***רשתות תקשורת פרויקט סיום – SQL***

***מגישים: אביב תורג'מן - 208007351***

***אלון סויסה – 211344015***

***תוכן עניינים:***

הסבר כללי על הפרויקט............................................. 3

הוראות קימפול והרצה............................................... 4

client.py.................................................................

**הסבר כללי על הפרויקט:**

אנחנו בחרנו לעשות את הפרויקט SQL על קבוצות מליגת האלופות. המידע של טבלת ה-SQL הוא השחקנים של הקבוצות והנתונים שלהם (שם, דירוג, קבוצה, תפקיד, גולים, בישולים).

תחילה יצרנו מחלקה בשם PL\_player.py שמאתחלת שחקן לפי הנתונים הנ"ל ונותנת גישה לנתונים אלו עם פונקציות get.

לאחר מכן יצרנו מחלקה חדשה בשם DATA.py שתחזיק את כל המידע של הקבוצות והשחקנים בתוך מערך של PL\_player שנקרא data. בנוסף, יצרנו מחלקה שנקראת query\_object.py שמטרתה היא ליצור שאילתות ויש לה מתודה אחת בשם do\_query אשר מחזירה מערך של PL\_player לפי השאילתה.

לאחר מכן יצרנו 2 שרתים (TCP\_server.py, UDP\_server.py)  
שיש להם גישה ל-DATA ותפקידם הוא להחזיר ללקוח תשובה לשאילתות על השחקנים.

בנוסף, יצרנו מחלקה שנקראת client.py אשר מממשת את צד הלקוח ותפקידה הוא לשאול את השרת (איזה שתבחר) שאילתות לגבי השחקנים ולהציג את התשובה על המסך.

על מנת שלא נכתוב קוד כפול של פונקציות למימוש הקשר בין השרת ללקוח, יצרנו מחלקה שנקראת functions.py אשר מחזיקה 4 מתודות (send\_with\_cc, increase\_window, receive, checksum).

**הוראות קימפול והרצה:**

תחילה יש להתקין את הספרייה pygame:

**windows/ MAC OS:**

<https://www.pygame.org/wiki/GettingStarted>

**Linux:**

<https://www.geeksforgeeks.org/install-pygame-in-linux>

לאחר שהתקנו pygame, יש להריץ את השרתים: DNS, DHCP:

python DNS.py

python DHCP.py

אחר כך יש להריץ תחילה את אחד השרתים  
(TCP\_server.py, UDP\_server.py) בטרמינל נפרד כך:

python TCP\_server.py **או** python UDP\_server.py

**לאחר ששלושת השרתים האלו רצים**, יש להריץ את client.py ולבחור ב- UDP/TCP בהתאם לשרת אותו בחרתם להריץ.

**הערות:**

* ב- linux יש לכתוב python3 במקום python.
* ניתן להריץ את כל השרתים ב-PyCharm אך יש לשים לב   
  שה- configuration מוגדר ל- python3 ולא מתחת.
* את הפרויקט עשינו על ווינדוס לכן על מנת לחוות חוויה מלאה של הפרויקט (סאונד וגופנים של הכתב), יש להריץ על ווינדוס.

**TCP\_server.py:**

למחלקה זו יש גישה לספריות:  
socket, threading, pickle, PL\_player, query\_object, DATA  
על מנת לפתוח קשר עם כמה לקוחות ולשלוח את המידע הנחוץ.  
תחילה הגדרנו כמה משתנים קבועים (Caps Lock) הגדרנו משתנה בשם LEN\_HEADER\_SIZE = 8 אשר יהווה את גודל ה-header שהוספנו להודעה, header זה יגיד לנו מהו גודל ההודעה. לאחר מכן הגדנו כי הפורמט שבו "נצפין"/"נפענח" את המידע הוא 'utf-8' ושגודל הצ'אנקים של המידע שנשלח יהיו 32 בתים. הגדרנו את ה-PORT ל-30015, וכדי לקבל את ה-IP של מחשב זה השתמשנו ב-

IP = socket.gethostbyname(socket.gethostname())

לאחר מכן יצרנו משתנה מסוג טאפל בשם ADDR מה-IP וה-PORT שיצרנו שהוא הכתובת יעד לסוקט ופתחנו TCP סוקט ליצירת קשר.

ADDR = (IP, PORT)

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  # TCP socket  
server.bind(ADDR)  # binding the address

למחלקה זו יש 3 מתודות:

* **handle\_client(conn, addr) –** מתודה זו מקבלת 2 פרמטרים:  
  1. conn – אובייקט מסוג סוקט של הלקוח בו מטפלים  
  2. addr – הכתובת של אותו לקוח   
   מתודה זו אחראית על מימוש קבלת השאילתה מהלקוח ושליחה של המידע המתאים ללקוח.  
  המתודה מקבלת מידע מהלקוח ע"י פונקציית recv,   
  ומפלטרת לפיו. אם ההודעה היא שאילתה, אז נקבל את התשובה לשאילתה עם המתודה filter\_by\_queries(queries), ונשלח את התשובה בחזרה ללקוח עם פונקציית send.  
  נעיר על כך שנשתמש בספרייה pickle ובייחוד במתודות pickle.loads ו-pickle.dumps על מנת להמיר את המידע שנקבל מבתים למחרוזות וההפך כיוון שהמידע שנשלח הוא בבתים.  
  המתודה ממשיכה לעבוד בלולאה אינסופית ב-thread שמותאם ללקוח עד אשר נקבל מהלקוח הודעת יציאה.
* **start() –** מתודה זו יוצרת thread חדש לכל לקוח שמנסה להתחבר עם השרת ומקצה ל-thread את handle\_client עם הסוקט והכתובת של הלקוח כמשימה.
* **filter\_by\_queries(queries)** **–** מתודה זו מקבלת פרמטר בשם queries שהוא רשימה של אובייקטים מסוג query\_obj ממחלקת query\_object.  
  לאחר קבלת השאילתות כפרמטר, המתודה פונה למחלקת query\_object, מבצעת את השאילתות שיש ברשימה ומחזירה רשימה של PL\_player אשר מהווה תשובה לשאילתות שהתקבלו כפרמטר.

**UDP\_server.py:**

למחלקה זו יש גישה לאותם ספריות כמו TCP\_server.py ואותם משתנים קבועים חוץ מ- LEN\_HEADER\_SIZE.  
בנוסף למשתנים אלו, הוספנו למחלקה הזו את המשתנים הבאים:  
PORT\_CHANGE – משתנה אשר בעזרתו ניצור פורט חדש לכל לקוח.  
clients[] – רשימה של לקוחות.  
change\_port\_lock, client\_lock – שני מנעולים שנועדו כדי לעדכן את שני המשתנים לעיל באופן אסינכרוני.  
באותו אופן של TCP, נפתח סוקט UDP ונקשר אותו.

server = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  # UDP socket

server.bind(ADDR)  # binding the address

למחלקה זו קיימים 6 מתודות:

* **filter\_by\_queries(queries) –** אותה פונקציה כמו של השרת TCP.
* **get\_change\_port() –** מתודה זו מחזירה את מספר הסוקטים שפתחנו עד כה על מנת ליצור פורט חדש בהתאם ללקוח החדש.
* **add\_client(client) –** מתודה זו מקבלת כתובת ומוסיפה אותה לרשימת הלקוחות (clients[]).
* **remove\_client(client) –** מתודה זו מקבלת כתובת ומוחקת אותה מרשימת הלקוחות (clients[]).
* **start() -** מתודה זו יוצרת thread חדש לכל לקוח שמנסה להתחבר עם השרת ומקצה ל-thread את handle\_client עם הסוקט והכתובת של הלקוח כמשימה, ומוסיפה את הכתובת של לקוח זה לרשימת הלקוחות (clients[]).
* **handle\_client(ip, port, chunk) –** מתודה זו מקבלת ip, port ו-chunk. ה- ip וה- port של הלקוח ליצירת טאפל בשם addr אשר מהווה את כתובת הלקוח, ו-chunk אשר יהווה את גודל הסגמנטים שיתקבלו למימוש Flow Control.  
  תחילה ניצור 5 משתנים: סוקט UDP, פורט ייחודי ללקוח, כתובת השרת ,דגל השווה ל-1 ומשתנה שיאחסן את המידע של ההודעה.  
  לאחר מכן נקשר את סוקט ה-UDP לכתובת השרת, נדאג לשחרר את הפורט הנוכחי בעת סגירת הקשר ע"י setsockopt ונדאג שפונקציית socket.recvfrom() לא תהיה פונקציה חוסמת ע"י שימוש ב- socket.setblocking(False) .  
  לאחר מכן נסיים את פתיחת הקשר.  
  אחר כך נקבל מהלקוח שאילתות / הודעת יציאה ונשלח לו תשובה בהתאם.

**DNS\_server.py:**

מחלקה זו עובדת בדיוק כמו מחלקת TCP\_server.py רק שבמקום לקבל שאילתות ולהחזיר תשובה לפי השאילתה, היא מקבלת מילת מפתח לחיפוש כתובת של השרת איתו הלקוח מתקשר,  
מחפשת את הכתובת במשתנה מסוג מילון שיצרנו שממפה כתובות ושולחת ללקוח את הכתובת של שרת זה.

**הערות:**

* הקשר מתבצע באמצעות סוקט TCP
* בשימוש שלנו בפרויקט זה, יש ל-DNS רק כתובת אחת אך הוא יכול להחזיק **יותר** מכתובת אחת ולהחזיר את הכתובת הרצויה בהתאם לבקשת הלקוח

**DHCP\_server.py:**