לבקשה מהמשתמש להמשיך, לאחר קבלת בקשה זו הוא גם מחזיר לו שהוא אכן ממשיך. השרת מפסיק לעבוד כאשר הוא מקבל הודעת end.

**Client.py**

מחלקה זו, כפי שניתן להבין משמה, מאגדת בתוכה את כל פעולות המשתמש.

כל אובייקט מסוג Client מחזיק בתוכו סוקט מסוג TCP ו UDP, שם משתמש ורשימה של ההודעות שקיבל.

מלבד האתחול של השדות הנ"ל, הקונסטרקטור קולט את השם משתמש ומוודא מול השרת שאכן ניתן להשתמש בשם זה. לאחר סיום האתחול, הקונסטרקטור מפעיל תהליך (thread) המקשיב להודעות מסוג TCP.

**def listen\_for\_messages:**

מאזין להודעות שמקבל מהשרת ומפעיל את הפונק' בהתאם. בנוסף, ממוסיף את ההודעות לרשימה שמאגדת בתוכה את כל ההודעות שקיבל המשתמש.

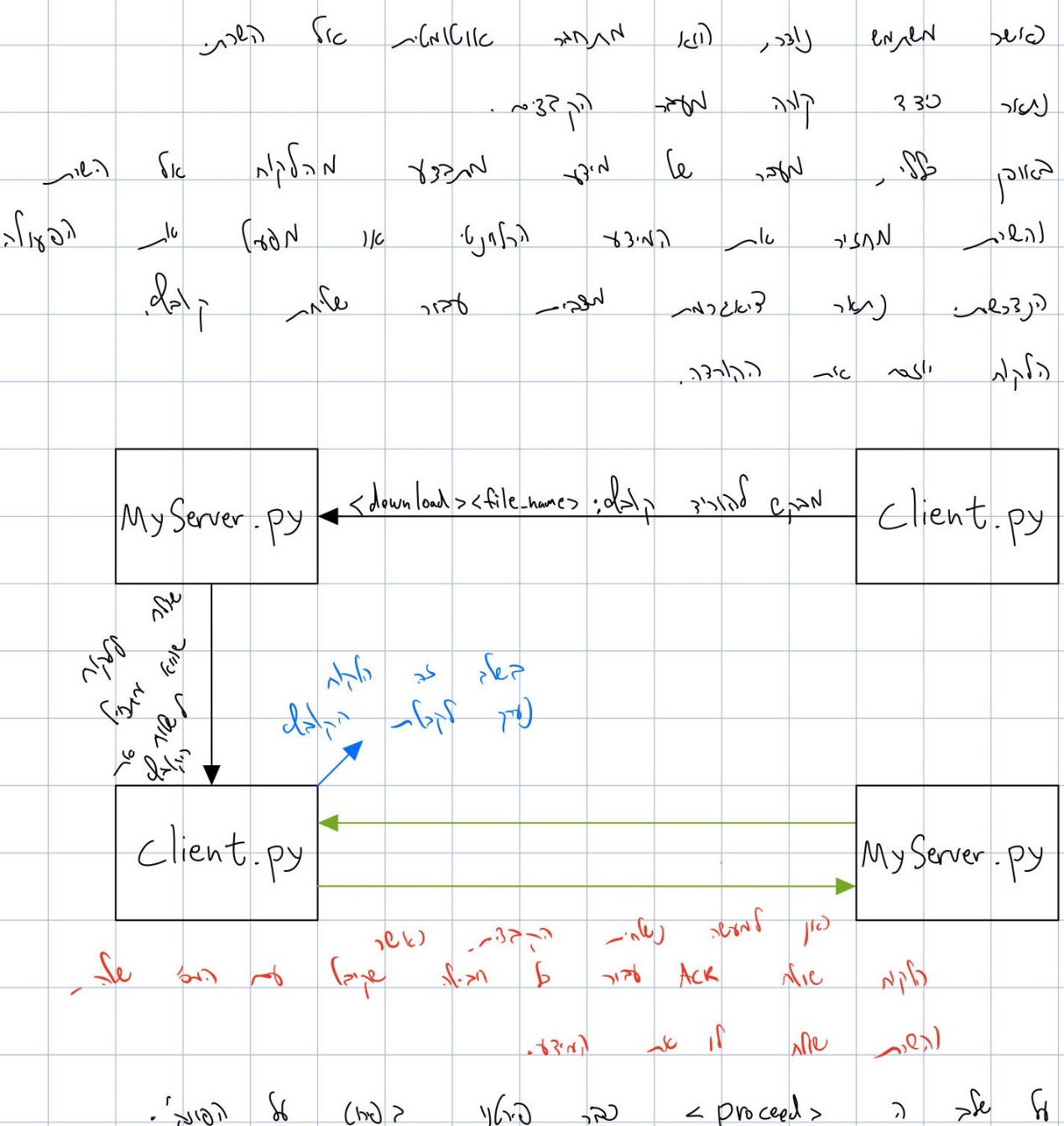
**def send\_message:**

פונק' זו, קולטת מאזינה לקלט מהמשתמש ושולחת אותו אל השרת באמצעות קשר TCP.

**def get\_file:**

פונק' זו מופעלת כאשר הפונק' listen\_for\_messages מקבל מהשרת כי הוא מתחיל בשליחה של קובץ, כאשר השרת שולח הודעה זו בעקבות בקשה של המשתמש (מעין 3-hand shake), פונק' זו מקבלת את המידע מהשרת, שומרת אותו ברשימה ומחזיקה ACK's בהתאם למס' הפאקטה. כאשר מתקבל 50% מהמידע, השרת מחכה להודעה מהמשתמש להמשך ההורדה של הקובץ. לאחר שהמשתמש מבקש את המשך השליחה, הפונק' ממשיכה בקבלת הקובץ. לאחר שקיבלה את כולו, מתחילה הפונק' בכתיבה של המידע על גבי קובץ.

**דיאגרמת מצבים:**



**איבוד פאקטות:**

כדי להתמודד עם איבוד פאקטות, כל חבילה שהלקוח מקבל, הוא מחזיר ACK עם מס' החבילה. (הלקוח יודע מה מספר החבילה כיוון שהיא משורשרת במידע שהוא מקבל מהשרת) כלומר, ACK0 אומר לשרת שהלקוח קיבל את הפאקטה הראשונה. וכן הלאה.  
במידה והלקוח החזיר NACK זה אומר שהפקטה שהתקבלה לא הגיעה בסדר הנכון, כלומר בגלל שהפקטות נשלחות לפי סדר, ישנו סיכוי שאם קיבלנו פקטה אחת (למרות שנשלחה בשלב מאוחר יותר) לפני פקטה אחרת, אז יש לנו חבילה שאבדה ברשת. במצב כזה, הלקוח ישלח NACK עם מס' הפקטה שהייתה אמורה להגיע. לכן, עבור הודעת NACK שהשרת מקבל מהלקוח, השרת ישלח שוב את החבילה שהייתה אמורה להגיע ליעדה. (selective repeat)

**בעיות latency:**

תחילה נסביר כי בעיות latency הן בעיות הנובעות מכך שמרגע התחלת פעולה מסויימת, עד הרגע שהוא באמת מתחיל. במילים פשוטות, בעיות השהייה.

כאשר הלקוח רוצה לעשות פעולה מסויימת, הוא שולח הודעה לשרת, השרת מאשר קבלה, ורק לאחר אישור הקבלה הלקוח מתחיל בפעולה שרצה לעשות.  
למשל, בשליחת קובץ, הלקוח מבקש להוריד קובץ מסויים, לאחר מעבר של 50% השרת מחכה להודעה מהלקוח על מנת להמשיך. הלקוח ממשיך בפעולת הקבלה רק לאחר קבלת המשך מהשרת. כך אנחנו יודעים שהשרת מוכן והוא ממשיך לשלוח הודעות. באופן דומה גם תחילת שליחת הקובץ מבצעת בדרך זו.

דוגמא נוספת, כאשר המשתמש רוצה להתנתק מהשרת, הוא שולח את ההודעה הזו לשרת והשרת יודע לנתק אותו. אך כדי שהוא אכן יקבל את ההודעה, עליו להמתין זמן מסויים כדי לקבל מהשרת את ההודעה שהוא התנתק. לכן השתמשנו בsleep קצר שיחכה לקבלת התשובה מהשרת.

**How To Run:**

1. כדי להפעיל את השרת יש להריץ את הקובץ MyServer.py
2. לאחר מכן, כדי ליצור קליינט יש להריץ את Client.py ולהכניס ע"י קלט את השם משתמש. במידה ורוצים להוסיף עוד משתמש, יש להריץ את אותו הקבוץ מטרמינל אחר.
3. על מנת לשלוח הודעות למשתמש כלשהו, יש לכתוב את השם של המשתמש, ללחוץ על Enter ולאחר מכן את תוכן ההודעה. כדי לשלוח לכולם, יש לכתוב all, ללחוץ על Enter ולאחר מכן את תוכן ההודעה.
4. על מנת לקבל את רשימת ההודעות שהמשתמש קיבל יש לכתוב: <get\_messages>
5. על מנת לקבל את רשימת המשתמשים יש לכתוב: <get\_users>
6. על מנת לקבל את רשימת הקבצים יש לכתוב: <get\_files>
7. על מנת להתנתק יש לכתוב: <disconnect>
8. על מנת להוריד קובץ, יש לכתוב: <download><file\_name> שם הקובץ צריך להיות מלא כולל סיומת, לפי הרשימה שמופיעה אם תכתוב את <get\_files>
9. על מנת להמשיך בהורדת הקובץ, יש לכתוב: <proceed>