מגישים:

Michael ozeri 302444229

Ozeri.mich@gmail.com

Itai Shchorry 305584690

itaishch@gmail.com

Oded Navon 303066674

oded.navon@gmail.com

טלפון (זמין 24/7) : 0547939912 (מיכאל)

תיאור הפרוטוקול:

- מימשנו צד שרת בקובץ server.c וצד לקוח בקובץ client.c.
- פרוטוקול החיבור דרך sockets הוא פרוטוקול TCP כפי שנלמד בתרגולים / שיעורים.
- נדרש להריץ את צד השרת לפני הרצת צד הלקוח (כדי שמישהו יאזין בצד השני)
- המימוש:
  - החזקה של מערך של struct (נקרא *userDB) שבכל מיקום שלו מכיל מצביע לstruct מסוג:

```
typedef struct USERDB
{
    char name[MAX_USERNAME];
    char password[MAX_PASSWORD];
    int numOfEmails;
    email emails[MAXMAILS];
} userDB;
```

  - בעצם לכל client שקיים במערכת יש שם, סיסמה, מספר אימיילים שיש לו בinbox, ואת מערך הemail ששומר את כל הinbox שלו. מימוש כל email בצורה הבאה באמצעות struct:

```
typedef struct EMAIL
{
    //mail id is the place in the array of the mail
    char from[MAX_CONTENT];
    char to[MAX_CONTENT];
    char subject[MAX_CONTENT];
    char content[MAX_CONTENT];
    bool deleted;
} email;
```

  - באתחול השרת בעצם רץ על קובץ הdatabase ומאתחל userDB לכל אחד מהמשתמשים
- כל שורה שהמשתמש מכניס בכל פונקציונליות נכנסת כשורה נפרדת שלאחריה לוחצים enter.
- פונקציונליות:
  - **QUIT –** בעצם ניתוק של הsocket הנוכחי שהשרת נמצא אתו בתקשורת, מצד הלקוח יש התנתקות מהשרת ולא ניתן יותר להריץ פקודות עד לחיבור מחדש, מצד השרת יש חזרה ללולאת while שמאזינה לחיבור של לקוח חדש. (השרת לא "יוצא / נכבה" לבד)

- אתחול מערך ה*userDB נמצא מחוץ ללולאת הwhile שמחכה כל פעם לclient חדש ולכן אם נשלחו מיילים מסוימים הם שמורים עבור כל client מהdatabase בנפרד אצלו בuserDB.
- **SHOW_INBOX** – השרת רץ על כל המיילים של המשתמש בתוך המערך email שבstruct שלו ובודק האם הemail.deleted == true. בעצם מחיקה של מייל מומשה באמצעות סימון המייל כ"נמחק" – כלומר לא באמת נמחק, יכול לספק הכנה להרחבה עתידית של מציאת מיילים מחוקים. (המימוש הזה אושר לנו במייל מהמתרגל...)
- **GET_MAIL mail_id** – הלקוח שולח דרישה למייל מס' ... – מספר המייל ממומש ע"י מיקומו במערך הemail בתוך הstruct של הuserDB הספציפי שאנחנו מחוברים אליו. ולכן הפונקציה תדפיס את הemail שנמצא במיקום הmail_id במערך.
- **DELETE_MAIL mail_id –** באותה צורה השרת הולך למיקום הemail[mail_id] במערך ומסמן את השדה של הstruct email שם להיות deleted = true
- **COMPOSE :**
  - קורה בשלושה שלבים – כל פעם שהclient מקיש שורה היא נשלחת לשרת שבודק אותה ובסופו של דבר השליחה של כל המידע של הלקוח נשלחת ב3 "מסרים". זאת אומרת:
  - To: Itai – ולחיצה על enter
  - Subject: hello world! – ולחיצה על enter
  - Text: hello Itai! Have a nice day! – ולחיצה על enter
  - הפונקציה בצד השרת רצה על הנתונים ב*userDB ומשווה את שם הנמענים לשמות במערך – אם הם שווים, המייל החדש נשמר אצלם בstruct.
  - השרת ישלח חזרה mail sent ללקוח.
  - הערה: במידה ויש נמען בשורת הto אשר לא נמצא כuser בשרת שלנו אנחנו נתעלם ממנו ופשוט לא נשלח לו את הmail. (גם אם כל הנמענים לא נמצאים)
  - הערה: אם שלחנו to: Oded, Oded, Oded המייל ישלח פעם אחת לOded.
- **פונקציות השליחה / קבלה:**
  - שוות בשני הצדדים.
  - כוללות פונקציית שליחה int sendFullMsg(int sock, char* buf) שעוטפת את הפונקציות:
    - sendLenOfMsg – שליחת אורך ההודעה לפני השליחה של הטקסט עצמו
    - sendAll – שליחת ההודעה עצמה.
  - כנ"ל פונקציית מעטפת recvFullMsg שעוטפת את הפונקציות:
    - recvLenOfMsg – קבלת אורך ההודעה הצפויה להישלח
    - Recvall – קבלת ההודעה עצמה.
  - פונקציות המעטפת הן אלו שנקראות למעשה בשני הצדדים על מנת לוודא את קבלת ההודעה כמו שצריך, הן מחשבות את אורך ההודעה שצריך להישלח ושולחות אותו לפני ההודעה בint כדי שפונקציית הקבלה תדע מה אורך ההודעה שהיא צריכה לקבל.
- הערות נוספות / מקרי קצה:
  - אם הclient מכניס שם משתמש או סיסמה לא נכונים – הוא מתנתק מהשרת.
  - במידה ויש נמען בשורת הto אשר לא נמצא כuser בשרת שלנו אנחנו נתעלם ממנו ופשוט לא נשלח לו את הmail. (גם אם כל הנמענים לא נמצאים)
  - אם שלחנו to: Oded, Oded, Oded המייל ישלח פעם אחת לOded.

- מבנה כללי של "תוכנת" הserver.c להבנה בhigh level:

```
int main(int argc, char** argv)
{

    checking arguments inserted correctly

    parsing userDb from Database and connecting sockets & listening

    while(1) //for each client that connects
    {
        checking user name and password inserted correctly

        "Connected to server"

        receiving user first command

        while(each time receiving another command which is not QUIT)
        {
            command situations that call other functions

            get another command and return to another loop

            if QUIT -> dissconnect user and wait for new client to connect
        }



    }

}
```

- פונקציית הclient.c מבצעת חיבור לשרת, ואז קוראת לפונקציית handleEvents(sock) שבה יש לולאת while שמחכה כל פעם לפקודה חדשה. עד שיוזן QUIT בסיום של פונקציית פקודה אחרת ואז הלקוח יתנתק.

בדיקה נעימה!