

פרויקט סיום -

DHCP, DNS local server, FTP Application server, RUDP

2-4	סקירת מערכת
5.....	פונקציונליות המערכת
6-8	איך מרכיבים
9-11	שאלות תיאורטיות
12-13	דיאגרמת מוצבים
14	מענה על שאלות
15-23	DHCP
24-28	DNS
29-33	TCPIP
34-53	RUDP
54-105	FTP
106-111	Utils
112	FTPExceptions
113	EOF

סקירת מערכת

המטרה עצמה נוגעת בכמה חלקים.

החלק הראשון מתאר את שרת ה DHCP:

שרת זה בנייתו על בסיס פרוטוקול DHCP, ראשי התיבות שלו הן Dynamic Host Configuration Protocol.

זהו פרוטוקול רשות המשמש להקצאה אוטומטית של כתובות IP ופרמטרים אחרים של תצורת רשת, כגון subnet mask, default gateway ושרת DNS לממשירים שונים ברשת.

משתמשים בשרת DHCP בדרך כלל ברשתות גדולות יותר, כגון סביבות ארגוניות כדי לפשט את ניהולו הרשת.

על ידי אוטומציה של הקצאת כתובות IP, שרת DHCP מפחית את הסבירות להתקנחות שיטות ייצרו כאשר אותה כתובות מוקצת לשני ממשירים שונים.

זה גם מקל על מנהלי רשות לעקוב אחר ממשירים שנמצאים ברשות ולפתור בעיות קישוריות.

שרת DHCP פועל על מודל שרת-לקוח, שבו הלקוח שולח בקשה למיידע על תצורת רשות לשרת DHCP אשר מגיב עם המידע המבוקש.

ניתן להשתמש ב-DHCP גם בשילוב עם פרוטוקולי רשות אחרים, כגון DNS ו-NAT.

החלק השני מתאר את שרת ה DNS:

שרת זה בנייתו על בסיס פרוטוקול DNS, ראשי התיבות שלו הן Domain Name System.

זהו פרוטוקול המשמש לתרגוםשמות דומייניים הנינטנים לקריאה אנושית, כגון www.google.com לכתובות קווינטנטן לקריאת מבונה כגון 192.0.1.1.

כאשר אנחנו מקלדים שם דומיין בדף האינטרנט שלנו, הדף שולח בקשה לשרת DNS לפענוח את שם הדומיין לכתובת קו.

שרת DNS לאחר מכן מגיב עם כתובות הקווינטנטן המתאימה כדי שהדף יוכל להשתמש כדי להתחבר לשרת האינטרנט הנמאחסן את האתר האינטרנט הרצוי.

DNS חיוני לתפקוד האינטרנט, מכיוון שהוא מאפשר למשתמשים לגשת לאתר אינטרנט ולמשאים מקוונים אחרים באמצעות שמות דומיין שקל לזכור, במקום לשנן כתובות IP מורכבות.

זה גם מאפשר שימוש בדואר אלקטרוני, הודעות מיידיות ושירותי רשות אחרים המסתמיכים על שמות דומיין כדי להזמין את השירות של הנמען.

DNS פועל על מבנה היררכי, עם רמות רבות של שמות דומיין, כגון דומיינים ברמה העליונה(TLDs – Top Level Domains) כמו .com, .org ועוד,

ותחת דומיינים כמו support בכתובת <https://support.wix.com> לדוגמה.

מבנה זה מאפשר ניהול יעיל וניתן להרחבת שמות דומיין ברחבי האינטרנט.

החלק השלישי מתאר את RUDP:

.Reliable User Datagram Protocol – RUDP

זהו פרוטוקול شبבת transport המספק העברת נתונים אמינים בין ישומים דרך רשת זו כמו האינטרנט.

RUDP נועד כדי לספק את המהימנות של TCP בשילוב עם הפשטות והיעילות של UDP.

כדי להבטיח איזoct, RUDP מרחיב את UDP בכך שהוא מוסיף:

1. אישור על פאקטות שהתקבלו(Ack)

2. windowing – הגדלה והקטנה של החלון בהתאם

3. flow control – הגבלה של קצב ובמota שליחת הפאקטות שנעשית על ידי השולח אל עבר המקלט כדי להימנע מעומס אצל המקלט.

4. Retransmission – שליחה מחדש של פאקטות שאבדו.

5. congestion control – השולח מסוגל את קצב שליחת הפאקטות בעקבות עומס על הרשת במגוון טכניקות שונות.

החלק הרביעי מתאר את שירות הFTP:

שירות האפליקציה בניו על בסיס פרוטוקול FTP – File Transfer Protocol –

זהו פרוטוקול סטנדרטי המשמש להעברת קבצים דרך הרשת.

שירות FTP מאפשר ללקוחות שמתחררים אליו להעלות אליו קבצים, להוריד מהם קבצים, לאפשר גישה לקבצים שנמצאים על השירות וכו'.

חלק השירותי FTP תומכים בקבלת חיבורים רבים בו זמנית, בסוגים שונים של העברת קבצים(אסקוי\ビナリ), ובהמשך העברת המידע לאחר תקלות במהלך ההעברה.

שירות FTP יכול לזרע על מגוון מערכות הפעלה כמו וינדוס, לינוקס ועוד.

פונקציונליות המערכת

DNS: קבלת כתובת IP של דומיין לפי בקשה המשתמש.

DHCP: הצעת כתובת IP בזמן מסויים לתחנה ברשות.

RUDP: פרוטוקול אמין מבוסס UDP עם שילוב של CC ו-FCI.

FTP

שרת האפליקציה שלנו קיבל חיבורים מלוקחות שונים.

כל לקוח יוכל להשתמש בפקודות הבאות אל מול השרת:

,XPWD ,CWD ,XCWD ,LIST ,NLST ,PORT ,EPRT ,PASS ,USER ,AUTH ,OPTS
,RMD ,XRMD ,MKD ,XMKD ,DELE ,CDUP ,XCUP ,SYST ,PASV ,TYPE ,PWD
.QUIT ,HELP ,APPE ,STOR ,RETR ,REST ,RNTO ,RNFR

כל קובץ תיוקה שיוצג על ידי השרת יוכל בין היתר את השינוי האחרון שנעשה בקובץ, את גודל הקובץ, את שם הקובץafi שנדרש בהוראות הפרויקט ועוד.

כמו כן, תיכלל האופציה להעביר קובץ כולל הפסקה באמצעות בנדראש.

AIR MERITS

המערכת רצה על מערכות הפעלה כמו ווינדוס ולינוקס.

:DNS

שלבי הריצה: פותחים את התקייה שעלייה נמצא הפרויקט, פותחים את הטרמינל וכותבים את הפקודה הבאה " sudo python3 dnsserver.py ", משתמשים ב sudo כדי לתת הרשות מנהל בין שמות עסקים עם פורטים שדרושים הרשות מנהל (53) כדי להתעסק איתם.

:DHCP

שלבי הריצה: פותחים את התקiya שעלייה נמצא הפרויקט, פותחים את הטרמינל וכותבים את הפקודה הבאה " sudo python3 dhcpc.py " משתמשים ב sudo כדי לתת הרשות מנהל בין שמות עסקים עם פורטים שדרושים הרשות מנהל (67-68) כדי להתעסק איתם.

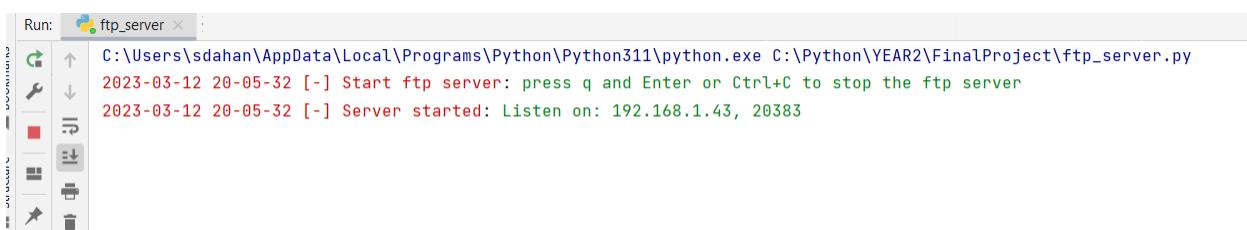
:FTP

שלבי הריצה:

:Windows

פותחים את cmd ובמקביל פותחים את pycharm או סביבת עבודה אחרת ומראים את הקובץ `ftp_server`.

דוגמה:



```
C:\Users\sdahan\AppData\Local\Programs\Python\Python311\python.exe C:\Python\YEAR2\FinalProject\ftp_server.py
2023-03-12 20:05:32 [-] Start ftp server: press q and Enter or Ctrl+C to stop the ftp server
2023-03-12 20:05:32 [-] Server started: Listen on: 192.168.1.43, 20383
```

בcmd כותבים `ftp`, אחר כך כותבים `open` ולאחר מכן את כתובת הkn והפורט שהשרות מאשר עליהם. (כתב בתרמונה לעיל).

כותבים את שם המשתמש הבא : `user` והסיסמה : `1234`

```
C:\Users\sdahan>ftp
ftp> open 192.168.1.43 20383
Connected to 192.168.1.43.
220 Welcome.
202 UTF8 mode is always enabled. No need to send this command
User (192.168.1.43:(none)): user
331 Please, specify the password.
Password:
230 Login successful.
ftp>
```

לאחר מכן משחקים עם זה כמו שרת ftp רגיל.
ניתן לדוגמא לכתוב dir ולקבל את כל הספריות בנתיב הנוכחי.

לדוגמה:

```
Command Prompt - ftp
ftp> open 192.168.1.43 20383
Connected to 192.168.1.43.
220 Welcome.
202 UTF8 mode is always enabled. No need to send this command
User (192.168.1.43:(none)): user
331 Please, specify the password.
Password:
230 Login successful.
ftp> dir
200 PORT command successful.
150 Starting data transfer.
drwxrwxrwx 1 0 0 0 Sep 16 06:33 $Recycle.Bin
drwxrwxrwx 1 0 0 0 Feb 15 13:40 $WinREAgent
-rwxrwxrwx 1 0 0 30 Mar 10 13:11 AUTOEXEC.BAT
drwxrwxrwx 1 0 0 0 Sep 27 10:05 Brother
drwxrwxrwx 1 0 0 4096 May 05 12:11 computer science
drwxrwxrwx 0 0 0 0 Feb 23 17:01 Config.Msi
drwxrwxrwx 1 0 0 0 Sep 16 06:21 dell
dr-xr-xr-x 1 0 0 4096 Oct 11 06:02 Documents and Settings
drwxrwxrwx 1 0 0 0 Sep 15 22:12 Drivers
-rw-rw-rw- 0 0 0 8192 Mar 09 13:39 DumpStack.log.tmp
drwxrwxrwx 1 0 0 0 Mar 16 13:59 emacs-25.1-x86_64-w64-mingw32
drwxrwxrwx 1 0 0 0 Mar 10 13:12 Flexlm
drwxrwxrwx 1 0 0 0 Aug 19 12:09 GoogleCloud
drwxrwxrwx 1 0 0 24576 Nov 09 16:56 HDLTurboWriter
-rw-rw-rw- 0 0 0 12784398336 Mar 12 06:42 hiberfil.sys
drwxrwxrwx 1 0 0 0 Sep 15 22:12 Hotfix
drwxrwxrwx 1 0 0 0 Sep 16 00:00 inetpub
drwxrwxrwx 1 0 0 0 Sep 15 14:09 Intel
dr-xr-xr-x 0 0 0 0 Sep 16 06:46 MSOCache
```

לינוקס:

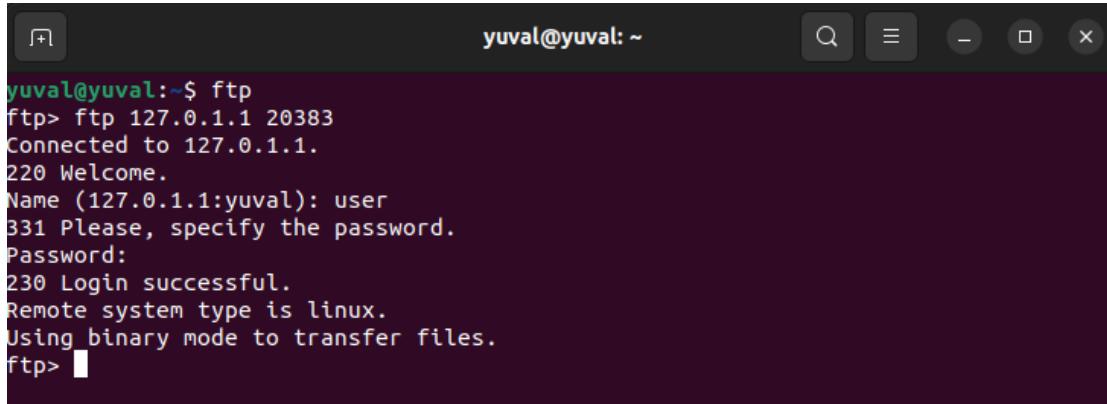
לדוגמה:

פותחים את הטרמינל ובמקביל פותחים את pycharm או סביבת עבודה אחרת
ומריצים את הקובץ .ftp_server.py.

```
/home/yuval/PycharmProjects/YEAR2/venv/bin/python /home/yuval/PycharmProjects/YEAR2/ReshatotTikshoret/ftp_server.py
2023-03-12 20:08:15 [-] Start ftp server: press q and Enter or Ctrl+C to stop the ftp server
2023-03-12 20:08:15 [-] Server started: Listen on: 127.0.0.1, 20383
2023-03-12 20:09:08 [-] Accept: New client connected 127.0.0.1, 44022
2023-03-12 20:09:08 [-] Client connected 127.0.0.1, 44022
```

בטרמינל כותבים `ftp`, אחר בר בותבים `ftp` ולאחר מכן מכן את כתובת הקו' והפורט שהשרות מאשר עליהם.(כתוב בתמונה לעיל).

כותבים את שם המשתמש הבא : `user` והסיסמה : `1234`.



```
yuval@yuval:~$ ftp
ftp> ftp 127.0.1.1 20383
Connected to 127.0.1.1.
220 Welcome.
Name (127.0.1.1:yuval): user
331 Please, specify the password.
Password:
230 Login successful.
Remote system type is linux.
Using binary mode to transfer files.
ftp>
```

: CLIENT

שלבי הרצה:

פותחים את הטרמינל בנתיב הנוכחי של הקובץ `ftp_client.py`,

כותבים `python3 ftp_client.py` , במקביל פותחים את הטרמינל בנתיב הנוכחי של הקובץ `ftp`, כותבים `python3 ftp_server.py` ומתחילה תקשורת בין השירות ללקוח.

כותבים בצד הלקוח `open`.

לאחר מכן כותבים את הקו' של השירות, לאחר מכן את הפורט של השירות, לאחר מכן את `user` בשם משתמש, לאחר מכן `1234` בסיסמא והחיבור מצטלח.

הוסףנו סרטון על בר.

שאלות תיאורטיות

שאלה 1 - ההבדלים בין פרוטוקול TCP לפורוטוקול QUIC הם :

1. תהליך פתיחת הקשר ולחיצת הידיים בQUIC הוא הרבה יותר קצר מאשר בTCP כיון שבזינוק באשר לקוח פותח חיבור חבילתי התגובה כוללת את הנתונים של פרוטוקולי האבטחה ומפתחות ההגדירה, ובכך קיצרנו את תהליך פתיחת הקשר כיון שבTCP קבלת פרוטוקולי האבטחה ומפתחות ההגדירה מתרבצת בעוד שלב וכעוז בקשה שנשלח, ובQUIC זה קורה במקרה מהבקשתה הראשונה.
2. בQUIC יש לנו כמה זרים שבהם עובר המידע. לכן, אם מתרחשת שגיאה בזרם אחד, זרים אחרים יכולים להמשיך לפעול מוביל להיות מושפעים לעומת TCP שבו המידע עובר בזרם אחד.
3. בQUIC יש לנו ID Connection שזהו מזהה חיבור, המאפשר לנקודות קצה בעלות IP או פורט שונים להיות מחוברים באותו חיבור.
דוגמא לכך היא האם הנקודת קצה שלנו היא פלאפון נייד ואנחנו עוברים מחיבור WIFI לחיבור סלולרי, החיבור עם השרת בפרוטוקול QUIC נשמר לנו ולא נסגר ברגע ששגרנו את WIFI, מה שקרה בTCP.
4. QUIC הוא פרוטוקול מוצפן וכיום TCP יהיה מוצפן יש לשימוש בפרוטוקולים נוספים על מנת להוסיף רובד אבטחה לדוגמא TLS.

שאלה 2 - ההבדלים בין CUBIC ל VEGAS הם:

1. בפרוטוקול CUBIC יש סוג של חלון שגדל וקטן בהתאם למצב של הרשת, ולפי גודל החלון נשלחות הפקטות. הוא מטפל בגודש רק לאחר שהוא מזהה את הגודש ובכך מקטין את גודל החלון.

ואילו פרוטוקול VEGAS מזהה את הגודש לפני בכר שהוא מודד את זמן ה הגיע של ה-ACK מרגע שליחת הפקטה ובכך מזהה בזמן אמת מה המצב של הרשת, וקובע את קצב שליחת הפקטות.

2. הטיפול של VEGAS בעת גודש מתבצע בכר שהוא מתאים את עצמו למצב הרשת בכלל רגע נתון لكن השינויים בו הם יחסית קיצוניים כיון שהוא יכול לשלוח מספר רב של פاكتות, ולאחר מכן אם הוא רואה שיש עומס לשילוח מספר קטן בהרבה של פاكتות.

הטיפול של CUBIC בעת גודש מתבצע באמצעות הקטנת החלון בחצי ואז שימוש בעיקרון slow start אשר מגדיל את גודל החלון ב-1 בכל פעם ולבן זה הופך את CUBIC לטיפול בגודש באופן פחות קיצוני.

שאלה 3 - פרוטוקול BGP הוא פרוטוקול ניטוב של המידע שעובר ברשת בכר שהוא מנהל טבלאות של הרשותות שמחוברות אליו והקשרים ביניהם לבין הרשותות אחרות.

אוף בחירת דרך איזה רשותות לעבר הוא בשליטת מנהל הרשות שבוחר כל מיני הגדרות שבאמצעותם הנטב יבחר רשות מסויימת אליה יעבור המידע.

OSPF הוא פרוטוקול ניטוב היררכי, הוא עובד עם אלגוריתם דיקסטרה למציאת המסלול הקצר ביותר. עבורו קצר שווה לעלות העברת המידע על גבי רוחב הפס בין הנטבים.

השוני בין שני הפרוטוקולים הוא ש OSPF בוחר מסלול קצר לפי רוחב הפס ואילו BGP בוחר את המסלול הקצר ביותר שעובר בכמה שפחוות AS (AS זה קובוצה של מספר נטבים באיזור מסוים).

לכן OSPF וגם BGP עובדים לפי מסלולים קצריים אך לכל אחד הגדרה שונה למסלול קצר.

שאלה 4

Application	Port Src	Port Des	IP Src	IP Des	Mac Src	Mac Des
FTP	20383	30084	12.3.20.3	100.100.100.(3-10)	-	-

אם נעבד עם NAT :

בעבודה עם NAT לכל תחנה ברשות יש כתובות IP פנימית באוטה הרשת וכל הנקודות קצה יש כתובות IP חיצונית שדרוכה הם יוצאים החוצה בעזרת הרouter.

לכן אם נעבד עם NAT הפורט מקור ישנה לפורט של הרouter והכתובת יעד של המקור תשנה מהכתובת IP של הנקודות קצה לכתובת IP החיצונית של הרouter ובכך יהיה שיוני בכתובת IP.

בחזרה של פאקטה קורה התהילך הפוך שהפקאה מגיעה לרouter, היא מmirה חזרה את הפורט יעד וככתובת היעד של הפקאה בחזרה למי שביצע

את הבקשה ובכך מנוט את הפקאה למקום הנכון.

בנוסף NAT לא משתמש בכתובות mac لكن הם לא רלוונטיות בשביול.

אם נעבד עם QUIC :

בעבודה עם QUIC כל נקודת קצה פותחת קשר ישיר עם השרת בלבד הסתמכות על Router אחד שדרכו כל הנקודות קצה שמחוברות באוטה רשת יוצאים רק דרך הרouter זהה, לכן כתובות IP והפורטם לא ישנים בשניibus עם QUIC.

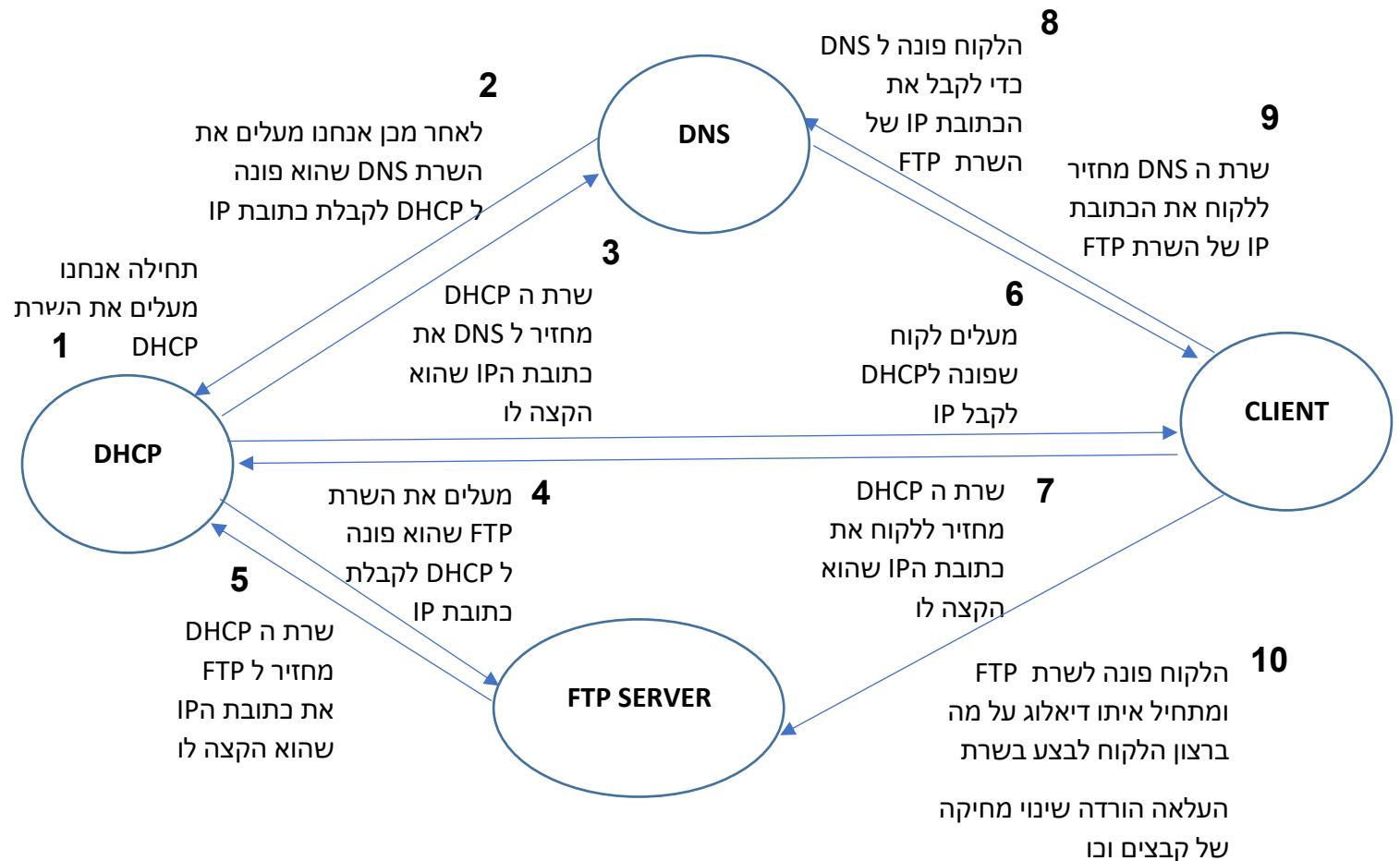
בנוסף QUIC לא משתמש בכתובות mac لكن הם לא רלוונטיות בשביול.

שאלה 5 - ההבדלים בין ARP ל DNS הם :

- ARP משמש לאיתור כתובת MAC של תחנה ברשות לפי IP שלה באוטה הרשת, בעוד שDNS משמש לאיתור כתובת IP של דומיין מסוים ברשת.
- ARP פועל באוטה הרשת בעוד DNS פועל על כל רשת האינטרנט.

דיאגרמת מלבנים

דיאגרמת מלבנים לאופן הפעולה של המערכת:



תיאור אופן הפעולה של המערכת:

תחילה נעה את שרת DHCP כיון שהוא השלב הראשון והוא מחלק בתובות IP לכל יתר הרכיבים במערכת.

בשלב השני נעה את שרת DNS כדי שהוא יפנה ל DHCP על מנת לקבל בתובות IP בשביולו.

בשלב הבא נעה את שרת אפליקציית ה FTP שהוא תחילת יפנה ל DHCP על מנת לקבל בתובות IP בשביולו.

בשלב האחרון נעה את הלוקו שהוא יבקש מה DHCP בתובת IP. לאחר מכן הוא יפנה לשרת DNS כדי לקבל בתובת IP של שירות FTP שלו הוא רוצה להתחבר. לאחר שהлокו קיבל את בתובת IP של שירות FTP, הוא מתחבר לשרת ומתחיל לבצע פקודות על השירות כגון העלה, הורדה, שינוי שם, מחיקה של קבצים או תיקיות וכו'.

מענה על שאלות

המערכת מתגברת על איבוד חבילות בר :

בחלק ממימוש UDP הוספנו פרוטוקול אמינות בבר שלל כל חבילה שהגיעה לצד המקלט הוא מוחזר בחזרה ack לשולח, ובבר אנחנו מצביעים בקרה על החבילות שלחנו.

שלא קיבל ack על חבילה שללו נדע שהחביבה הלבा לאיבוד ובבר נדע לשולח אותה שוב, לבן נשלח את הפאקטאות החל מהבית האחרון שידוע לנו שהגיעו למקבל וכמה נמשיך עד שכול המידע הגיע לצד השני כולל הפאקטאות שהלבו לאיבוד.

המערכת מתגברת על בעיות latency(השהייה) בר :

בכל פעם שאחננו שלוחים חבילה, אנחנו כוללים מידע על המספר הסידורי של הפקאה ששלחנו.

בצד המקלט, אנחנו עוקבים אחר המספר הסידורי של הפאקטאות שקיבלו ברכף מהשלוח.

כיוון שאחננו שלוחים פאקטאות בגודל קבוע (1024 ביטים) אז אנחנו תמיד יודעים עד איזה בית של מידע קיבלנו בזכות המספר הסידורי של הפאקטה.

לבן תמיד נסיף פאקטה למקום המתאים לה ברשימה ובבר ניצור רצף של ביטים עד שנקבל את כל המידע מהשלוח.

DHCP

שרת DHCP זה שרת שמקצה בתובות IP לנקודת קצה על ידי תהליך שכולל 4 שלבים :

1. נקודת קצה שולחת הודעת discovery – לגלות מי הוא השירות DHCP.
2. השירות DHCP מוחזיר הודעת offer שמחזיר לנקודת קצה הודעה שהוא השירות ומצביע לו בתובת IP מסוימת.
3. נקודת הקצה שולחת הודעת request ש商量קשת את בתובת IP הזאת ועוד הגדרות DHCP מהשרת.
4. השירות DHCP מוחזיר הודעת ack שהוא מאשר שהקצה לו את בתובת IP הזאת ומוחזיר לו עוד הגדרות DHCP כמו בתובת IP של רואטר, שירות DNS, subnet mask .

לבניית השירות DHCP השתמשנו בספרייה `scapy` של פיתון.

```
ipdictionary = {2: 'free', 3: 'free', 4: 'free', 5: 'free', 6: 'free', 7: 'free', 8: 'free', 9: 'free', 10: 'free'}
```

```
currentip = None
dnsip = None
dnsrequest = 0
```

תחילה הגדרנו משתנים גלובליים שהם dictionary של סימונות בתובת IP, משתנה שמחזיק את בתובת IP הנוכחית שהוקצתה, בתובת IP של השירות DNS, ומשתנה שאומר אם השירות DNS פנה כדי לקבל בתובת IP.

```
def getIpFromPool():
    for ip in ipdictionary:
        if ipdictionary[ip] == "free":
            ipdictionary[ip] = "occupied"
            return "100.100.100."+str(ip)

if __name__ == '__main__':
    sniff(filter="udp and (port 67 or 68)", prn=dhcpserver, iface="lo")
```

הגדרכנו פונקציה בשם `getIpFromPool` שהיא עוברת על פול היסודות שהגדרכנו ובודקת אם יש סימות לא בשימוש, מסמנת שכעת הכתובת בשימוש ומחייבת את הכתובת IP שהתחילה שלה היא 100.100.100. והחלק האחרון זה היסות מהפול. פונקציית `main` שהוא מסניפה לפי סינון של UDP פורט 67 או 68(שאלה פורטים של DHCP) – ברגע שהוא תפסה פאקטה היא שולחת אותה לפונקציית `.dhcpserver`

```
def dhcpserver(packet):
    if DHCP in packet and packet[DHCP].options[0][1] == 1:
        clientmac = packet[Ether].src
        global currentip
        if not currentip:
            currentip = getIpFromPool()
        global dnsrequest
        global dnsip
        if dnsrequest == 0:
            dnsrequest = 1
            dnsip = currentip

        offerpacket = Ether(dst=clientmac) / \
            IP(src="100.100.100.254", dst="255.255.255.255") / \
            UDP(sport=67, dport=68) / \
            BOOTP(op=2, yiaddr=currentip, siaddr="100.100.100.254", chaddr=clientmac) / \
            DHCP(options=[("message-type", "offer"), ("server_id", "100.100.100.254"), ("name_server", dnsip),
                          ("subnet_mask", "255.255.255.0"), ("router", "100.100.100.1"), ("lease_time", 86400), "end"])

        time.sleep(0.7)
        sendp(offerpacket)
        return
```

פונקציית `dhcpserver`

תחילה נודע שקיים פאקטה DHCP וגם שהסוג שלה הוא (1), נחל ממנה את הכתובת mac של נקודת הקצה ונתקaza כתובות IP מסוימת ונבדוק אם זה הבקשה הראשונה שהDNS קיבל לפי הערך שקיים בdnsrequest. אם זה הבקשה הראשונה סימן שהוא של DNS ש牒בוקש IP (הגדרכנו שהDNS הוא הראשון שעולה לפי האפליקציה ולפחות) לכן נשנה את המשתנה dnsrequest ונדע שהקצנו IP DNS, שכן נשים במשתנה dnsip את הIP שהקצנו.

לאחר מכן נבנה פאקטה להחזרה, בשדה IP של המקור נשים את הIP של הרשת DHCP וביעד 255.255.255.255 בין שאנו לא יודעים עוד למי להחזיר ביעון שכן לו עוד IP וכן שדר את הפאקטה לבולם, נגדיר את הפורטים ל 67 ו 68 .

בחלק של bootp בכתובת IP לבקשת גדרת נציגו את ה IP של DHCP ובכתובת mac את כתובת mac שהילצנו מהבקשתה. ולבסוף נוסיף עוד הגדרות של dhcp, נציין שזאת הודעת offer ונוסיף את ה IP של ה IP של DNS, subnet mask, DHCP, IP רואוטר, וככמה זמן נקצתו לו את כתובת IP. ונשלח את הפאקטה חוזרת.

```
elif DHCP in packet and packet[DHCP].options[0][1] == 3:
    clientmac = packet[Ether].src
    ackpacket = Ether(dst=clientmac)/ \
        IP(src="100.100.100.254", dst="255.255.255.255")/ \
        UDP(sport=67, dport=68)/ \
        BOOTP(op=2, yiaddr=currentip, siaddr="100.100.100.254", chaddr=clientmac)/ \
        DHCP(options=[("message-type", "ack"), ("server_id", "100.100.100.254"), ("name_server", dnsip),
                      ("subnet_mask", "255.255.255.0"), ("router", "100.100.100.1"), ("lease_time", 86400), "end"])
    currentip = None
    sendp(ackpacket)
```

אם הפאקטה היא מסוג request (3), נחלץ את כתובת mac מהבקשתה ונבנה פאקטה חדשה כדי להחזיר באופן הבא: בשדה IP של המקור נשים את ה IP של השירות DHCP וביעד 255.255.255.255 ביזון שאנחנו לא יודעים עוד למי להחזיר כי אין לו עוד IP ולבן נשדר את הפאקטה לבולם, ונגידיר את הפורטים ל 67 ו 68. בחלק של bootp בכתובת IP לבקשת גדרת נציגו את ה IP שרת, נגידיר את ה IP של DHCP ובכתובת mac את כתובת mac שהילצנו מהבקשתה. ולבסוף נוסיף עוד הגדרות של dhcp, נציין שזאת הודעת offer, ונוסיף את ה IP של DNS, subnet mask, IP רואוטר, וככמה זמן נקצתו לו את כתובת IP.

בנוסף בינו clientdhcp שדרכו ביקש בתשובות IP לנקודות קצה. תחילת הגדרנו שהכתובת mac של המבקש היא 00:00:00:00:00:00 ומשתנה שיחזיק את ה IP שהקצו לנו ואת ה IP של ה DNS.

```

def getip():
    discoverpacket = Ether(dst="ff:ff:ff:ff:ff:ff", src=clientmac) / \
                     IP(src="0.0.0.0", dst="255.255.255.255") / \
                     UDP(sport=68, dport=67) / \
                     BOOTP(op=1, chaddr=clientmac) / \
                     DHCP(options=[("message-type", "discover"), "end"])

    srp1(discoverpacket, timeout=0.5, iface="lo", verbose=False)
    sniffforoffer()
    return ips
if __name__ == '__main__':
    getip()

```

הגדכנו פונקציית `main` שהיא מרים את פונקציית `getip`.

פונקציית `getip` בונה פאקטת `discovery` באופן הבא: בשדה `Ether` נגדיר את הכתובת `mac` של היעד והמקור, בשדה `IP` של המקור נשים את `0.0.0.0` בין שאי לנו עוד `IP` וביעד `255.255.255.255` בין שאנו לא יודעים עוד למי לשלוח (לא יודעים מ `IP` של שירות `DHCP`), נגדיר את ה포רטים ל `67` ו `68`.

בחלק של `bootp` נגדיר את הכתובת `mac` של נקודת הקצה (`00:00:00:00:00:00`).
ולבסוף נגדיר שזו בקשה `DHCP` מסוג `discover`.

נשלח את הפקאה במכשיר `01` (מחשב עם עצמו) ונקרה לפונקציית `sniffforoffer`.

```

def sniffforoffer():
    sniff(filter="udp and (port 67 or 68)", stop_filter=lambda p: True, prn=request)

```

בפונקציית `sniffforoffer` נסניף פאקטות `UDP` ב포רט `67` ו `68` ונשלח אותם לפונקציית `request`.

```

clientmac = "00:00:00:00:00:00"
ips = None

def request(packet):
    if DHCP in packet and packet[DHCP].options[0][1] == 2:
        requestpacket = Ether(dst="ff:ff:ff:ff:ff:ff") / \
            IP(src="0.0.0.0", dst="255.255.255.255") / \
            UDP(sport=68, dport=67) / \
            BOOTP(op=1, chaddr=clientmac) / \
            DHCP(options=[("message-type", "request"), ("requested_addr", packet[BOOTP].yiaddr), ("server_id", packet[BOOTP].siaddr), "end"])

    srp1(requestpacket, timeout=0.5, iface="lo", verbose=False)
    sniffforack()
return

```

בפונקציית `request` נזודא שהסנפנו פאקטה DHCP והוא מסוג `offer` (2).

לאחר מכן נבנה פאקטה `request` באופן הבא: בשדה `Ether` נגדיר את הכתובת `mac` של היעד והמקור, בשדה `IP` של המקור נשים את `0.0.0.0` כיון שאין לנו עוד `IP` וביעד `255.255.255.255` כיון שאנו לא יודעים עוד למי לשלוח (לא יודעים מ `IP` של שרת DHCP), נגדיר את ה포רטים ל `67` ו `68`.

בחילוק של `bootp` נגדיר את הכתובת `mac` של הנקודת קצה (`00:00:00:00:00:00`).

ולבסוף נגדיר זו בקשה DHCP מסוג `request`, בשדה `requested_addr` נגדיר את `IP` שהTCP DHCP הצביע לנו גם נגדיר את `IP` של השרת DHCP.

נשלח את הפקאהו במשם OS (מחשב עם עצמו) ונראה לפונקציית `sniffforack`.

```

def sniffforack():
    sniff(filter="udp and (port 67 or 68)", stop_filter=lambda p: True, prn=ack)

```

בפונקציית `sniffforack` נסניף פאקטות UDP בפורט `67` ו `68` (DHCP) ונסלח אותן לפונקציית `ack`.

```

def ack(packet):
    if DHCP in packet and packet[DHCP].options[0][1] == 5:
        global ips
        ips = (packet[BOOTP].yiaddr, packet[DHCP].options[2][1])
        return
    else:
        sniffforack()

```

.(5) ack נודע שהסנפנו פאקטה DHCP והוא מסגד בפונקציית ack.

נחלץ מהפאקטה את הכתובת IP שהTCP/IP הקרה לנו, נשמר אותה במשתנה הגלובלי `IPS` ביחד עם הכתובת IP של הDNS שגם אותה נחלץ מהפאקטה.

לבסוף אחרי שקיבלו את IP נחזיר בחזרה לפונקציה ip get ונחזיר ממנה את המשתנה IPS לנוקודת קצה שפנתה כדי לקבל IP.

אם היא לא מסוג ack נחזיר לה סנייף פאקטות.

הרצה של DHCP

```
roy@roy-VirtualBox: ~/PycharmProjects/FinalProject
```

Sent 1 packets.
.Sent 1 packets.

```
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 clientdhcp.py  
('100.100.100.2', '100.100.100.2')  
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 clientdhcp.py  
('100.100.100.3', '100.100.100.2')  
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 clientdhcp.py  
('100.100.100.4', '100.100.100.2')  
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ 
```

בצד שמאל ניתן לראות את השרטת שהוא מקבל מספר בקשות ון מנוקדות קצרה.

בצד ימין ניתן לראות 3 נקודות קצרה ש商量קשות IP – תחילת אנחנו מגדירים שהDNS יבקש ראשון IP וכן הוא מקבל tuple שמכיל את הIP שהקצנו לו ואת IP של ה-NS (אותו IP).

לאחר מכן עוד 2 נקודות קצרה שמקבלות IP – בצד שמאל הIP שהקצנו להם ובצד ימין הIP של השרת DNS.

תמונות dhcp.pcap נלקחו מההקלטה wireshark

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
2	0.727521875	100.100.100.254	255.255.255.255	DHCP	318	DHCP Offer - Transaction ID 0x0
3	0.754694763	0.0.0.0	255.255.255.255	DHCP	300	DHCP Request - Transaction ID 0x0
4	1.471094106	100.100.100.254	255.255.255.255	DHCP	318	DHCP Offer - Transaction ID 0x0
5	1.520076382	100.100.100.254	255.255.255.255	DHCP	318	DHCP ACK - Transaction ID 0x0
6	1.576283317	100.100.100.254	255.255.255.255	DHCP	318	DHCP ACK - Transaction ID 0x0
7	4.500265081	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
8	5.219113150	100.100.100.254	255.255.255.255	DHCP	318	DHCP Offer - Transaction ID 0x0
9	5.250590147	0.0.0.0	255.255.255.255	DHCP	300	DHCP Request - Transaction ID 0x0
10	5.975993741	100.100.100.254	255.255.255.255	DHCP	318	DHCP Offer - Transaction ID 0x0
11	6.012408195	100.100.100.254	255.255.255.255	DHCP	318	DHCP ACK - Transaction ID 0x0
12	6.063067717	100.100.100.254	255.255.255.255	DHCP	318	DHCP ACK - Transaction ID 0x0
13	8.957248218	0.0.0.0	255.255.255.255	DHCP	288	DHCP Discover - Transaction ID 0x0
14	9.679640024	100.100.100.254	255.255.255.255	DHCP	318	DHCP Offer - Transaction ID 0x0
15	9.727143511	0.0.0.0	255.255.255.255	DHCP	300	DHCP Request - Transaction ID 0x0
16	10.430889475	100.100.100.254	255.255.255.255	DHCP	318	DHCP Offer - Transaction ID 0x0
17	10.479163653	100.100.100.254	255.255.255.255	DHCP	318	DHCP ACK - Transaction ID 0x0
18	10.511171642	100.100.100.254	255.255.255.255	DHCP	318	DHCP ACK - Transaction ID 0x0

ניתן לראות בווירשאrk את התהליך מול השרת DHCP .

בחלק הכהול רואים את הבקשה הראשונה של השרת DNS – תחיליה יש הודעה request offer ואז ack ולבסוף discover

בחלק האדום ניתן לראות את נקודת הקצה השנייה המבקשת IP ובחלק בירוק את הנקודת השלישייה המבקשת IP.

Wireshark · Packet 1 · new dhcp.pcapng

```

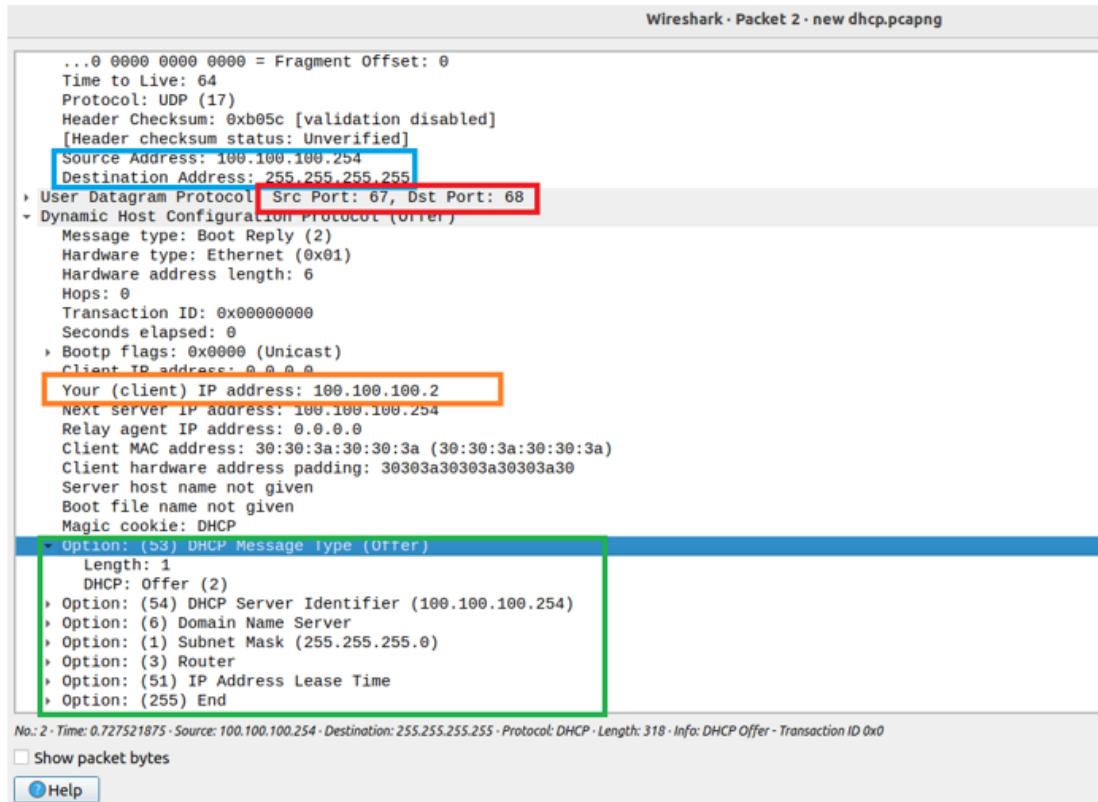
.... 0101 = Header Length: 20 bytes (5)
> Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 272
Identification: 0x0001 (1)
> 000. .... = Flags: 0x0
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: UDP (17)
Header Checksum: 0x79dd [validation disabled]
[Header checksum status: Unverified]
Source Address: 0.0.0.0
Destination Address: 255.255.255.255
> User Datagram Protocol, Src Port: 68, Dst Port: 67
- Dynamic Host Configuration Protocol (Discover)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x00000000
  Seconds elapsed: 0
  > Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 0.0.0.0
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: 30:30:3a:30:30:3a (30:30:3a:30:30:3a)
  Client hardware address padding: 30303a30303a30303a30
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
- Option: (53) DHCP Message Type (Discover)
  Length: 1
  DHCP: Discover (1)
  > OPTION: (255) End

```

No.: 1 · Time: 0.000000000 · Source: 0.0.0.0 · Destination: 255.255.255.255 · Protocol: DHCP · Length: 288 · Info: DHCP Discover - Transaction ID 0x0

Show packet bytes

בתמונה אפשר לראות את הפאקטה DHCP discover, בחלק הבהיר את הIP יעד
ומקור 0.0.0.0 כי אין לנו עוד IP ו- 255.255.255.255 כדי לשדר ברודקאסט. בחלק
האדום את ה포רטים שעובדים אותם בtcp פורטים 67 ו 68, ובירוק את סוג
.discover שהיא



בתמונה אפשר לראות את הפאקטה DHCP offer , בחלק הבהיר את הIP
מקור 100.100.100.254, הIP של השרת DHCP ויעד 255.255.255.255 כדי לשדר
ברודקאסט. בחלק האדום את ה포רטים שעובדים אותם בtcp פורטים 67 ו 68,
בירוק את סוג הבקשה שהיא offer ועוד הגדרות DHCP כמו subnet mask , רוטר
, IP של DNS, ולכמה זמן אני ה DHCP מקצה לי את הIP .
ובנוסף בחלק הבהיר ניתן לראות את הIP שהשרת DHCP הציע.

Wireshark - Packet 3 - new dhcp.pcapng

```

Identification: 0x0001 (1)
  000. .... = Flags: 0x0
  ...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: UDP (17)
Header Checksum: 0x79d1 [validation disabled]
[Header checksum status: Unverified]
Source Address: 0.0.0.0
Destination Address: 255.255.255.255
User Datagram Protocol, Src Port: 68, Dst Port: 67
Dynamic Host Configuration Protocol (Request)
  Message type: Boot Request (1)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x00000000
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0
  Your (client) IP address: 0.0.0.0
  Next server IP address: 0.0.0.0
  Relay agent IP address: 0.0.0.0
  Client MAC address: 30:30:3a:30:30:3a (30:30:3a:30:30:3a)
  Client hardware address padding: 30303a30303a30303a30
  Server host name: not given
  Boot file name not given
  Magic cookie: DHCP
  Option: (53) DHCP Message Type (Request)
    Length: 1
    DHCP: Request (3)
  Option: (50) Requested IP Address (100.100.100.2)
  Option: (54) DHCP Server Identifier (100.100.100.254)
  Option: (255) End
    Option End: 255
No.: 3 Time: 0.754694763 - Source: 0.0.0.0 - Destination: 255.255.255.255 - Protocol: DHCP - Length: 300 - Info: DHCP Request - Transaction ID 0x0
 Show packet bytes
 Help

```

בתמונה הבאה אפשר לראות את הפאקטה DHCP request, בחלק הבהיר את הIP
יעד ומוקור 0.0.0.0 כי אין לנו עוד IP או 255.255.255.255 כדי לשדר ברודקאסט,
בחלק האדום את ה포רטים שעובדים אותם בdhcp פורטים 67 ו 68, בירוק את סוג
הבקשה שהיא request, הכתובת שהציגו לנו ואת הכתובת של השרת DHCP.

Wireshark - Packet 5 - new dhcp.pcapng

```

...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: UDP (17)
Header Checksum: 0xb95c [validation disabled]
[Header checksum status: Unverified]
Source Address: 100.100.100.254
Destination Address: 255.255.255.255
User Datagram Protocol, Src Port: 67, Dst Port: 68
Dynamic Host Configuration Protocol (ACK)
  Message type: Boot Reply (2)
  Hardware type: Ethernet (0x01)
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0x00000000
  Seconds elapsed: 0
  Bootp flags: 0x0000 (Unicast)
  Client IP address: 100.100.100.2
  Your (client) IP address: 100.100.100.2
  Next server IP address: 100.100.100.254
  Relay agent IP address: 0.0.0.0
  Client MAC address: 30:30:3a:30:30:3a (30:30:3a:30:30:3a)
  Client hardware address padding: 30303a30303a30303a30
  Server host name: not given
  Boot file name not given
  Magic cookie: DHCPR
  Option: (53) DHCP Message Type (ACK)
  Option: (54) DHCP Server Identifier (100.100.100.254)
  Option: (6) Domain Name Server
  Option: (1) Subnet Mask (255.255.255.0)
  Option: (3) Router
  Option: (51) IP Address Lease Time
    Length: 4
    IP Address Lease Time: (86400s) 1 day
  Option: (255) End
No.: 5 Time: 1.520076382 - Source: 100.100.100.254 - Destination: 255.255.255.255 - Protocol: DHCP - Length: 318 - Info: DHCP ACK - Transaction ID 0x0
 Show packet bytes
 Help

```

בתמונה אפשר לראות את הפאקטה DHCP ack בחלק הבהיר את הIP מוקור
100.100.100.254 DHCP של השרת IP 255.255.255.255 ועוד 255.255.255.255 כדי לשדר
ברודקאסט , בחלק האדום את ה포רטים שעובדים אותם בdhcp פורטים 67 ו 68
בירוק את סוג הבקשה שהוא offer ועוד הגדרות DHCP כמו subnet mask
, IP של DNS ולכמיה זמן אני הIP מקצה ליתר הכתובת את IP
שהקצנו לנו.

DNS

שרת DNS הוא שרת "ש망יר" בתובת אינטרנט וodomianים בתובת IP של אותוodomian.

לבנית שרת DNS עבדנו עם ספריית scapy שאיתה נפרק פאקוטות DNS ונרכיב פאקוטות חדשות של DNS.

תחילה הגדרנו משתנה גלובלי בשם ipaddres שיביל את הIP של השירות DNS.

```
if __name__ == '__main__':
    getipfordns()
    while True:
        sniff(count=3, filter="udp port 53", prn=dnssniffer, iface="lo")
```

הגדרנו שבהתחלת לשמורים את השירות DNS הוא ניגש ל DHCP לקבל IP בעזרת פונקציית getipfordns.

לאחר מכן נרצה בלולאה אינסופית ונסניף פאקוטות UDP בפורט 53 במשת O שזה המשך של המחשב שמדובר עמו עצמו.

פונקציית getipfordns:

```
def getipfordns():
    tupleip = dhcp.getip()
    ipaddres = tupleip[0]
```

היא מבקשת מ DHCP בתובת IP ומכניסה למשתנה ipaddres את הIP שקיבלנו.

```
def dnssniffer(packet):
    domain = packet[DNSQR].qname.decode()
    print(f"Received DNS request for {domain}")
    domainstr = str(packet[DNSQR].qname)
    if "royandyuval.com" in domainstr:
        ip = "12.3.20.3"
```

לאחר שהסנו פאקטת DNS נחלץ ממנה את הדומיין ונמיר אותה מביטים לסטירינג.
נבצע בדיקה אם הדומיין שקיבלנו בבקשתו הוא הדומיין של האפליקציה שלנו, אם כן
נחזיר את הכתובת 12.3.20.3.

```
else:
    ip = socket.gethostbyname(domain)
    dns_response = DNS(
        id=packet[DNS].id,
        qr=1,
        qd=packet[DNS].qd,
        an=DNSRR(rrname=domain, rdata=ip))
    dnsresponse = IP(dst=packet[IP].src, src=packet[IP].dst) / UDP(dport=packet[UDP].sport, sport=packet[UDP].dport) / dns_response
    sendp(dnsresponse)
```

אם קיבלנו בקשה לדומיין שהוא לא האפליקציה שלנו נבקש את הכתובת IP של הדומיין דרך פונקציית `gethostbyname` של ספריית `socket`.

יצור פאקטה חדשה שתיהיה הפקעת תשובה בר:

נגידר שהפאקטה היא תשובה באמצעות `qr=1` וכןיף לפאקטה את הIP שלה ולבסוף נחבר את כל השבבות ונשלח את הפאקטה.

הרצה של השירות DNS:

הרצה של שאלחת DNS לכתובת של אתר האוניברסיטה של אריאל,

```
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 dnsserver.py
Received DNS request for www.ariel.ac.il.
.
Sent 1 packets.
Received DNS request for www.ariel.ac.il.
.
Sent 1 packets.
Received DNS request for www.ariel.ac.il.
.
Sent 1 packets.
^Z
[6]+  Stopped                  sudo python3 dnsserver.py
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$
```

```
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 clientdns.py
34.96.118.58
True
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$
```

ניתן לראות מצד שמאל את השירות שהוא מקבל הודעה DNS לדומיין www.ariel.ac.il.

מצד ימין ניתן לראות את הIP שחוזר למי שביצע את הבקשתה.

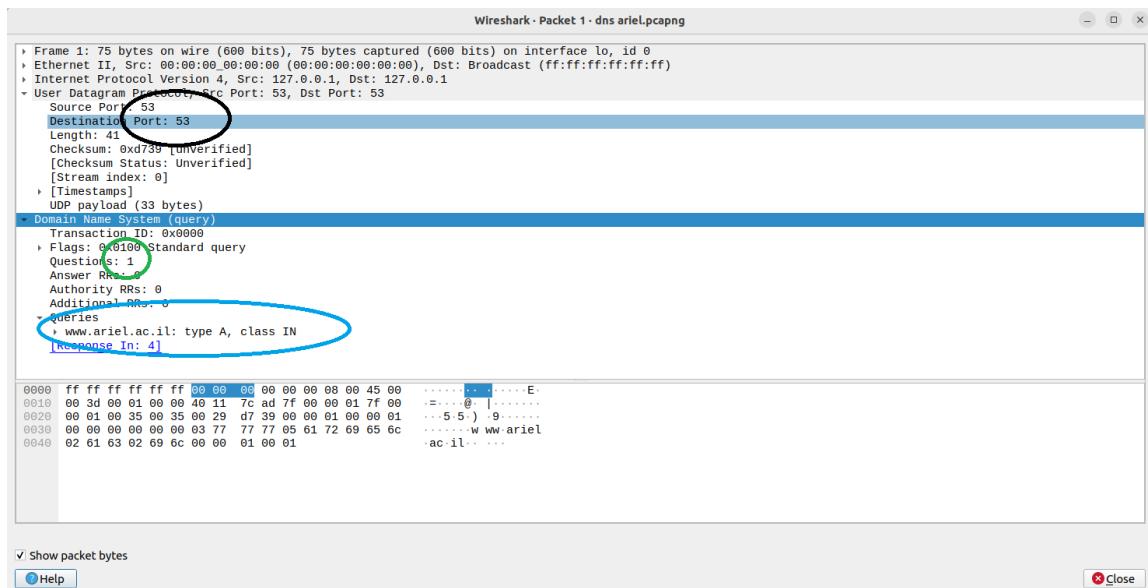
תמונות dns ariel.pcap ל��וחות מהקלטה wireshark

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	127.0.0.1	127.0.0.1	DNS	75	Standard query 0x0000 A www.ariel.ac.il
2	0.000868890	127.0.0.1	127.0.0.53	DNS	102	Standard query response 0x3e04 A www.ariel.ac.il A 34.96.118.58 OPT
3	0.001060402	127.0.0.53	127.0.0.1	DNS	102	Standard query response 0x3e04 A www.ariel.ac.il A 34.96.118.58 OPT
4	0.027792843	127.0.0.1	127.0.0.1	DNS	106	Standard query response 0x0000 A www.ariel.ac.il A 34.96.118.58 OPT
5	0.044273579	127.0.0.1	127.0.0.53	DNS	86	Standard query response 0x1874 A www.ariel.ac.il OPT
6	0.044534417	127.0.0.53	127.0.0.1	DNS	109	Standard query response 0x1874 A www.ariel.ac.il A 34.96.118.58 OPT
7	0.0669529827	127.0.0.1	127.0.0.1	DNS	106	Standard query response 0x0000 A www.ariel.ac.il A 34.96.118.58 OPT
8	0.087572524	127.0.0.1	127.0.0.53	DNS	86	Standard query response 0x1170 A www.ariel.ac.il OPT
9	0.087717203	127.0.0.53	127.0.0.1	DNS	102	Standard query response 0xf176 A www.ariel.ac.il A 34.96.118.58 OPT
10	0.104688126	127.0.0.1	127.0.0.53	DNS	106	Standard query response 0x3e04 A www.ariel.ac.il A 34.96.118.58

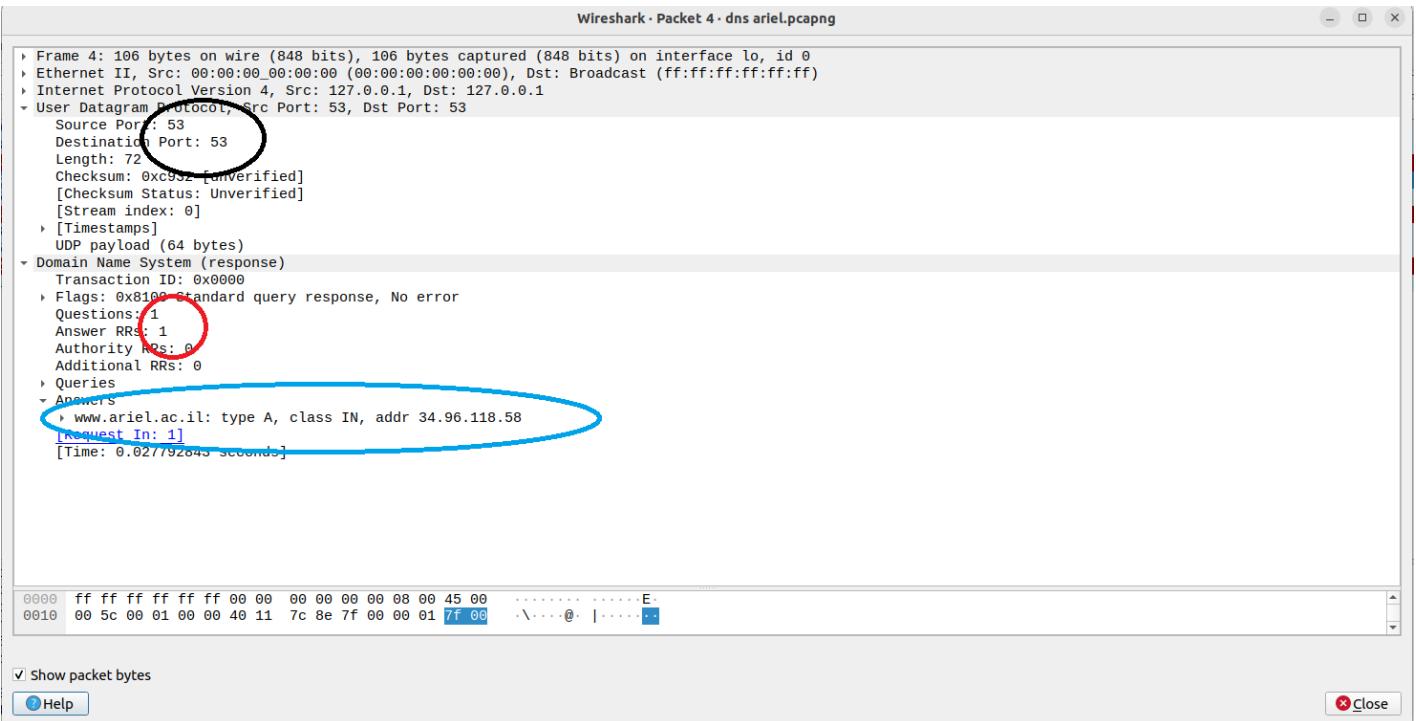
בתמונה אפשר לראות את שאלתת הבקשה לכתובת IP לדומיין של ארייל בצד ימין.

בצד ימין אפשר לראות את התשובה DNS החיצוני של IP של הדומיין של ארייל.

בצד ימין ניתן לראות שאנו מקבלנו מבחן ברגע לדומיין של ארייל בחזרה למי שביצע את הבקשה.



בתמונה אפשר לראות את הפקטה שלחו לשרת DNS, בצד ימין אפשר לראות את הפורטים שהם 53 שמתאים לDNS, בצד ימין אפשר לראות שהגשנו רק בקשה אחת, ובצד ימין ניתן לראות לאיזה דומיין הגשנו את הבקשה.



בתמונה אפשר לראות את הפאקטת תשובה מהשרת DNS, במצב שחור אפשר לראות את הפורטים שהם 53 שמתאים לDNS, במצב אדום אפשר לראות שהגשנו רק בקשה אחת וקיבלנו תשובה אחת, ובצבע הבהיר ניתן לראות לאיזה דומיין הגשנו את הבקשה וביחד איתו את התשובה שקיבלנו עם הIP.

הרעה של שאלת DNS לדומיין של האפליקציה שלנו,

```

roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 dnsserver.py
Received DNS request for www.royandyuval.com.
.
Sent 1 packets.
Received DNS request for www.royandyuval.com.
.
Sent 1 packets.
Received DNS request for www.royandyuval.com.
.
Sent 1 packets.
^Z
[13]+  Stopped                  sudo python3 dnsserver.py
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ sudo python3 clientdns
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ 12.3.20.3
roy@roy-VirtualBox:~/PycharmProjects/FinalProject$ 
```

ניתן לראות כיצד שמאלי את השרת כשהוא מקבל הודעה DNS לדומיין www.royandyuval.com

בצד ימין ניתן לראות את הIP שוחזר למי שביצע את הבקשה.

תרומות dns royandyuval.pcap לקוחות מהקלטה wireshark

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.0.1	DNS	79	Standard query 0x0000 A www.royandyuval.com
2	0.032512757	127.0.0.1	127.0.0.1	DNS	114	Standard query response 0x0000 A www.royandyuval.com A 12.3.20.3
3	0.068133863	127.0.0.1	127.0.0.1	DNS	114	Standard query response 0x0000 A www.royandyuval.com A 12.3.20.3
4	0.099319922	127.0.0.1	127.0.0.1	DNS	114	Standard query response 0x0000 A www.royandyuval.com A 12.3.20.3

בתמונה אפשר לראות את הבקשה שנשלחה לשרת DNS בצלב יירוק ובצבע הכלול את ה IP של האפליקציה שאנו חזוים.

```

User Datagram Protocol, Src Port: 53, Dst Port: 53
  Source Port: 53
  Destination Port: 53
  Length: 45
  Checksum: 0xf8b1 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  > [Timestamps]
  UDP payload (37 bytes)
-> Domain Name System (query)
  Transaction ID: 0x0000
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  -> Queries
    > www.royandyuval.com: type A, class IN
    [Response In: 2]
  
```

בתמונה אפשר לראות את הפקטה ששלחו לשרת DNS, בצלב הכלול אפשר לראות את הפורטים שהם 53 שמתאים ל DNS, בצלב בתום אפשר לראות שהגשנו רק בקשה אחת, ובצבע יירוק ניתן לראות לאיזה דומיין הגשנו את הבקשה.

```

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
-> User Datagram Protocol, Src Port: 53, Dst Port: 53
  Source Port: 53
  Destination Port: 53
  Length: 80
  Checksum: 0xe358 [unverified]
  [Checksum Status: Unverified]
  [Stream index: 0]
  > [Timestamps]
  UDP payload (72 bytes)
-> Domain Name System (response)
  > Transaction ID: 0x0000
  > Flags: 0x1100 Standard query response, No error
    Questions: 1
    Answer RRs: 1
    Authority RRs: 0
    Additional RRs: 0
  -> Answers
    > www.royandyuval.com: type A, class IN, addr 12.3.20.3
    [Retransmitted response. Original response in: 2]
    [Retransmission: True]
  
```

בתמונה אפשר לראות את הפקטה תשובה מהשרת DNS, בצלב הכלול אפשר לראות את הפורטים שהם 53 שמתאים ל DNS, בצלב בתום אפשר לראות שהגשנו רק בקשה אחת וקיבלו תשובה אחת, ובצבע הירוק ניתן לראות שהגשנו לדומיין של האפליקציה את הבקשה וביחד איתו את התשובה שקיבלו עם ה IP.

TCP – tcpipSocket

בחלק זה נסביר על הקוד שכתבנו למימוש סוקט TCP.

געבור חלק חלק ונסביר כל פונקציה.

קבינסטנסיות:

1. SOCKET_MAX_TIMEOUT - מספר השניות המקסימלי שהסוקט יוכל להיות במצב המתנה. הגדרנו ל-60 שניות.
2. MAX_SIMULTANEOUS_CONNECTIONS – מספר החיבורים המקסימלי שהסוקט יוכל לקבל בו זמן. הגדרנו ל-5 חיבורים בו זמן.

:TCPIPSocket אובייקט

ה-Data members

```
class TCPIPSocket:  
    # the ipv4 address and port of the receiver side (or this server as a receiver)  
    receiverAddress = None  
  
    # the tcpip socket we open to the receiver side as senders, or as a receiver for listening  
    tcpipSocket = None
```

המוביל בתובותיו מסוג IPV4 ופורט של הצד מקבל.
tuple = receiverAddress
.TCP = tcpipSocket

פונקציות:

:Connect(1

```
def connect(self, address):
    # save the receiver address
    self.receiverAddress = address
    # open a TCPIP socket and connect to the receiver host and port
    self.tcpipSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.setTimeout(SOCKET_MAX_TIMEOUT)
    # connect to the client IP and port using the created socket
    self.tcpipSocket.connect(address)
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket

תפקידה של הפונקציה הוא להתחבר לצד מקבל בעזרת בתובות קו ופורט של הצד המקבל.

הfonקציה מקבלת tuple address בשם שמכיל בתובות קו ופורט של הצד מקבל ומבצעת השמה של זה לתוך הdata member בשם .receiverAddress

אל תוך ה data member בשם tcpipSocket מבצעת השמה של סוקט tcp שנוצר בעזרת הפונקציה () socket.socket() אשר מקבלת את AF_INET (מסמל בתובות קו מסוג IPV4) ואת SOCK_STREAM (מסמל שהסוקט מסוג TCP).

לאחר מכן, מגדירה שהטימאאות של הסוקט הנוכחי יהיה 60 שניות.

בנוסף, מחברת את הצד מקבל דרך סוקט לצד השולח בעזרת connect() שמקבלת .address.

:close(2

```
def close(self):
    # close the tcpip socket
    self.tcpipSocket.close()
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket

תפקידה של הפונקציה הוא לסגור את הסוקט בעזרת הפונקציה () close של ספריית .socket

:send(3

```
def send(self, dataToSend):
    self.tcpipSocket.send(dataToSend)
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket.
תפקידה של הפונקציה הוא לשלוח מידע שנשלח בתור בתים וכן מקבלת בפרמטר את המידע.
שולחת את המידע באמצעות הפונקציה ()send של סדרית socket.

:setTimeout(4

```
def setTimeout(self, socketMaxTimeout):
    self.tcpipSocket.settimeout(socketMaxTimeout)
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket.
תפקידה של הפונקציה הוא להגדיר את הזמן המаксימלי של הsocket במצב המתנה.
לכן מקבלת בפרמטר את הזמן.
מגדירה את הטימראות באמצעות הפונקציה ()settimeout של סדרית socket.

:receive(5

```
def receive(self, maxBufferLength):
    dataReceived = self.tcpipSocket.recv(maxBufferLength)
    return dataReceived
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket.
תפקידה של הפונקציה הוא לקבל מידע מלקוחות(הצד השולח) ולהחזיר אותו.
מקבלת בפרמטר את הגודל המרבי של הבארך לקבלת מידע כל פעם.
מגדירה זאת באמצעות הפונקציה ()recv של סדרית socket.

:listen(6

```
def listen(self, address):
    # save the receiver address
    self.receiverAddress = address
    # open a TCPIP socket and bind it to the host & port
    self.tcpipSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.tcpipSocket.bind(address)
    self.tcpipSocket.listen(MAX_SIMULTANEOUS_CONNECTIONS)
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket

תפקידה של הפונקציה הוא לחבר את הsocket של הצד המקבל לחיבורים של לקוחות, וכן מקבלת כפרמטר tuple address בשם שມכיל כתובות IP ופורט של הצד המקבל.

הfonקציה מבצעת השמה של data member address לערך receiverAddress בשם .receiverAddress

לאחר מכן, אל תוך ה data member tcpipSocket מבצעת השמה של סוקט tcp שנוצר בעזרת הפונקציה () אשר מקבלת את AF_INET(מסמל כתובות IP) ו-SOCK_STREAM(מסמל שהsocket מסוג TCP).

לאחר מכן מחברת את הsocket לבתוות הקו והפорт שהתקבלו כפרמטר address באמצעות הפונקציה () bind של ספריית socket.

בנוסף, מתחילה את הsocket להקשיב ל 5 חיבורים במקביל.

:accept(7

```
def accept(self):
    # wait & accept incoming connections
    clientSocket, clientAddress = self.tcpipSocket.accept()
    # create a new TCPIPSocket for the connected client and return it to caller
    clientTCPIPSocket = TCPIPSocket()
    # set client parameters to the TCPIPSocket
    clientTCPIPSocket.tcpipSocket = clientSocket
    clientTCPIPSocket.receiverAddress = clientAddress
    # return the new TCPIPSocket and clientAddress
    return clientTCPIPSocket
```

פונקציה זאת מבוצעת על האובייקט .TCPIPSocket

תפקידה של הפונקציה הוא לקבל חיבור מלוקה ולהחזיר את הסוקט, בתובת `i` וה포רט של הלוקה הנ"ל.

הfonקציה מקבלת את הסוקט ואת בתובת הקו והפורט(`tupple`) ובמצעת השמה למשתנים בשם `clientAddress` ו `clientSocket` בהתאם בעזרת הפונקציה `.socket()` של ספריית `accept()`

יצרת סוקט `tcp` חדש בשם `clientTCPIPSocket` על ידי האובייקט `.TCPIPSocket` מבצעת לו השמה למשתנה `data members` של סוקט הלוקה ובתוות הקו והפורט של הלוקה ומחייבת אותו בסוקט `tcp` שלם.

RUDP

בחלק זה נסביר על הקוד שכתבנו למימוש סוקט RUDP. נعبر חלק חלק ונסביר כל פונקציה.

קבינסטנסיות:

1. MTU – גודל מקסימלי של פאקטת מידע. הגדרנו ל1024 בתים.
2. HEADER_LENGTH – גודל header של הפאקטה, הגדרנו ל12.
3. SOCKET_MAX_TIMEOUT – מספר השניות המקסימלי שהסוקט יכול להיות במצב המתנה. הגדרנו ל60 שניות.
4. SLEEP_BETWEEN_RETRIES – מספר השניות של sleep על thread לפני שליחה מחדש של פאקטה, כאשר הגענו למספר הפאקטות המקסימלי שאבדו.
5. MAX_SEND_RETRIES – המספר המаксימלי לשילוח מחדש של פאקטה, אם מגיעים למספר זהה תיזורק שגיאה. הגדרנו ל600.(בערך 30 שניות)
6. MAX_WINDOW_SIZE – גודל החלון, מספר הפאקטות המקסימלי שנשלחות בו בזמןית.
7. SYN_PACKET_TYPE – בית שמסמל שזאת פאקטת SYN.(מוגדרת 0)
8. PACKET_TYPE_DATA – בית שמסמל שזאת פאקטת DATA.(מוגדרת 1)
9. PACKET_TYPE_ACK – בית שמסמל שזאת פאקטת ACK.(מוגדרת 2)
10. PACKET_TYPE_END – בית שמסמל שזאת פאקטת END.(מוגדרת 3)
11. PACKET_TYPE_RST – בית שמסמל שזאת פאקטת RST.(מוגדרת 4)

מימושו את RUDP באמצעות העקרונות הבאים:

אמינות – על כל פאקטה שנחנכו שולחים אנחנו מחייבים לאריך עבור אותה פאקטה ובמקביל אם לא הגיע ACK שולחים את הפאקטה שוב.

congeston control - על כל פאקטה שנחנכו מקבלים ACK כמו שצרכן באופן ישיר אנחנו מגדילים את החלון באחד. כאשר שלחנו פאקטה מסוימת יותר מפעם אחת סימן שהיא איבוד ולכן נקטין את החלון ב-1.

- **Flow control**

הערה חשובה: כדי להריץ את אפליקציית הקט על בסיס RUDP, יש לשנות את TCP/IP בקובץ `ftp_server.py` ל-`False` כי אחרת הוא מוגדר לרוץ על בסיס TCP.

פונקציית parsePacket

```
def parsePacket(receivedPacket):
    # get the first 4 bytes and convert them into int, this will be the received packetType
    receivedPacketType = int.from_bytes(receivedPacket[0:4], 'big')

    # get the next 4 bytes and convert them into int, this will be the received sequenceNumber
    receivedSequenceNumber = int.from_bytes(receivedPacket[4:8], 'big')

    # get the next 4 bytes and convert them into int, this will be the received dataLength
    receivedDataLength = int.from_bytes(receivedPacket[8:HEADER_LENGTH], 'big')

    # get the last bytes from end of header, read up to receivedDataLength and set it as received data
    receivedData = receivedPacket[HEADER_LENGTH:HEADER_LENGTH + receivedDataLength]

    return receivedPacketType, receivedSequenceNumber, receivedDataLength, receivedData
```

פונקציה שמקבלת פאקטה כstring ומחזירה 4 ערכים – הערך של הפאקטה, payloadLength של הפאקטה, אורך המידע בפאקטה (אורך header), payloadNumber של הפאקטה.

הערך של הפאקטה מתקבל על ידי המרת 4 הביטים הראשונים של string לint, בסדר ביטים של bigEndian (משמאל לימין).

payloadNumber של הפאקטה מתקבל על ידי המרת 4 הביטים הבאים של string, בסדר ביטים של bigEndian (משמאל לימין).

אורך payload מתקבל על ידי המרת 4 הביטים הבאים (עד אורך header שהוא 12) של string לint, בסדר ביטים של bigEndian (משמאל לימין).

הpayload של הפאקטה מתקבל על ידי קבלת הביטים מסוף header ועד גודל header+אורך payload, כדי למצוא את סוף הפאקטה הנוכחית.

אובייקט :RUDPSocket

ה-Data members-

```

class RUDPSocket:
    # a flag that indicates if the socket is opened and connected and has finished the handshake (sent & received SYN)
    isConnected = False

    # flag indicating connection is closed
    isClosed = False

    # an event that indicates the socket has connected and is open
    isConnectedEvent = threading.Event()

    # a buffer that accumulate the DATA packets in correct order into a full byte data buffer
    receivedDataBuffer = b''

    # a flag that indicates if a data was read from the socket and is ready to be consumed by the caller
    isDataReady = False

    # an event that indicates the data is ready for read by the caller
    isDataReadyEvent = threading.Event()

    # the udp socket we open to the receiver side as senders, or as a receiver for listening
    rudpSocket = None

    # the ipv4 address and port of the receiver side on this socket
    receiverAddress = None

    # the ipv4 address and port of this side on the socket
    selfAddress = None

    # a sequence number that is incremented each time a packet is sent (the max sequence is 65,535 [FFFF])
    sequenceNumber = 0

```

```

    # thread lock to use when changing the sequenceNumber member
    sequenceNumberLock = threading.Lock()

    # a member that holds the RUDP window size (max simultaneous sent packets)
    windowSize = 1

    # thread lock to use when changing the windowSize member
    windowSizeLock = threading.Lock()

    # dictionary that holds all the sequence numbers and packets that were not acknowledge yet
    waitingForAcknowledge = {}

    # thread lock to use when changing the waitingForAcknowledge member
    waitingForAcknowledgeLock = threading.Lock()

```

isConnected = דגל שמסמן שהסוקט נפתח, הת לחבר וסיים את "לחיצת הידיים", משמע שלח וקיבל SYN.

isClosed = דגל שמסמן שהסוקט סגור.

isConnectedEvent = thread event של שמתריע שהסוקט הת לחבר ופתח.

recieverDataBuffer = סטרינג ריק (של bytes) שמשמש בuffer ל-data.

isDataReady = דגל שמסמן שהdata נקראת מהסוקט ומובנה.

isDataReadyEvent = event של thread שמתրיע שהdata מוכנה לקריאה.

rudpSocket = סוקט udp(נפתח מהצד השולח לצד המקלט או במקבל בשבייל האזנה).

receiverAddress = כתובת קו ופורט של הצד המקלט.

selfAddress = כתובת קו ופורט של הסוקט עצמו(האובייקט).

sequenceNumber = מספר של הפאקטה הנשלחת, גדל ב-1 כל פעם שנשלחת פאקטה חדשה. (מוגדר ב-0 בהתחלה)

sequenceNumberLock = מנעול שנטפס על ידי thread ייחיד כדי לבצע שינוי ב sequenceNumber ומגן על שינויים(sequenceNumber) ב-zerospace מהרבים.

windowSize = גודל חלון השיליחה(**מספר הפאקטות המקסימלי שניתן לשולח בו זמינות**). מוגדר ב-1 בהתחלה.

windowSizeLock = מנעול שנטפס על ידי thread ייחיד כדי לבצע שינוי בגודל החלון ומגן על שינויים (windowSize) ב-zerospace מהרבים.

waitingForAcknowledge dictionary = שמחזיק key: sequenceNumber של פאקטות שלא קיבל ack ובvalue: את הפאקטות עצמן.

waitingForAcknowledgeLock = מנעול שנטפס על ידי thread ייחיד כדי לבצע שינוי ב-waitingForAcknowledge ומגן על שינויים ב-waitingForAcknowledge מהרבים.

פונקציות:

:Connect(1

```
def connect(self, address):
    # save the address we are connecting to (the other side address)
    self.receiverAddress = address

    # create a UDP socket, and send SYN to receiver
    self.rudpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.rudpSocket.settimeout(SOCKET_MAX_TIMEOUT)
    self.sendSynPacket()

    # launch a thread that listen to received control packets (ACK/SYN/END messages)
    listenerThread = threading.Thread(target=self.handleControlPackets)
    listenerThread.start()

    # launch thread that retransmission waiting for ACK packets
    retransmissionThread = threading.Thread(target=self.retransmitWaitingPackets)
    retransmissionThread.start()
```

פונקציה זאת מבוצעת על האובייקט **RUDPSocket**.

פונקציה שיצרת חיבור באמצעות סוקט **UDP**.

שומרת את הכתובת וה포רט שאליים אנחנו מתחברים.

יצרת סוקט **udp**, מגדרה לו **timeout** ושולחת פאקטה מסוג **SYN**.

יצרת **thread** שמאזין לפאקטות **ACK\SYN\END**, ומפעילה אותן.

יצרת **thread** נוספת שתפקידו לעשות **retransmission** לפאקטות שלא קיבלן **ack** ומפעילה אותן.

:close(2

```
def close(self):
    if self.isConnected:
        self.sendRSTPacket()
        # mark sender socket as closed
        self.isConnected = False
        self.isClosed = True
        # close the udp socket
        self.rudpSocket.close()
```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.
 נעשית בדיקה: אם הsocket מחובר, אז נשלח פקעת RST.
 לאחר מכן מוגדר `isClosed` כ`True` והsocket נסגר.

:send(3

```
def send(self, dataToSend):
    # make sure the socket is connected (SYN has been sent and received)
    if not self.isConnected:
        self.isConnectedEvent.wait(SLEEP_BETWEEN_RETRIES * MAX_SEND_RETRIES)

    # slice the data into smaller chunks at MTU size
    # calculate what is the total bytes we are about to send in this packet
    totalBytesToSend = len(dataToSend)
    # reset the total sent bytes counter
    totalBytesSent = 0
    # loop until there is nothing left to send
    while totalBytesSent < totalBytesToSend:
        # get the next chunk of bytes in the MTU size from the data to send
        nextDataChunkToSend = dataToSend[totalBytesSent:MTU]

        # if we reached the maximum number of lost packets waiting for acknowledgement
        # then wait until one of them succeed before you send the next packet
        number_of_retries = 0
        while len(self.waitingForAcknowledgement) >= self.windowSize:
            if number_of_retries >= MAX_SEND_RETRIES:
                # clear the waiting for ack dictionary so next send will start fresh
                with self.waitingForAcknowledgementLock:
                    log("send(): Got waitingForAcknowledgementLock")
                    self.waitingForAcknowledgement.clear()
                    raise Exception('Failed to send data, not all packets got ACKs')
            number_of_retries = number_of_retries + 1
            time.sleep(SLEEP_BETWEEN_RETRIES)
```

```

# if we had to wait for ack to return then reduce the window size so fewer packets are lost
if numberOfRetries > 0:
    self.reduceWindowSize()

# send the current data chunk
self.sendDataPacket(nextDataChunkToSend)
# add another chunk size to the totalBytesSent counter
totalBytesSent = totalBytesSent + MTU

# after all packets are sent, send the END packet
self.sendENDPacket()

# wait for all ACK packets to return or raise exception after timeout has reached
numberOfRetries = 0
while len(self.waitingForAcknowledge) > 0:
    if numberOfRetries >= MAX_SEND_RETRIES:
        # clear the waiting for ack dictionary so next send will start fresh
        with self.waitingForAcknowledgeLock:
            log("send(): Got waitingForAcknowledgeLock again")
            self.waitingForAcknowledge.clear()
            raise Exception('Failed to receive ACK packets for all the sent packets')
    numberOfRetries = numberOfRetries + 1
    time.sleep(SLEEP_BETWEEN_RETRIES)

```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.
ראשית נבדוק אם הסקוט פועל, ולאחר מכן גדר 2 משתנים: אחד יחזיק את גודל המידע שיש לשלוח והשני בכמה ביטים שלחנו עד כה.
נרצה בולולאה כל עוד לא שלחנו את כל הביטים, נקח את ה `chunck` הבא שנרצה לשלוח ונגדיר משתנה שיספור לנו כמה פאקטות מਮותינות לאישור.
ואז נכנס לולאה שרצה כל עוד מספר הפקאות שמחכotta לאישור גדול או שווה לגודל החלון, וביצוע בדיקה:
אם שלחנו את הפאקטה 600 פעמים ולא קיבלנו עלייה `ack` אז נאפס את המילון שמחזיק את הפאקטות שמחכotta ל `ack` ונזרוק שגיאה, אם שלחנו פחות מ- 600 פעמים נוסיף 1 למספר הניסיונות.
אם לא קיבלנו `ack` ישර אחריו שלחנו את הפאקטה ונקטין את גודל החלון ב 1 באמצעות פונקציית `reduceWindowSize`.
נשלח את הפאקטה ונוסיף את כמות הביטים שלחנו למשתנה שסופר כמה ביטים שלחנו עד כה.

אחריו שלחנו את בול המידע נשלח פאקטה של END.

לבסוף הגיעו לlolaha שרצה כל עוד יש לנו פאקטות שלא קיבלן ack עדין, נבצע בדיקה:

אם שלחנו את הפאקטה 600 פעמים ולא קיבלנו עליה ack אז נאפס את המילון שמחזיק את הפאקטות שמחוברות ל-ack ונדריך שגיאה, אם שלחנו פחות מ-600 פעמים נסיף 1 למספר הבניינונות.

:setTimeout(4

```
def setTimeout(self, socketMaxTimeout):
    self.rudpSocket.settimeout(socketMaxTimeout)
```

פונקציה זאת מבוצעת על האובייקט `.RUDPSocket`.
הfonקציה מגדרת את הזמן המקסימלי שהסוקט יחבר לפעולות האזנה, שליחה
וקבלה.
נגיד על הסוקט את הזמן שעליו להמתין עד לקבלת מידע אחרת יקבל `timeout`.

:receive(5

```
def receive(self, maxBufferSize):
    if not self.isClosed:
        # wait for socket to connect
        if not self.isConnected:
            self.isConnectedEvent.wait(SLEEP_BETWEEN_RETRIES * MAX_SEND_RETRIES)

        # wait for data to be ready and return it
        if not self.isDataReady:
            self.isDataReadyEvent.wait(SLEEP_BETWEEN_RETRIES * MAX_SEND_RETRIES)
            self.isDataReadyEvent.clear()

        result = self.receivedDataBuffer
        self.receivedDataBuffer = b''
        self.isDataReady = False

    return result
```

פונקציה זאת מבוצעת על האובייקט `.RUDPSocket`.
הfonקציה מקבלת Bytes `data` מלקחות.

אם הsocket לא סגור, אז אם הsocket לא מחובר אז הevent שמתրיע שהsocket מחובר מקבל sleep.

בנוסף אם data לא מוכנה אז הevent שמתריע שהdata מוכנה מקבל sleep כדי לקבל את המידע כמו שציריך, לאחר מכן מנוקים את הevent.

התוצאה result מקבלת את מה שנכנס לבאפר ולאחר מכן מאפסת את הבאפר כדי שימושו לקלוט מידע. לבסוף, מוחזירה את result.

:listen(6

```
def listen(self, address):
    # open a UDP socket and bind it to host & port
    self.rudpSocket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.rudpSocket.settimeout(SOCKET_MAX_TIMEOUT)
    self.rudpSocket.bind(address)
```

פונקציה זאת מבוצעת על האובייקט RUDPSocket.

הfonקציה יוצרת סוקט UDP חדש ומגדירה את timeout הרצוי. לאחר מכן, מבוצעת עליו פעולה bind לבתובת דוא ופורט.

:accept(7

```
def accept(self):
    # wait & accept incoming connections
    # ask udp socket to receive data in the Max Transmission Unit size
    receivedPacket, clientAddress = self.rudpSocket.recvfrom(MTU)

    # create a new RUDPSocket
    clientRUDPSocket = RUDPSocket()
    # connect to the client with the newly created socket
    clientRUDPSocket.connect(clientAddress)
    # return the new RUDPSocket to the caller
    return clientRUDPSocket
```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.
הfonקציה מנסה לחברים ובשילוב חיבור מלוקה היא מחזירה סוקט, בתובת ד'
ופורט של הלוקה.

:retransmitWaitingPackets(8

```
def retransmitWaitingPackets(self):
    while True:
        # if socket has been closed - stop retransmitWaitingPackets thread from working
        if self.isClosed:
            break
        log(f"retransmitWaitingPackets() {self.waitingForAcknowledge}")
        with self.waitingForAcknowledgeLock:
            for currentSequenceNumer, currentPacket in self.waitingForAcknowledge.items():
                # log(f"retransmitWaitingPackets(): {currentPacket} to: {self.receiverAddress}")
                self.rudpSocket.sendto(currentPacket, self.receiverAddress)
        time.sleep(7)
```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.

תפקידה של הפונקציה הוא לשלוח מחדש פאקטות שלא קיבלו ack.

הfonקציה נכנסת לולאה אינסופית:

אם הsocket סגור יוצאים מהלולה.

אחרת הפונקציה ניגשת על ידי מניעול כדי למנוע גישה של threads.

היא עוברת בולולאת `for` על ה`waitForAcknowledge` dictionary,

היא עוברת על שני משתנים בכל איטרציה: על המספר הסידורי הנוכחי של הפאקטה ועל הפאקטה הנוכחיית.

בכל איטרציה היא שולחת מחדש את את הפאקטה הנוכחיית מחדש בתובת receiverAddress.

בכל איטרציה של הלולאת while נשים sleep.

: handleControlPackets(9

```
def handleControlPackets(self):
    # calculate the maximum number of expected packets, by calculating: maxBufferLength : (packetSize - headerSize)
    number_of_expected_packets = 5000 / (MTU - HEADER_LENGTH)
    # prepare a byte array to store all the received data packets
    received_data_array = [b''] * round(number_of_expected_packets)
    first_packet_sequence_number = 0
    while True:
        # if socket has been closed - stop handleControlPackets thread from working
        if self.is_closed:
            break
        # read until there is nothing to read
        try:
            # peek at the buffer and check if the next message is sent by client or ourselves
            peeked_data, peeked_address = self.rudp_socket.recvfrom(MTU, socket.MSG_PEEK)
            if peeked_address != self.self_address:
                # read bytes from the socket
                received_packet, client_address = self.rudp_socket.recvfrom(MTU)
                log(f"receive(): {received_packet} from: {client_address}")
                if received_packet and client_address != self.self_address:
                    # save the sender ip and port as the receiver address (so initiator port (8080) will be abandoned)
                    self.receiver_address = client_address
                    # parse the received packet
                    received_packet_type, received_sequence_number, received_data_length, received_data = parse_packet(received_packet)

                if received_packet_type == PACKET_TYPE_SYN:
                    log("receive(): Got SYN packet")
                    # received SYN packet from sender, mark socket as connected & reply with ACK with SYN SequenceNumber
                    # save the expected first packet sequence number, it will be used later
                    # to calculate each arriving data packet place in the receivedDataArray
                    first_packet_sequence_number = received_sequence_number + 1
                    self.send_ack_packet(received_sequence_number)
                    # sleep for 100 milliseconds to allow other side to consume the sent message
                    time.sleep(0.1)
                elif received_packet_type == PACKET_TYPE_DATA:
                    log(f"handle_sender_control_packets(): Got DATA packet, receivedSequenceNumber: {received_sequence_number}, firstPacketSequenceNumber: {first_packet_sequence_number}")
                    if not self.is_data_ready:
                        # received DATA packet from the sender add to total data buffer (in correct order) & return ACK
                        received_data_array[received_sequence_number - first_packet_sequence_number] = received_data
                        self.send_ack_packet(received_sequence_number)
                elif received_packet_type == PACKET_TYPE_ACK:
                    log("handle_sender_control_packets(): Got ACK packet")
                    # if ACK packet received from the receiver then remove the received SequenceNumber from the waitingForAcknowledge
                    with self.waiting_for_acknowledge_lock:
                        if len(self.waiting_for_acknowledge) > 0:
                            popped_packet = self.waiting_for_acknowledge.pop(received_sequence_number)
```

```

    if poppedPacket:
        # increase the window size (since we succeeded)
        self.increaseWindowSize()
        # if we received an ACK for the SYN then mark socket as connected
        poppedPacketType, poppedSequenceNumber, poppedDataLength, poppedData = parsePacket(poppedPacket)
        if poppedPacketType == PACKET_TYPE_SYN:
            self.isConnected = True
            self.isConnectedEvent.set()
            log("SYN ACK received")
    elif receivedPacketType == PACKET_TYPE_END:
        # received END packet that means the current data buffer transmission ended,
        # next packets belongs to the next data buffer, return data buffer to caller
        log("handleSenderControlPackets(): Got END packet")
        self.receivedDataBuffer = b''.join(receiveddataArray)
        receiveddataArray = [b''] * round(numberOfExpectedPackets)
        firstPacketSequenceNumber = receivedSequenceNumber + 1
        self.isDataReady = True
        self.isDataReadyEvent.set()
    elif receivedPacketType == PACKET_TYPE_RST:
        # received RST packet from sender, close the socket
        log("handleSenderControlPackets(): Got RST packet")
        self.isConnected = False
        self.close()
        break
    else:
        # if we received any other packet type then print error message
        log(f"handleSenderControlPackets(): unexpected packet type: {receivedPacketType}, ignoring it")
except Exception as err:
    # error occurred, maybe socket was closed by caller, break from loop
    if "timed out" not in str(err):
        if 'forcibly closed' not in str(err):
            log("Warning some problem occurred while trying to receive data from socket: " + str(err))
        else:
            # other side closed the socket, close this side too
            self.close()
            break

```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.

תחילה נגידר משתנה שיחזק את הגודל של מספר הפאקטות הצפי, רישימה ריקה. בוגדל המספר הצפי ומשתנה שיחזק את המספר הסידורי של הפאקטה שקיבלנו.

ברוץ בולולה אינסופית ונציג בלי לקרוא על הפאקטה הראשונה שנמצאת במאפר.

אם היאMPI שהוא שונה מהIP של מי שקורא את הפאקטה נמשיך.

נקרא את הפאקטה הזאת ונחלק אותה לחלק של המידע וחלק של הכתובת של השולח, ואז אם הפאקטה לא ריקה וגם היא לא נשלחה מאותו IP אז נשמר את הIP של השולח, ונשלח את המידע לפונקציית `parsePacket` לפירוק הפאקטה ל 4 חלקים – סוג, מספר סידורי, גודל המידע והמידע עצמוו.

עכשו נפעיל בהתאם לכל סוג האופציות שהגדכנו - אם קיבלנו פאקטה מסוג SYN נשלח על הودעה זו `.ack`.

אם הפאקטה מסוג DATA סימן קיבלנו מידע ולבן נוסיף את המידע זהה לרישימה שהגדכנו ונשלח `ack` על הודעה זו.

אם הפקאה מסוג ACK סימן שקיבלנו ack על הפקאה ששלחנו וכן נוציא אותה מהmillion שמחזיק את הפקאות שמתכונות ל ack ונגדיל את החלון באחד.

אם קיבלנו פאקטה ACK על הודעה SYN נסמן את הsocket כפעיל.

אם קיבלנו פאקטה מסוג END סימן שקיבלנו את כל המידע וכן נכניס את כל המידע לשנתנה אחד ונAPS את הרשימה.

אם הפקאה מסוג RST נסגור את הsocket.

בכל החלק של הטיפול בפקאה עטוף ב try | except אך אם באיזשהו שלב נזרקת שגיאה, נתפוש אותה ונדפיס שהיא שגיאה.

:getNextSequenceNumber(10

```
def getNextSequenceNumber(self):
    # acquire the sequence number lock so only this thread can change the sequence number value
    with self.sequenceNumberLock:
        if self.sequenceNumber < 65535:
            # increase the sequence number by 1
            self.sequenceNumber = self.sequenceNumber + 1
        else:
            # reset the sequence number back to 0 since it is about to exceed the 4 byte max number (FFFF)
            self.sequenceNumber = 0
    return self.sequenceNumber
```

fonkziah zat mbotzuta ul avobjeket .RUDPSocket.

Tefkida shel fonkziah ha la hachzir at ha mspur sidori ba shel fakotot.

Hofonkziah nigash ul ydi mnuel cdi l'mnou gisha shel threads.

Am ha mspur sidori (mspur fakotot shnshlhot vish ulihun makb) katan mmspur
ha apsoriot l'shaloch faktha shmorabbat 161 bityim, az ha zher sequenceNumber
ba chad.

Achrot, maapsim otu l'spura machad.

Lebasuf machzirim at ha mspur sidori chad.

: reduceWindowSize(11)

```
def reduceWindowSize(self):
    with self.windowSizeLock:
        if self.windowSize > 1:
            self.windowSize = self.windowSize - 1
```

פונקציה זאת מבוצעת על האובייקט .RUDPSocket
נבדוק אם גודל החלון גדול ממש מ 1, אם כן נחסיר ממנו אחד.
הגישה אל החלון נעשית על ידי מנעול.

: increaseWindowSize(12)

```
def increaseWindowSize(self):
    with self.windowSizeLock:
        if self.windowSize < MAX_WINDOW_SIZE:
            self.windowSize = self.windowSize + 1
```

פונקציה זאת מבוצעת על האובייקט .RUDPSocket
נבדוק שהגודל של החלון קטן ממש מהגודל המקסימלי שהגדכנו (10), אם כן נעלם
את גודל החלון ב 1.
הגישה אל החלון נעשית על ידי מנעול.

: sendSynPacket(13

```
def sendSynPacket(self):
    log("sendSynPacket()")
    # get the next valid sequence number, send the packet and add it to waiting for acknowledge dictionary
    sequenceNumber = self.getNextSequenceNumber()
    rudpPacket = self.sendRUDPPacket(PACKET_TYPE_SYN, sequenceNumber, bytes("", "utf-8"))
    with self.waitingForAcknowledgeLock:
        log("sendSynPacket(): Got waitingForAcknowledgeLock")
        self.waitingForAcknowledge[sequenceNumber] = rudpPacket
```

fonktsiya zot mbozuta ul' avobiikt RUDPSocket.

fonktsiya sholchta paketa msog SYN.

thchila nikch at mspur sidiroi ha'el paketa vneslich otta um dgl shiz'in
sz'at paketa ch Sy v mspur sidiroi shla, vnnchis otta l'dictionary shmbil at
paketoit shla kiblo ack udin.

hgisha al' waitingForAcknowledge nusiyt ul' idy mnugol.

: sendDataPacket(14)

```
def sendDataPacket(self, dataToSend):
    log("sendDataPacket()")
    # get the next valid sequence number, send the packet and add it to waiting for acknowledge dictionary
    sequenceNumber = self.getNextSequenceNumber()
    rudpPacket = self.sendRUDPPacket(PACKET_TYPE_DATA, sequenceNumber, dataToSend)
    with self.waitingForAcknowledgeLock:
        self.waitingForAcknowledge[sequenceNumber] = rudpPacket
```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.
הfonקציה שולחת פאקטה מסוג `Data` מתקבלת בפרמטר את המידע שיש לשלוח, לוקחת את המספר הסידורי של הפאקטה הבאה ושולחת את הפאקטה באמצעות `sendRUDPPacket` עם השדות – דגל `PACKET_TYPE_DATA`, מספר סידורי של הפאקטה זהו והמידע שיש לשלוח. שמצין שזאת פאקטה `Data`, מספר סידורי של הפאקטה זהו והמידע שיש לשלוח. ונכנס אותה ל`dictionary` שמכיל את הפאקטות שלא קיבלו `ack` עדין. הגישה אל `waitForAcknowledge` נעשית על ידי מנעול.

: sendENDPacket(15)

```
def sendENDPacket(self):
    log("sendENDPacket()")
    # get the next valid sequence number
    sequenceNumber = self.getNextSequenceNumber()
    self.sendRUDPPacket(PACKET_TYPE_END, sequenceNumber, bytes("", "utf-8"))
```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.
הfonקציה שולחת פאקטה מסוג `END`. פאקטה זו מיועדת להגיד שהלחנו את כל הפקאות שרצינו לשלוח. שולחת פאקטה `END` באמצעות `sendRUDPPacket` עם השדות – דגל `PACKET_TYPE_END` ומספר סידורי של הפאקטה זהו.

: sendRSTPacket(16

```
def sendRSTPacket(self):
    log("sendRSTPacket()")
    # get the next valid sequence number
    sequenceNumber = self.getNextSequenceNumber()
    self.sendRUDPPacket(PACKET_TYPE_RST, sequenceNumber, bytes("", "utf-8"))
```

fonkzia zat mbotzuta ul avbiket .RUDPSocket

fonkzia sholchta paketa msg RST.

sholchta paketa RST laipos bamezuot sendRUDPPacket um shidot - dagl
shmeizin shazat paketa RST | mspur sidori shel paketa hzo.

: sendAckPacket(17

```
def sendAckPacket(self, sequenceNumberToAck):
    log("sendAckPacket()")
    self.sendRUDPPacket(PACKET_TYPE_ACK, sequenceNumberToAck, bytes("", "utf-8"))
```

fonkzia zat mbotzuta ul avbiket .RUDPSocket

fonkzia sholchta paketa msg Ack

mekbalt bpermter ul iiza mspur sidori shel paketa lslch ack sholchta paketa
ack bamezuot sendRUDPPacket um shidot - dagl shmeizin shazat paketa ack
uel iiza paketa lslch ack.

: sendRUDPPacket(18

```
def sendRUDPPacket(self, packetType, packetSequenceNumber, packetData):
    # set the value of the packet header fields
    packetDataLength = len(packetData) # payload length, the length of the data bytes

    # convert the packet header fields into 4 bytes, in big-endian order
    packetTypeBytes = packetType.to_bytes(4, byteorder='big')
    packetSequenceNumberBytes = packetSequenceNumber.to_bytes(4, byteorder='big')
    packetDataLengthBytes = packetDataLength.to_bytes(4, byteorder='big')

    # create the packet header by combining the packetType, packetSequenceNumber, packetDataLength bytes together
    packetHeader = packetTypeBytes + packetSequenceNumberBytes + packetDataLengthBytes

    # create the packet by combining the packetHeader, packetBody together
    rudpPacket = packetHeader + packetData
    # print(rudpPacket)

    # send the RUDP packet to the receiver using the open socket
    log(f"sendRUDPPacket(): sending packet to: {self.receiverAddress}")
    self.rudpSocket.sendto(rudpPacket, self.receiverAddress)

return rudpPacket
```

פונקציה זאת מבוצעת על האובייקט `RUDPSocket`.
תפקידיה של הפונקציה הוא להרכיב פאקטה מסוג `RUDP`.
הפונקציה מקבלת בפרמטרים את סוג הפקטה, המספר הסידורי שלה ואת ה `data` עצמה.
מחזיקים משתנה שמכיל את אורך ה `payload`.
מmirים את סוג הפקטה ל `4 bytes` בסדר של `big-endian`.
מmirים את המספר הסידורי של הפקטה ל `4 bytes` בסדר של `big-endian`.
מmirים את אורך ה `payload` ל `4 bytes` בסדר של `big-endian`.
משרשרים את סוג הפקטה, המספר הסידורי ואורך ה `payload` אשר מומרים ל `header bytes` לרץ אשר מרכיב את `header` של הפקטה.
מחברים את `header` עם `data` עצמה כדי לקבל את הפקטה בשלמותה.
לאחר מכן שולחים את פאקטה ה `rudp` לצד המქבל על ידי ה `socket` לבתוות של הצד המქבל.
לאחר מכן מחזירים את ה `socket` עצמו.

FTP Application server

בחלק זה נסביר על הקוד שכתבנו למימוש שרת ה-FTP.

מעבר לכך חלק ונסביר כל פונקציה.

בקוד יישמו דברים שלמדו בקורסים מונחה עצמים ומבנה מערכות אשר תרמו מאוד לישום חלק מהפונקציות.

קבינטנטות:

1 SERVER_HOST : עטוף ב `try except` על מנת למצוא את את בתובת הקן הנוכחי של המכונה, ואם הפונקציה נתקלה בשגיאה אז הכתובת מוגדרת דיפולטיבית כתובת 127.0.0.1.

2 SERVER_PORT : הפורט של השירות שמוגדר ל 20+שלוש ספרות אחרונות של הת"ז של יובל.

3 RETURN_PORT : הפורט של הקליינט שמוגדר ל 30+שלוש ספרות אחרונות של הת"ז של רוי.

4 DEFAULT_USER : שם המשתמש שאיתו נתחבר לשרת.

5 DEFAULT_PASSWORD : הסיסמה שאיתה נתחבר לשרת.

6 isListening : דגל שמסמן אם השירות מאשר או בבוי.

7 allThreads : שכביל שמות threads פעילים.

8 allow_delete : דגל שמסמן אם המשתמש רשאי למחוק קבצים\תיקיות בשרת.

9 isTCPIP : דגל שאם הוא True אז ftp מתבסס על פרוטוקול TCP ואם False אז RUDP. (הדיפולט הוא True)

10 mainServerSocket : הsocket של השירות שמשמש לקבל חיבורים.

11 allConnectsClients : מערך שמחזיק את הלקוחות המחוברים הנוכחיים.

אובייקט :FtpServerProtocol

האובייקט יורש ממחלקה `.threading.Thread`

:Data members-ה

```

class FtpServerProtocol(threading.Thread):
    cwd = "/"
    threadName = "Th"
    dataSocketIP = "127.0.0.1"
    dataSocketPort = RETURN_PORT
    rest = False
    pasv_mode = False
    authenticated = False
    mode = 'A'
    startingPosition = 0
    isAppend = False
    # this is the port used by the server in passive mode to receive data from client
    passivePort = -1
    fileRenameFrom = None
    username = None
    passwd = None
    dataSocket = None
    passiveSocket = None

```

בנאי(Constructor)

המבנה מקבל גישה גלובלית `allThreads`.

מקבל בפרמטרים שלו את הסוקט להעברת פקודות ללקוח, את שם `thread` שמנצל בחיבור ללקוח.

ויצר `data member` חדש בשם `commandSocket` ומבצע לו השמה של הסוקט להעברת פקודות שהוא מקבל. בנוסף, ייצר `data member` `receiverAddress` נוסף בשם `clientAddress` לשינוי לsocket `.newCommandSocket`.

מוסיף את שם `thead`thread `allThreads` עם הערך `working` כדי שהשרת ידע שהחיבור ל`thead` זהה פעיל, מתחילה את `thead` לפי פונקציית `super` שלו.

פונקציות:

:run(1

```
def run(self):
    global isListening

    # when a client connects - send it a welcome message
    self.sendWelcome()

    while True:
        # if the server admin pressed q+Enter then close connection to the client and quit this thread
        # so the server can shut down properly
        if not isListening:
            self.QUIT('')
            break

        try:
            self.commandSocket.settimeout(5.0)
            data = self.commandSocket.receive(1024).rstrip()
            if (data is not None) and (len(data) > 0):
                # decode the received data as byte array and convert it (decode) into string using UTF8
                try:
                    cmd = data.decode('utf-8')
                    log("Data from client: " + cmd)
                except AttributeError:
                    cmd = data

                try:
                    # parse command and arguments from the data that we received from the client
                    cmd, arg = cmd[0:4].strip().upper(), cmd[4:].strip() or ''
                    # try to find a function that has the same name as the received command
                    func = getattr(self, cmd)
                    # execute the function with the received arguments
                    func(arg)
                except AttributeError as err:
                    self.sendCommand('500 Syntax error, command unrecognized.\r\n')
                    logCommand('Receive', err)
                except Exception as err:
                    logCommand("Error, unknown command from client: ", err)
                    self.sendCommand('500 could not interpret your command, please try again.\r\n')
                if not cmd:
                    break
            except socket.error as err:
                if err.__class__.__name__ != 'TimeoutError':
                    if 'forcibly closed' not in str(err):
                        logCommand('General Error while receiving data from client', err)
                    else:
                        break

        # once this thread run function has finished (got out of the while loop)
        # then log that the client has disconnected
        log("Client: " + str(self.clientAddress) + " disconnected")
```

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`. הפונקציה מקבלת גישה גלובלית ל-`isListening`. נשלחת הודעה `welcome` ללקוח. מתחילה לולאה אינסופית:

בודקים אם השרת לא מאזין ואם הוא אכן לא מאזין, מתבצעת קריאה לפונקציית `QUIT` עם פרמטר של סטרינג ריק על מנת לבבות את השרת כמו שצਰיך, לבסוף מבצעים `break` מהלולאה.

אם השרת עדיין מאזין, הפונקציה נכנסת למבנה קוד עטופה ב-`try` ו-`except`. מגדירים טימאאות של 5 שניות לסקוט של פקודות הלקוח.

אם לא מקבלת `data` מהלקוח לאחר 5 שניות נזרקת שגיאה.

מתתקבלת `data` בגודל של 1024 בתים דרך `commSock.recv` אשר מפושטים בלי רווחים באמצעות הפונקציה (`rstrip`).

נעשית בדיקה: אם `data` קיימת והאורך שלה מעל 0 אז נכניסים למבנה קוד נוספת שעטופה ב-`try` ו-`except`.

משתנה בשם `cmd` שambil את הפוקודה המתתקבלת מקבל את המידע וממיר אותו לסטRING ומדפיס אותו.

אם נזרקת שגיאה בגל שה-`data` לא מסוג `bytes` אז `cmd` שווה ל-`data` כמו שהוא. נכניסים למבנה קוד נוספת שעטופה ב-`try` ו-`except`.

המשתנים `cmd` (פקודה) ו-`arg` (Argument) מתקבלים מה-`cmd` שקדם לו כ-`bytes` סטרינג בכרך `cmd` מקבל את ה-4 `chars` הראשונים (באותיות גדולות), ו-`arg` מקבל את שאר `chars` החל מה-`char` החמישי ועד הסוף או שהוא מקבל סטרינג ריק.

לאחר מכן, באמצעות הפונקציה (`getattr`) מחפשים פונקציה שמ被执行ת על האובייקט `FtpServerProtocol` שיש לה את אותו שם שמתובל ב-`cmd`. (נועד כדי לקרוא לאחת מפקודות הпрוטוקול שמימושו). כל זה נכנס למשתנה בשם `func`.

לאחר מכן `func` מופעלת עם `arg` שהתקבל. (כל פוקודה בпротокол שמקבלת ארגומנט)

אם נזרקת שגיאה מסווג `AttributeError` נשלחת הודעה ללקוח שהפקודה לא מזוהה. אם נזרקת שגיאה כללית, מודפס שפקודה לא מזוהה התקבלה מהלקוח ונשלחת הודעה ללקוח שינסה שוב.

לאחר מכן נעשית בדיקה אם לא התקבל בכלל `cmd` ואם כן אז יוצאים מהלולאה.

אם נזקקת שגיאה בזמן קבלת `data`, היא נתפסת ומתרבצת בדיקה.
אם השגיאה אינה מסוג `TimeoutError class`, אז עושים עוד בדיקה לראות שה`error`
עצמו שהתקבל לא כותב `closed`. אם הוא לא כותב אז מודפס שיש שגיאה
כללית בזמן קבלת המידע וממשיכים בתוך הלולאה לקבלת `data`.
אחרת, יוצאים מהלולאה.

באשר פונקציית `recv` מסתיימת(ביציאה מהלולאה) וה`thread` סיים את פעולתו
מודפס שהלוקה הנוכחי(עם כתובת הקו והפורט) הtentak.

:getAbsolutePath(2

```
def getAbsolutePath(self, dirPath):
    log('getAbsolutePath(' + dirPath + ')')
    # if user did not supply a path then use empty string (so we actually use CWD)
    if not dirPath:
        dirPath = ""

    # if user path starts with / then get the absolute path
    if dirPath.startswith(os.path.sep):
        result = os.path.abspath(dirPath)
    else:
        # if user path is relative (does not start with /)
        # then join the CWD and user path then get the absolute path
        result = os.path.abspath(os.path.join(self.cwd, dirPath))

    log('getAbsolutePath() returning: ' + result)
    return result
```

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`, ומתקבל בפרמטר נתיב של ספרייה\קובץ.

בפונקציה הזאת אנחנו קודם כל מדפיסים את הנתיב שהתקבל.

אם לא סופק שם נתיב על ידי הלקוח `dirpath` יהיה `string` ריק.(לבן משתמש בתיקייה הנוכחית)

אם סופק שם נתיב על ידי הלקוח שמתחל ב`separator` של מערכת הפעלה "\\" אז `result` יהיה שווה לנטיב האבסולוטי, לפי פונקציה `os.abspath()`. (לדוגמא:

יסופק `C:\temp` ---> (`result = C:\temp`)

אחרת, אם סופק נתיב על ידי הלקוח שאינו מתחל ב`separator` (משמעות נתיב לטיבי\יחסי) אז מחברים את `dirpath` לattrib `data member cwd` בשם `cwd`(שמחזיק בתיקייה הנוכחית) באמצעות הפונקציה (`os.path.join()`).

הפונקציה מחברת את המשתנים הללו עם "\" ביניהם וນוצר נתיב.

על זה מפעילים את הפונקציה `os.abspath()` על מנת לקבל את הנתיב האבסולוטי שיוכנס לתוך המשתנה `result`.

(לדוגמא: יסופק `temp` והוא `cwd=T`, וכן יצא `cwd=C:\T` והנתיב האבסולוטי יתנו `C:\T\ cwd`).

לאחר מכן, תבוצע הדפסה של הנתיב שהוגדר והוא יוחזר.

:openSocket(3

```
def openSocket(self):
    log('openSocket()')
    # check if user is authenticated
    self.isUserAuthenticated()

    if self.pasv_mode:
        # since client asked us to work in passive mode (FTP server launch a socket, and client connect to it)
        # instead of client launching a socket and sever connect to it
        log("openSocket(): waiting for client to connect in passive mode")
        self.dataSocket = self.passiveSocket.accept()
        log("openSocket(): connected to client in passive mode")
        self.dataSocketIP = self.dataSocket.receiverAddress[0]
        self.dataSocketPort = self.dataSocket.receiverAddress[1]
        log(f"openSocket(): dataSocketIP: {self.dataSocketIP} dataSocketPort: {self.dataSocketPort}")
    else:
        # create an outgoing connection socket
        if isTCPIP:
            self.dataSocket = TCPIPSocket()
        else:
            self.dataSocket = RUDPSocket()
        # connect to the client IP and port using the created socket
        dataSocketAddress = (self.dataSocketIP, self.dataSocketPort)
        self.dataSocket.connect(dataSocketAddress)
        log("openSocket(): connected to client in active mode")
        log(f"dataSocket IP: {self.dataSocketIP} Port: {self.dataSocketPort}")
        log(f"openSocket(): dataSocketIP: {self.dataSocketIP} dataSocketPort: {self.dataSocketPort}")
```

פונקציה זאת מבוצעת על האובייקט `.FtpServerProtocol`

פונקציה לפיתוח סוקט כלשהו(סוקט שרת, סוקט העברת מידע בין שרת ללקוח).

בפונקציה הזאת אנחנו קודם כל מדפים שמתחליל תהליך פתיחת סוקט.

לאחר מכן אנחנו צריכים לבדוק אם המשתמש עבר אוטנטיקציה בחיבור לשרת, אם לא אז נדרקת שגיאה ומתקבלת הודעה שהלkoח צריך להתחבר.

לאחר מכן, חלה בדיקה אם החיבור בין השרת ללקוח במצב פאסיבי או לא.

אם הוא במצב פאסיבי, השרת יוצר את החיבור עם הלkoח דרך **הסוקט של השרת**.

השרת יוצר סוקט פאסיבי בעזרת `accept()` ומשיר ל-`dataSocket`, כאשר החיבור נוצר עם הלkoח, הכתובות זו של הסוקט מתקבלת מהמקום הראשון בarray של `receiverAddress`.

אם השרת לא במצב פאסיבי, תבוצע בדיקה:

אם `isTCPIP` הוא `True` אז יפתח סוקט TCP, אחרת יפתח סוקט UDP

לכן הלקוח הוא זה שיציר את החיבור עם השרת ומציין בתובת `data` ופורט שהשרת צריך להתחבר אליהם על מנת לקבל `data` מהלקוח או לשלוח `data` ללקוח.

ונוצר `update` אשר מחזיק בכתובת הקו ופורסם של הלקוח.

הtuple הזה מועבר כפרמטר לפונקציית `connect()` אשר מתחילה חיבור בין הלקוח לשרת בעזרת הסקט `.dataSocket`.

:closeSocket(4)

```
def closeSocket(self):
    log('closeSocket()')
    try:
        # if there is an open data socket then close it
        if self.dataSocket is not None:
            self.dataSocket.close()

        # if there is an open server socket then close it
        if self.passiveSocket is not None:
            self.passiveSocket.close()
            returnPortToPool(self.passivePort)

    except socket.error as err:
        logCommand('closeSocket has failed', err)
```

פונקציה זאת מבוצעת על האובייקט `.FtpServerProtocol`.

פונקציה לסגירת סוקט בלשחו(סקט שרת, סוקט העברת מידע בין שרת ללקוח).

בפונקציה הזאת אנחנו קודם כל מדפסים שמתחליל תהליך סגירת סוקט.

התהליך הזה עטוף בזאת `try and except` על מנת טיפול בשיגיאות באשר קיימות.

בדיקה ראשונית היא במקרה שהסקט להעברת המידע(אם קיים) בין הלקוח לשרת הוא לא `None` ובהתאם לסגור אותו באמצעות הפונקציה `(.close)`.

בדיקה שנייה היא במקרה שהשרת של הסקט(`passive mode`) הוא לא `None` ובהתאם לסגור אותו באמצעות הפונקציה `(.close)`, ולאחר מכן באמצעות הפונקציה `returnPortToPool` להחזיר את הפורט של השרת לתוך `dictionary` אשר מכיל פורטים מסוימים ונונן מייד עליהם האם הם תפוסים או משוחררים.

לאחר הפעלת הפונקציה הזאת, הפורט של השרת משוחרר וניתן להשתמש בו שוב.

במידה ונדרקת שגיאה במהלך התהילה, היא נתפסת ומודפס שסגירת הסוקט נבשלה
ובנוסף אליו שגיאה זאת. (תחת שגיאה מסווג `socket.error`)

:sendCommand(5

```
def sendCommand(self, cmd):
    # encode the cmd string into byte array
    sentLength = self.commandSocket.send(cmd.encode('utf-8'))
    return sentLength
```

פונקציה זאת מבוצעת על האובייקט `.FtpServerProtocol` פונקציה פשוטה אשר מקבלת פקודה כלשהי מסוג של `String` ושולחת אותה ללקוח דרך סוקט של פקודות בשם `.commSock`.
הfonקציה (`encode()` ממיר את `String` המתתקבל בפקודה לזרם של בתים באמצעות קידוד utf-8 אשר נמצא בשימוש רב ומבטיח שהפענוח של זרם הבטים יבוצע באופן על ידי הלקוח. הפונקציה גם מחזירה את הפקודה עצמה.

:sendData(6

```
def sendData(self, data):
    # send data on the socket as byte array
    self.dataSocket.send(data)
```

פונקציה זאת מבוצעת על האובייקט `.FtpServerProtocol` פונקציה פשוטה אשר מקבלת מידע מיידע כלשהו ושולחת אותו ללקוח על גבי סוקט של העברת מידע בין השרת ללקוח בשם `.dataSock`.
הfonקציה שולחת את המידע דרך הסוקט במערך של בתים.

:OPTS(7

```
def OPTS(self, onOff):
    log("OPTS(" + onOff + ")")
    self.sendCommand('202 UTF8 mode is always enabled. No need to send this command\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הפקודה הזאת משמשת לציין אופציה נוספת להפעלת שנן הנוכחי של FTP.
במימושים שונים של ה프וטוקול לפקודה הזאת קיימים כמה אופציות ביןיהם:
MODE, UTF8, EPRT, EPSV, EPRT וко.

בפונקציה הזאת בחרנו למשה שאופציה היחידה הקיימת היא UTF8 אשר
מאפשרת שימוש בקידוד של 8-utf. בעצם השימוש שלנו הוא שהפונקציה הזאת לא
יכולת לשנות את סוג האופציה והוא תמיד UTF8.

לכן אנחנו מדפיסים שSTS מופעל או לא לפני הפעלה שהפונקציה מקבלת
ושולחים הודעה ללקוח דרך הסוקט `commSock` שUTF8 תמיד מופעל, ואין צורך
להשתמש בפקודה הזאת מפני שאנחנו מגבילים אותה לתפקיד אחד בלבד.

פונקציה זאת תופעל באופן אוטומטי בעת חיבור של לקוח לשרת.

:AUTH(8

```
def AUTH(self, user):
    log("AUTH(" + user + ")")
    self.sendCommand('500 Insecure server, it does not support FTP over TLS/SSL.\r\n')
```

פונקציה של אוחת הפקודות בפרוטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הפונקציה הזאת מטפלת בפקודה AUTH אשר מטפלת באבטוח השרת אך בחרנו
לא למש ולכן רק נודיעו ללקוח שאנחנו לא תומכים בפקודה בזאת.

:USER(9

```
def USER(self, user):
    log("USER(" + user + ")")

    # if no user has been supplied - return error to the client
    if not user:
        self.sendCommand('501 Missing required argument.\r\n')
    else:
        # if the correct user has been supplied then set it to the username member
        if user == DEFAULT_USER:
            self.username = user
        else:
            # if the wrong user has been supplied then reset the username member
            self.username = None

    # for security reasons always ask for a password (so a hacker cannot know the real usernames of this server)
    self.sendCommand('331 Please, specify the password.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הפקודה הזאת משמשת לחיבור ויזיהו של הלוקום עם השרת.

קודם כל יודפס שהפקודה `USER` מופעלת עם השם של `user` שמנסה להתחבר.

בבדיקה ראשונית היא כדי לבדוק אם פרמטר נשלח בבקשתו של שם `user`.

אם לא, נשלחת הודעה לлокום שחסר ארגומנט.

אם כן, קיימת עוד בדיקה בנוגע לזהות `user`.

אם `user` שמתתקבל שווה לשם `user` שהגדכנו קבוע מראש מראש עד מתבצעת השמה `username` בשם `data member` של האובייקט.

אם `user` שמתתקבל אינו שווה לשם שהוגדר מראש מראש עד מתבצעת השמה של `None` `data member`. במקרה זה לא מתבצעת שליחת הודעה לлокום לשם `user` משתמש שאינו קיים במערכת מצורכי בטיחות(שהאקר לא ידע אילו `users` קיימים במערכת).

לכן מתבצעת שליחת הודעה לлокום שהשרת מבקש לקבל את הסיסמה.

:PASS(10

```
def PASS(self, passwd):
    log("PASS(" + passwd + ")")

    # if a password is empty or not the correct password, or the username is wrong then return error to the client
    if (not passwd) or (self.username != DEFAULT_USER) or (passwd != DEFAULT_PASSWORD):
        # reset user and password so next attempts will start from scratch
        self.username = None
        self.passwd = None
        self.sendCommand('530 Login incorrect.\r\n')

    else:
        # if the user is the correct user, and the password is
        # the correct password then mark client as authenticated
        self.passwd = passwd
        self.sendCommand('230 Login successful.\r\n')
        self.authenticated = True
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הפקודה הזאת משמשת להמשך חיבור וזיהוי של הלוקו עם השרת.

קודם כל יודפס שהפקודה PASS מופעלת עם הסיסמה שניתנה על ידי `user` שמנסה להתחבר.

מתבצעת בדיקה אשר אומרת שם לא סופקה סיסמא, או שם `user` שסופק לא שווה לשם שהוגדר מראש (עדין יכול חלק של בקשת הסיסמה בגל בטיחות), או שהסיסמה שסופקה אינה תואמת לסיסמה שהוגדרה מראש, אך ה`members` של האובייקט (סיסמה ושם משתמש) יוגדר `None` ותישלח הודעה לлокו שהחיבור לא בוצע באופן נכון.

אחרת, אם הסיסמה והשם משתמש הם הנכונים אז ה`members` של האובייקט (סיסמה ושם משתמש) יקבלו את ההשמה של הפרמטרים שסופקו ותישלח הודעה לлокו שהחיבור צלח. כמו כן, ה`members` של האותנטיקציה יוגדר `True` לאור הצלחת החיבור בין השרת לлокו.

:isUserAuthenticated(11)

```
def isUserAuthenticated(self):
    # check if user is loggedon (authenticated), if not return error and ask to login
    if not self.authenticated:
        self.sendCommand('530 Please log in with USER and PASS first.\r\n')
        log("User not authenticated, please login first")
        raise UserNotAuthenticatedException('User not loggedin')
```

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
הfonקציה זהה בודקת שה`user` שמנסה לבצע פקודות כלשהן מצד הלוקו הוא אכן מזוהה, מאושר ועבר חיבור בהצלחה. פונקציה זאת נועדה לוודא שבכל פקודה או עד שה`user` מבצע לאורך הרתקשות עם השרת, הוא מורשה לעשות זאת.

נעשית בדיקה האם `user` לא עבר אוטנטיקציה(`data member` בשם `authenticated` מוגדר כ`false`), אז נשלחת הودעה ללוקו שהוא צריך להתחבר עם שם משתמש וסיסמה, מודפסת הודעה שה`user` לא עבר אוטנטיקציה וצריך להתחבר.

כמו כן, נזרקת שגיאה על ידי `class` שבנוינו(`UserNotAuthenticatedException`), שאומר שה`user` אינו מחובר.

:EPRT(12

```
def EPRT(self, args):  
    self.PORT(args)
```

פונקציה של אחת הפקודות ב프וטוקול FTP.
פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
הfonקציה הזאת דומה מאד לפונקציית `PORT` והוא מודרנית יותר ממנה.
היתרון שלה על `PORT` הוא התמיהה בIPv4 בלבד בנוסף לIPv4.
בחרנו למש רק לבתוות IPv4 וכן הfonקציה הזאת רק משמשת כalias
לfonקציית `PORT` וקוראת לה.(קיימת בغالל הדרישה בחלק מערכות הפעלה
לפקודה בשם `EPRT` ולא `PORT`)

:PORT(13

```
def PORT(self, args):
    log("PORT(" + args + ")")

    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        if self.pasv_mode:
            self.passiveSocket.close()
            self.pasv_mode = False

        # convert arguments into array (split by ,)
        ipAndPortArray = args.split(',')

        # get the first 4 parts of the array and join them with . into an ip address
        self.dataSocketIP = '.'.join(ipAndPortArray[:4])

        # get the last 2 array entries and calculate the client port
        # (bit shift left the first part, and add to it the second part)
        self.dataSocketPort = (int(ipAndPortArray[4]) << 8) + int(ipAndPortArray[5])

        # return message to the client that the port configuration has succeeded
        self.sendCommand('200 PORT command successful.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("Port function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.
פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
הfonקציה הזאת משמשת כדי לפתח חיבור בין לקוח לשרת על מנת להעביר קבצים.
הפקודה PORT מפרטת את כתובות הקו + פורט הלקוח, כדי שהשרת ידע
לאן להתחבר כדי לשלוח/לקבל data.
דוגמה: 192.168.1.2,7,138 גורר שהלקוח מציין כתובות 1.2
ובפורט 1802 (כפי הנוסחה למציאת פורט היא $1802 = (7 * 256) + 138$)

הfonקציה מקבלת ארגומנט בשם args שהוא string כמו בדוגמה המבטה
כתובות קו ופורט data של הלקוח המוצג בפורמט `h1,h2,h3,h4,p1,p2` כאשר
`h1,2,3,4` מייצגים את כתובות הקו ו`p1,2` מייצגים את הפורט.

הfonקציה מדפסת את args, ולאחר מכן נכנסת לתבנית קוד עטופה בtry וexcept.

נעשית בדיקה אם `theUser` עבר אוטנטיקציה. לאחר מכן נעשית בדיקה אם החיבור במצב פאסיבי לפי המEMBER `data` שמייצג בשם `pasv_mode`.

אם הוא במצב פאסיבי יש לסגור את הסוקט של השירות ולשנות את `pasv_mode` ל`false`. (כפי בפקודה PORT הלקוח יוצר את הקשר ולא השירות)

לאחר מכן, ה`string` בשם `args` מפוצל למערך לפי ", "(כל הנראה למערך בגודל 6).

הערך `dataSockAddr` בשם `data` מקבל את 4 המיקומות הראשונים למערך מופרדים בעזרת ". על מנת לקבל את בתובת הקו.

הערך `dataSockPort` בשם `data` מקבל את הפורט על ידי הנוסחה שבדוגמה:

המקום החמישי למערך מומר ל`atoi` ומבצעת עליו פעולה `bitwise shift-left` של 8 פעמים (שקלול להכפלה של 2 בחזקת 8) על מנת להכפיל את הערך ב-256.

המקום השישי למערך מומר ל`atoi` ומחושב בסכום, ומתקבל מספר הפורט.

לאחר מכן נשלחת הودעה ללקוח שהפקודה צלחה.

אם נדרקת שגיאה בזמן הפקודה, היא נתפסת ומתבצעת בדיקה.

אם השגיאה אינה מסוג `class` שבנוינו (`chSionUserNotAuthenticatedException`), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה (שתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:NLST(14

```
def NLST(self, dirpath):  
    self.LIST(dirpath)
```

פונקציה של אחת הפקודות ב프וטוקול FTP.
פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
הfonקציה הזאת שונה מfonקציית LIST והוא פחות שימושית ממנה.
הfonקציה הזאת רק מספקת את שמות הקבצים ללקוח ולא פרטים נוספים עליהם.
בחרנו למש שבאשר הלקוח רוצה לראות את רשימת הקבצים\ספריות הוא יקבל
יותר פרטים (כמו גודל,הרשאות,תאריךינו) ולא רק את השמות.
לכן הfonקציה הזאת רק משמשת כalias לfonקציית LIST וקוראת לה. (קיימת בגל
הדרישה בחלק מערכות הפעלה לפקודה בשם NLST ולא LIST)

:LIST(15

```
def LIST(self, dirpath):
    log("LIST(" + dirpath + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        pathname = self.getAbsolutePath(dirpath)

        if not os.path.exists(pathname):
            # file or folder does not exist - return error to the client
            self.sendCommand("550 Couldn't open the file or directory.\r\n")
        else:
            # send to client that we have received the request and starting to work on it
            self.sendCommand("150 Starting data transfer.\r\n")

            # open socket connection to client on the address and port he has set using the previous PORT command
            self.openSocket()

            # if the user asked to list a file (not a directory) then get
            # file properties and return them on the previously opened socket
            if not os.path.isdir(pathname):
                # get file properties (change date / size / owner...)
                fileMessage = fileProperty(pathname)
                # send data to client on data socket as byte array
                # (inside the function it will decide if to send text or binary byte array)
                fileMessageByteArray = bytes(fileMessage + '\r\n', encoding="utf-8")
                self.sendData(fileMessageByteArray)

            else:
                # if this is a directory (not a file) then loop through the directory
                # files and folders and write their properties to the previously opened socket
                for file in os.listdir(pathname):
                    # get file/folder properties (change date / size / owner...)
                    fileMessage = fileProperty(os.path.join(pathname, file))
                    # send data to client on data socket as byte array
                    # (inside the function it will decide if to send text or binary byte array)
                    fileMessageByteArray = bytes(fileMessage + '\r\n', encoding="utf-8")
                    self.sendData(fileMessageByteArray)

            # at the end close the previously opened socket
            self.closeSocket()

            # send success message to the client
            self.sendCommand('226 Operation successful.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("LIST function failed", err)
            self.closeSocket()
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonkzia מציiga את הקבצים והספריות בנטייב נתון, لكن מקבלת כפרמטר נתיב רצוי `.dirpath`.

הfonkzia מדפיסה את `dirpath`, ולאחר מכן נכנסת למבנה קוד עטופה ב`try` ו`except`.

נעשית בדיקה אם `user` עבר אונטנטיקציה.

נעשית קרייה לפונקציה `getAbsolutePath` שמקבלת את `dirpath` כפרמטר ומחזירה את הנתיב האבסולוטי הרצוי `pathname`.

תבצע בדיקה: אם הנתיב אינו קיים תישלח הודעה ללוקו שלא ניתן לפתח את התקינה או הקובץ הנ"ל. אם קיים אז תישלח הודעה שמתחליה העברה של נתונים.

לאחר מכן יפתח סוקט בין השרת ללוקו בעזרת `()openSocket()` עם בתובת קו ופורט שהתקבל מהлокו על ידי הפקודה `PORT`.

לאחר מכן תבוצע עוד בדיקה: אם הנתיב הוא של קובץ ולא תקין, אז ישלחו הפרטים של אותו קובץ(הרשאות,שעת שינוי,שם וכו') ללוקו.

אם הנתיב הוא של תקין, אז נעביר בלולאה על כל הקבצים\תיקיות באותו נתיב ונסלח את הפרטים של כלום(הרשאות,שעת שינוי,שם וכו') ללוקו.

לאחר מכן יסגר הסוקט ותישלח הודעה ללוקו שהפעולה הצלחה.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתרוצעת בבדיקה.

אם השגיאה אינה מסוג `class` שבנו (`UserNotAuthenticatedException`), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האונטנטיקציה(שמרתחשת בהתחלה וזרוקת שגיאה ספציפית) וכן תודפס סוג השגיאה, יסגר הסוקט ותישלח הודעה ללוקו שהפקודה נכשלה.

:XCWD(16

```
def XCWD(self, cmd):  
    self.CWD(cmd)
```

fonkzia shel achat hakodot beprutokol FTP.

fonkzia zat mbotza'et ul avivik FtpServerProtocol.
fonkzia ha'ezat domha l'fonkzia CWD.

l'kn fonkzia ha'ezat rak meshmat basia l'fonkzia CWD kura'at la. (kiymat
bagel hadrisha bchuk mmurbot ha'hafuta l'pko'ah b'shem CWD vla XCWD)

:CWD(17)

```
def CWD(self, dirpath):
    log("CWD(" + dirpath + ")")

    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        pathname = self.getAbsolutePath(dirpath)

        # convert the absolute path into ftp server absolute path
        pathname = getFTPPPath(pathname)

        # if dirpath is not a directory or dirpath does not exist then return an error
        if not os.path.exists(pathname) or not os.path.isdir(pathname):
            self.sendCommand('550 CWD failed Directory not exists.\r\n')
        else:
            # set CWD member to the received dirpath and return success message
            self.cwd = pathname
            self.sendCommand('250 CWD Command successful.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("CWD function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה משנה את התקינה הנוכחית לתקינה רצiosa, לבן מקבלת כפרמטר `.dirpath`

הfonקציה מדפיסה את `dirpath`, ולאחר מכן נכנסת לבניית קוד עטופה ב`try` ו`except`.

נעשית בדיקה אם `user` עבר אונטיקציה.

נעשית קרייה לפונקציה `getAbsolutePath` שמקבלת את `dirpath` כפרמטר ומחזירה את הנתיב האבסולוטי הרצוי `.pathname`.

לאחר מכן `pathname` שווה להפעלת הפונקציה `getFTPPPath` על `pathname`. (מוסבר על הפונקציה בחלק של `Utils`)

לאחר מכן מתבצעת בדיקה נוספת: אם אין נתיב צזה או שהנתיב זהה איןנו ספרייה מוחזרת הודעה בהתאם ללקוח.

אחרת, הערך `data member` בשם `pwd`(התיקייה הנוכחית) מקבל את הערך של `pathname` ונשלחת הודעה ללקוח שהפקודה צלחה.

אם נזרקת שגיאה בזמן הפקודה, היא נפתחת ומתבצעת בדיקה.

אם השגיאה אינה מסוג ה`class` שבנוינו (`UserNotAuthenticatedException`), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שמתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:XPWD(18)

```
def XPWD(self, cmd):  
    self.PWD(cmd)
```

פונקציה של אחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `.FtpServerProtocol`.

הfonקציה הזאת שונה מפונקציית `PWD` והוא פחות שימושית ממנה.

הפונקציה הזאת מציגה את התיקייה הנוכחיית כמו `PWD` אבל בנוסף מספקת מידע כמו הרשותות ואת יוצר התיקייה.

בחרנו למשם שהfonקציה הזאת רק תציג את שם התיקייה הנוכחיית.

לכן הפונקציה הזאת רק משתמש כ`alias` לפונקציית `PWD` וקוראת לה.(קיימת בגל הדרישת חלק מערכות הפעלה לפקודה בשם `XPWD` ולא `PWD`)

:PWD(19

```
def PWD(self, cmd):
    log("PWD(" + cmd + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        self.sendCommand('257 "%s".\r\n' % self.cwd)
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("PWD function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה מציגה את התיקייה הנוכחית ומקבלת פקודה(`cmd`) בלבד.

הfonקציה מדפסה שהיא מופעלת, ולאחר מכן נכנסת למבנה קוד עטופה בזע.

`.except`

נעשית בדיקה אם `user` עבר אוטנטיקציה.

נשלחת הודעה ללקוח עם התיקייה הנוכחית שמוגדרת לפי הערך של ה `data` `member` בשם `cwd`.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומבצעת בדיקה.

אם השגיאה אינה מסוג `class` שבנו (`UserNotAuthenticatedException`), אז

אנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה

האוטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) וכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:TYPE(20

```
def TYPE(self, type):
    log("TYPE(" + type + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # if client sent i or I then set transfer mode to binary
        if type.upper() == 'I':
            self.mode = 'I'
            self.sendCommand('200 Binary mode.\r\n')

        # if client sent a or A then set transfer mode to Ascii
        elif type.upper() == 'A':
            self.mode = 'A'
            self.sendCommand('200 Ascii mode.\r\n')

        # if client sent type other then I or A then return an error response
        else:
            self.sendCommand(type + ': unknown mode.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("TYPE function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אחת הפקודות בפרוטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה מגדרת את אופן העברת הנתונים, בינארית או אסקריינט.

הfonקציה מדפיסה את סוג העברת הנתונים (בינארית או אסקריינט), ולאחר מכן נכנסת לתבנית קוד עטופה ב `try` ו- `except`.

נעשה בדיקה אם `user` עבר אונטנטיקציה.

אם ה `type` שהוגדר על ידי `user` שווה ל "I", אז ה `data member` בשם `mode` מוגדר ל "I" ונסלחת הודעה ללקוח שהמוד הוא בינארית.

אם ה `type` שהוגדר על ידי `user` שווה ל "A", אז ה `data member` בשם `mode` מוגדר ל "A" ונסלחת הודעה ללקוח שהמוד הוא אסקריינט.

אחרת, נשלחת הודעה ללקוח שהמוד אינו מוכר.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתרכעת בדיקה.
אם השגיאה אינה מסוג `class` שבינו (`UserNotAuthenticatedException`), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה
הອוטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג
השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:PASV(21

```
def PASV(self, cmd):
    log("PASV(" + cmd + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # mark passive mode flag to true
        self.pasv_mode = True

        # create a new server socket based on the selected protocol
        if isTCPIP:
            self.passiveSocket = TCPIPSocket()
        else:
            self.passiveSocket = RUDPSocket()

        # bind server socket to the server IP and unique port number (so each client can have a unique port number)
        self.passivePort = getPortFromPool()
        self.passiveSocket.listen((SERVER_HOST, self.passivePort))

        # send the client that we entered a passive mode, with the socket info
        self.sendCommand('227 Entering Passive Mode (%s,%u,%u).\\r\\n' %
                         ('.'.join(SERVER_HOST.split('.')), self.passivePort >> 8 & 0xFF, self.passivePort & 0xFF))
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("PASV function failed", err)
            self.sendCommand('500 Operation Failed.\\r\\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.
פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
הfonktsia מבנינה את השרת למצב פאסיבי ומקבלת פקודה(`cmd`) בלבד.
הfonktsia מדפיסה שהיא מופעלת, ולאחר מכן נכנסת לתבנית קוד עטופה בזען
.except.
נעשית בדיקה אם `user` עבר אוטנטיקציה.
הfonktsia בשם `pasv_mode` member מוגדר ל`True`.
נעשית בדיקה: אם `isTCPIP` הוא `True` אז הsocket הfasivyi מסוג TCP, אחרת הוא
מסוג RUDP.

בעשית הגדרה של `data member` בשם `passivePort` אשר מייצג את הפורט של השירות הפסיבי, על ידי פונקציה שבניינו בשם `getPortFromPool` אשר מספקת פורט ייחודי לכל קליננט שמתחבר לשרת.

לאחר מכן מוחזקים את `passivePort` ואת המשתנה `SERVER_HOST` tupple בטור שנקננס בfrmater לפונקציית `(bind()` אשר משייכת את הנתונים הלו(פורט וכתובת קו) לסקוט של השירות. כמו כן, השירות יכול להאזין לחיבור 1 בו זמנית.

לאחר מכן הפונקציה `(getsockname()` שמופעלת על הסקוט של השירות מוחזירה שני ערכים – את כתובת הקו שלו ואת הפורט ומבצעת השמה למשתנים `port`, `addr`, `port` בהתאם.

לאחר מכן נשלחת הודעה ללקוח שהשרת נבנה במצב פאסיבי ומפורטים בכתובת הקו והפורט. זה בעצם החישוב ההופיע לפקודת `PORT`.

הביטוי מחושב על ידי ביצוע `shift-right` ב-8 על הפורט וריבבת `mask` של `0xFF`, וריבבת `mask` של `0x00` על הפורט בלבד.

דוגמה:
`addr = "127.0.0.1"`
`port = 2675`

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתבצעת בדיקה.

אם השגיאה אינה מסווג כlass שבניינו (`UserNotAuthenticatedException`), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:SYST(22

```
def SYST(self, arg):
    logCommand('SYS', arg)
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        self.sendCommand('215 %s type.\r\n' % sys.platform)
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("SYST function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

fonkzia של אוחת הפקודות ב프וטוקול FTP.

fonkzia זאת מבוצעת על האובייקט FtpServerProtocol.

הfonkzia מחרירה הودעה ללקוח שמספרת את מערכת הפעלה של השרת.(וינדוס,לינוקס וכו..)

הfonkzia מקבלת ארגומנט(arg) בלבדו.

הfonkzia מדפיסה שהיא מופעלת, ולאחר מכן נכנסת לתבנית קוד עטופה בtry .except

נעשה בדיקה אם user עבר אונטנטיקציה.

נשלחת הודעה ללקוח על איזו מערכת הפעלה השרת פועל באמצעות sys.platform.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתבצעת בדיקה.

אם השגיאה אינה מסוג המוגדר בפוקודה (UserNotAuthenticatedException), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האונטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:XCUP(23)

```
def XCUP(self, cmd):
    self.CDUP(cmd)
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.
 פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
 הפונקציה הזאת דומה לפונקציית `CDUP`.
 لكن הפונקציה הזאת רק משתמשת כalias לפונקציית `CDUP` וקוראת לה. (קיימת
 בכלל הדרישה בחלק ממערכות הפעלה לפקודה בשם XCUP ולא CDUP)

:CDUP(24)

```
def CDUP(self, cmd):
    log('CDUP()')
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # set the CWD to its parent folder (add .. to the path)
        self.cwd = os.path.abspath(os.path.join(self.cwd, '..'))
        # return success message to the client
        self.sendCommand('250 CDUP command successful.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("CDUP function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות בפרוטוקול FTP.
 פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.
 הפונקציה משנה את התקינה הנוכחיות לתיקיות האב שלה ומתקבלת פקודה(`cmd`)
 כלשהי.
 הפונקציה מדפסה שהיא מופעלת, ולאחר מכן נכנסת לתבנית קוד עטופה ב`try`
`.except`.
 נעשית בדיקה אם `user` עבר אונטנטיקציה.

השיינט member data בשם pwd(תיקייה נוכחית) מקבל את הנתיב האבסולוטי של תיקייה האב של התיקייה הנוכחית בעזרת הפקנץיה join.os שմכרפת את התיקייה הנוכחית עם ".." שמסמל הגעה לתיקיית האב.

נשלחת הודעה ללקוח שהפקודה צלחה.

אם נזרקת שגיאה בזמן הפקודה, היא נتفسת ומתבצעת בדיקה.

אם השגיאה אינה מסוג ה- class UserNotAuthenticatedException שבנוינו (UserNotAuthenticatedException), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:DELE(25

```
def DELE(self, filename):
    log("DELE(" + filename + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        pathname = self.getAbsolutePath(filename)

        # if the file or folder does not exist the return error message to the client
        if not os.path.exists(pathname):
            self.sendCommand('550 Failed to delete file: %s, file does not exists.\r\n' % pathname)

        # if user is not allowed to delete files and folders from the server then return an error
        elif not allow_delete:
            self.sendCommand('450 Failed to delete file: %s, server does not allow delete.\r\n' % pathname)

        # if the user is allowed to delete files and folders and the file/folder exist - then delete it
        else:
            os.remove(pathname)
            self.sendCommand('250 File deleted.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("DELE function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה מוחקמת תקייה או קובץ מהשרת וכן מקבלת `filename` כפרמטר.

הfonקציה מדפסה את שם הקובץ, ולאחר מכן נכנסת למבנה קוד עטופה ב`try`.
`except`

נעשית בדיקה אם `user` עבר אונטנטיקציה.

נעשית קרייה לפונקציה `getAbsolutePath` שמקבלת את `filename` כפרמטר
ומחזירה את הנתיב האבסולומי הרצוי `pathname`.

תבצע בדיקה: אם הנתיב אינו קיים תישלח הודעה ללקוח שלא ניתן למחוק את
התקייה או הקובץ הנ"ל.

אם `user` לא מושחה למחוק תיקיות\קבצים מהשרת אז תישלח הודעה של שגיאה.

(ניתן לדעת זאת לפי `data member` בשם `allow_delete`).

אחרת, התיקייה\קובץ נמחקים לפי הפונקציה `os.remove()`, ותישלח הודעה ללקוח
שהפעולה הצלירה.

אם נזרקת שגיאה בזמן הפקודה, היא נפתחת ומתבצעת בדיקה.

אם השגיאה אינה מסוג ה `class UserNotAuthenticatedException`, אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שמרתחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:XMKD(26)

```
def XMKD(self, dirname):
    self.MKD(dirname)
```

פונקציה של אחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה הזאת דומה לפונקציית MKD למעט שינוי קטן שחוות אש מסויימות מניחות `characters` ftp כריכים להיות באורך של 4 אותיות. לכן בmimeosh שלנו הפונקציות אותו דבר.

הfonקציה הזאת יוצרת תיקייה בתיקייה הנוכחיות, מקבלת כפרמטר את שם התיקייה(`dirname`).

לכן הפונקציה הזאת רק משמשת כalias לפונקציית `MKD` וקוראת לה.(קיימת בגל הדרישה בחלק מערכות הפעלה לפקודה בשם XMKD ולא MKD)

:MKD(27)

```
def MKD(self, dirname):
    log("MKD(" + dirname + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        pathname = self.getAbsolutePath(dirname)

        # if the directory that we try to create already exist then return error message
        if os.path.exists(pathname):
            self.sendCommand('550 MKD failed, directory "%s" already exists.\r\n' % pathname)
        else:
            # create the directory at the current working directory
            os.mkdir(pathname)
            self.sendCommand('257 Directory created.\r\n')

    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("MKD function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אחת הפקודות ב프וטוקול FTP.

.FtpServerProtocol האובייקט על מבוצעת פונקציה.

הfonkziah הzzat יוצרת תיקייה בתיקייה הנוכחית, מקבלת כפרמטר את שם התיקייה dirname().

הfonkziah מדפסת את שם התיקייה, ולאחר מכן נכנסת לבניית קוד עטופה ב try .excepti

ונשיית בדיקה אם user עבר אונטנטיקציה.

בעשית קרייה לפונקציה getAbsolutePath שמקבלת את filename כפרמטר ומחזירה את הנתיב האבסולוטי הרצוי pathname .

תתבצע בדיקה: אם התיקייה שמנשים ליצורobar קיימת תישלח הודעה ללקוח שהפעולה נכשלה ושהתיקייהobar קיימת.

אחרת, התיקייה נוצרת לפי הפונקציה () os.mkdir , ותישלח הודעה ללקוח שהתיקייה נוצרה.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתרכעת בבדיקה.

אם השגיאה אינה מסוג ה class שנבינו (UserNotAuthenticatedException) , אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האונטנטיקציה (שתרחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:XRMD(28

```
def XRMD(self, dirname):
    self.RMD(dirname)
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה הזאת דומה לפונקציית `RMD` למעט שינוי קטן שחוות אש מסויימת מניות שמשתמש `ftp characters` צריכים להיות באורך של 4 אותיות. לכן בימוש שלנו הפונקציות אותו דבר.

הfonקציה הזאת מוחקת תיקיה מהשרת ואם לתיקיה הזאת יש עוד תיקיות מתחתייה אז היא מוחקת גם אותן. לכן הפונקציה מקבלת את שם התיקייה כפרמטר.

לכן הפונקציה הזאת רק משתמשת כalias לפונקציית `RMD` וקוראת לה.(קיימת בגל הדרישת בחלק מערכות הפעלה לפקודה בשם `XRMD` ולא `RMD`)

:RMD(29

```
def RMD(self, dirname):
    log("RMD(" + dirname + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        pathname = self.getAbsolutePath(dirname)

        # if the directory that we try to delete doesn't exist then return error message
        if not os.path.exists(pathname):
            self.sendCommand('550 RMD failed, directory "%s" does not exists.\r\n' % pathname)

        # if user is not allowed to delete files and folders from the server then return an error
        elif not allow_delete:
            self.sendCommand('450 Failed to delete folder: %s, server does not allow delete.\r\n' % pathname)

        # remove the directory that we received
        else:
            shutil.rmtree(pathname)
            self.sendCommand('250 Directory deleted.\r\n')

    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("RMD function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonktsiya זאת מוחקת תיקייה מהשרת ואם לתקייה הזאת יש עוד תיקיות מתחתייה אז היא מוחקת גם אותם. لكن הפונקציה מקבלת את שם התקייה כפרמטר.

הfonktsiya מדפסה את שם התקiya, ולאחר מכן נכנסת לבניית קוד עטופה ב`try` ו`except`.

נעשה בדיקה אם `theUser` עבר אוטנטיקציה.

נעשה קרייה לפונקציה `getAbsolutePath` שמקבלת את `filename` כפרמטר ומחזירה את הנתיב האbsolutי הרצוי `pathname`.

תבצע בדיקה: אם הנתיב אינו קיים תישלח הודעה ללקוח שלא ניתן למחוק את התקiya הנ"ל.

אם `theUser` לא מושה למחוק תיקיות מהשרת או תישלח הודעה של שגיאה.

(ניתן לדעת זאת לפי `data member` בשם `allow_delete`).

אחרת, התקiya נמחקת לפי הפונקציה (`shutil.rmtree`), ותישלח הודעה ללקוח שהפעולה הצליחה.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתרכעת בדיקה.

אם השגיאה אינה מסוג `class` שבנו (`UserNotAuthenticatedException`), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) וכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:RNFR(30

```
def RNFR(self, filename):
    log("RNFR(" + filename + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        pathname = self.getAbsolutePath(filename)

        # if the file/dir that we try to rename doesn't exist then return error message
        if not os.path.exists(pathname):
            self.sendCommand('550 RNFR failed, file/dir "%s" does not exists.\r\n' % pathname)
        else:
            self.fileRenameFrom = pathname
            self.sendCommand("350 File exists, ready for destination name.\r\n")
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("RNFR function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה הזאת מקבלת שם של תיקייה\קובץ שורצים לשנות את שם ושמירת
אותו בזאת `data member` בשם `fileRenameFrom`.

הfonקציה מדפיסה את שם התיקייה\קובץ, ולאחר מכן נכנסת למבנה קוד עטופה
`try except`.

נעשה בדיקה אם `user` עבר אונטנטיקציה.

נעשה קרייה לפונקציה `getAbsolutePath` שמקבלת את `filename` כפרמטר
ומחזיר את הנתיב האבסולוטי הרצוי `pathname`.

תרבוץ בדיקה: אם הנתיב אינו קיים תישלח הודעה ללקוח שלא קיימים
תיקייה\קובץ זהה.

אחרת, ה `data member` בשם `fileRenameFrom` שווה ל`pathname`, ונשלחת
ההודעה שקיים זהה קובץ ושיש לחת את השם החדש.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת וממבצעת בדיקה.

אם השגיאה אינה מסוג ה `class UserNotAuthenticatedException`, אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שמרתחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:RNTO(31)

```
def RNTO(self, filename):
    log("RNTO(" + filename + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # get the absolute path to the file / folder
        fileRenameTo = self.getAbsolutePath(filename)

        # if the file/dir that we try to rename exist then return error message
        if os.path.exists(fileRenameTo):
            self.sendCommand('550 RNTO failed, file/dir "%s" already exists.\r\n' % fileRenameTo)
        else:
            # perform the rename action
            os.rename(self.fileRenameFrom, fileRenameTo)
            # return success message
            self.sendCommand('250 File or directory renamed successfully.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("RNTO function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה הזאת מקבלת שם חדש לתיקייה\קובץ שורצים לשנות את שם(מהפקודה RNFR) ומשנה את השם.

הfonקציה מדפיסה את שם התיקייה\קובץ, ולאחר מכן נכנסת לבניית קוד עטופה `try` ו-`except`.

נעשית בדיקה אם `user` עבר אוטנטיקציה.

נעשית קרייה לפונקציה `getAbsolutePath` שמקבלת את `filename` כפרמטר ומחזירה את הנתיב האבסולוטי הרצוי `pathname`.

תבצע בדיקה: אם התיקייה\קובץ שמנסים לשנות את שמה כבר קיימת עם אותו השם הרצוי והחדש תישלח הודעה ללקוח שקייםים כבר תיקייה\קובץ בשם זה.

אחרת, על ידי os.rename() שמקבלת את השם של התיקייה שרצים לשנות את שמה ואת השם החדש.

לאחר מכן, תישלח הודעה שהם שונה בהצלחה.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתבצעת בדיקה.

אם השגיאה אינה מסוג המאובט `UserNotAuthenticatedException`, אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שמרתחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:REST(32)

```
def REST(self, newStartingPosition):
    log("REST(" + newStartingPosition + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # convert the received newStartingPosition from string to int and save it
        self.startingPosition = int(newStartingPosition)
        self.sendCommand('250 File position reseted.\r\n')
    except Exception as err:
        if err.__class__.__name__ != 'UserNotAuthenticatedException':
            logCommand("REST function failed", err)
            self.sendCommand('500 Operation Failed.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה הזאת מקבלת כפרמטר סטריניג של האינדקס של המיקום בקובץ, שאומר לשרת מאיפה לקרוא את `data` ובהתאם לשלוח אותו ללקוח.

הfonקציה מדפסה את שם התיקייה, ולאחר מכן נכנסת לתבנית קוד עטופה ב `try` .`except`

נעשה בדיקה אם `user` עבר אוטנטיקציה.

הfonקציה מקבלת את הערך של האינדקס מומר ל `int`.

נשלחת הודעה ללקוח שהפעולה הצליחה.

אם נזרקת שגיאה בזמן הפקודה, היא נתפסת ומתבצעת בדיקה.

אם השגיאה אינה מסוג ה `class UserNotAuthenticatedException` (UserNotAuthenticatedException), אז אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה (שמרתחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה ותישלח הודעה ללקוח שהפקודה נכשלה.

:RETR(33

```
def RETR(self, filename):
    log("RETR(" + filename + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # if user did not supply a filename then return error message
        if not filename:
            self.sendCommand('500 Operation Failed, Please supply a filename to download.\r\n')
        else:
            # get the absolute path to the file / folder
            fileToDownload = self.getAbsolutePath(filename)

            # check if the file to download exist on the server
            if not os.path.exists(fileToDownload):
                self.sendCommand('500 Operation Failed, The filename does not exist.\r\n')

            else:
                # if we are working in binary mode then open the file to Read in binary mode
                if self.mode == 'I':
                    file = open(fileToDownload, 'rb')
                else:
                    # if we are working in ascii mode then open the file to Read in ascii mode
                    file = open(fileToDownload, 'r')

                # send message to client to tell it that we are working on his request
                self.sendCommand('150 Opening data connection.\r\n')
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonktsiya hzat mklbilat cprmtur sm shel kovz moridah otto mahsharit lkkot.

hfonktsiya mdafisa at sm kovz, vla'achr mkn ncnstt ltbnit kod utopha bzyt .except.

nusiyt bdkha sm user ubr autntiktsya.

nusiyt bdkha: sm la sopek shom prmtur maha user nshlch hzda lkkot lsfk sm shel kovz.

אחרת, נעשית קרייה לפונקציה `getAbsolutePath` שמקבלת את `filename` כפרמטר ומחזירה את הנתיב האבסולוטי הרצוי `.fileToDownload`.

לאחר מכן נעשית עוד בדיקה, אם לא קיים在乎 קובץ נשלחת הודעה ללקוח בהתאם.

אחרת, אם המוד שספק על ידי `user` הוא בינהר או שהשרת פותח את הקובץ במצב קרייה בינהר ואם המוד הוא לא בינהר השרת פותח את הקובץ במצב קרייה אסקוי(טקסט), ונשלחת הודעה ללקוח שנפתח חיבור.

```
# open the dataSocket to the client
self.openSocket()

# set read starting position to the startingPosition var
file.seek(self.startingPosition)

# reset the starting position back to 0 so next download will start from the beginning of the file
self.startingPosition = 0

# loop and read all the data from the file and write it into the socket
# until there is nothing more to read
while True:
    if self.mode == 'I':
        # read 1024 bytes from the file
        data = file.read(1024)

        # check if data was read from the file, if not - get out of the loop (finish reading)
        if not data:
            break
    else:
        # because the client asked us to work in ascii mode, we need to send it a CRLF character
        # at the end of each line, so make sure each line ends with \r\n
        currentLine = file.readline()

        # check if data was read from the file, if not - get out of the loop (finish reading)
        if not currentLine:
            break
```

נפתח סוקט להעברת הנתונים, והסמן מוזע לקרייה לפי המEMBER `data` בשם `startingPosition`. מיד אחרי זה אנחנו מAppending אותו כדי שההורדה הבאה תתחילה מההתחלת של הקובץ ולא תפגע בהורדנה.

נכנסים לולאה נוספת ובודקים: אם המוד הוא בינהר או קוראים את `data` בגודל של 1024 בתיים, ואם לא מתקבל יותר `data` אז יוצאים מהlolאה.

אחרת, המוד הוא אסקוי(טקסט) ולכן אנחנו מחווים לבדוק שיש סימנת של `(\r\n)` בסוף כל שורה(מעטימי מערכת הפעלה).

לכן נקרא את `data` בשורות בכל פעם, ונציב במשתנה בשם `currentLine`. אם לא מקבל יותר מידע, נצא מהlolאה.

```

if not currentLine.endswith("\r\n"):
    # remove the last character [should be \n (Unix systems) or \r (Mac systems)]
    currentLine = currentLine[:-1]
    # add to the end of the line the CRLF end line characters (Windows systems)
    data = bytes(currentLine + '\r\n', 'utf-8')

    # send to the dataSock the 1024 bytes you read from file, the sendData function will decide
    # if to send it binary or ascii (text) base on the client preferences
    self.sendData(data)

# close the file and allow others to use it
file.close()

# close the dataSocket
self.closeSocket()

# send the client a success message
self.sendCommand('226 Transfer completed.\r\n')
except Exception as err:
    if err.__class__.__name__ != 'UserNotAuthenticatedException':
        logCommand("RETR function failed", err)
        self.closeSocket()
        self.sendCommand('500 Operation Failed.\r\n')

```

לכן אם השורה שנקראת לא מסתיימת ב-CRLF (הכוונה שמסתיימת בח' בلينוקס לדוגמא) אז נוריד את הסיוומת מכל שורה שנקראת, נוסיף את הסיוומת ח'ז', נמיר את השורה לבתים ונשים ב-`data`(בכי ה-`data` מועברת כמערך של bytes לפי הגדרת הפונקציה `sendData`).

נשלח את המידע ללקוח.

לאחר שלא יהיה יותר `data`, נסגור את הקובץ ואת הסקוט, ונסלח הודעה ללקוח שההעברה התרחשה בהצלחה.

אם נזרקת שגיאה בזמן הפקודה, היא נפתחת ומתבצעת בדיקה.

אם השגיאה אינה מסווג כ-`UserNotAuthenticatedException`, אך אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שמרתחשת בהתחלה וזרקת שגיאה ספציפית) ולכן תודפס סוג השגיאה, ייסגר הסקוט ותישלח הודעה ללקוח שהפקודה נכשלה.

:STOR(34

```
def STOR(self, filename):
    log("STOR(" + filename + ")")
    try:
        # check if user is authenticated
        self.isUserAuthenticated()

        # if user did not supply a filename then return error message
        if not filename:
            self.sendCommand('500 Operation Failed, Please supply a filename to upload.\r\n')
        else:
            # get the absolute path to the file / folder
            fileToUpload = self.getAbsolutePath(filename)

            if self.isAppend:
                # open the file to Write from the end of the file (append) in binary mode
                file = open(fileToUpload, 'ab')
                # reset the append flag back to it's default value (false)
                self.isAppend = False
            else:
                # open the file to Write (new file or overwrite) in binary mode (always write byte array to a file)
                file = open(fileToUpload, 'wb')

            # send message to client to tell it that we are working on his request
            self.sendCommand('150 Opening data connection.')

            # open the dataSocket to the client
            self.openSocket()
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `FtpServerProtocol`.

הfonקציה הזאת מקבלת כפרמטר שם של קובץ ומעלה אותו מהלך לשרת.

הfonקציה מדפסה את שם הקובץ, ולאחר מכן נכנסת למבנה קוד עטופה ב`try`

`.except`.

נעשית בדיקה אם `user` עבר אונטנטיקציה.

נעשית בדיקה: אם לא סופק שם פרמטר מה`user` נשלחת הודעה ללקוח לספק שם של קובץ.

אחרת, נעשית קרייה לפונקציה `getAbsolutePath` שמקבלת את `filename` כפרמטר ומחזירה את הנתיב האבסולוטי הרצוי `fileToUpload`.

לאחר מכן נעשית עוד בדיקה, אם המEMBER `data` בשם `isAppend` שווה ל`True`, אז השרת פותח קובץ לבתיבה במוד `ab` שזה אומר בתיבה מסוף הקובץ באופן בינארי, וההMEMBER `isAppend` בשם `data`/member שלו שהוא `False`.

אחרת, הדרישה היא להעלות קובץ חדש(או דרישת של קובץ קיים) ולכן השרת יפתח את הקובץ לבתיבה במוד `w` שזה אומר בתיבה מתחילה הקובץ באופן בינארי.

נשלחת הودעה ללקוח שנפתח חיבור ונפתח סוקט להעברת הנתונים.

```
# loop and read all the data from the socket and write it into the file
# until there is nothing more to read
while True:
    # read from the socket 1024 bytes/chars (based on client preferences)
    data = self.dataSock.recv(1024)

    # check if data was received, if not - get out of the loop (finish reading)
    if not data:
        break

    # # if client asked us to receive in ascii mode, then we need to convert (decode) the strings into
    # # byte array using UTF8 mapping
    # if self.mode == 'A':
    #     data = data.decode("utf-8")

    # write the 1024 bytes you read from the socket into the file
    file.write(data)

    # close the file and allow others to use it
    file.close()

    # close the dataSocket
    self.closeSocket()
```

נכנים לולאה אינסופית וקוראים את `data` בגודל של 1024 בתים, כתובים את `data` אל תוך הקובץ ואם לא מתקבלת יותר `data` אז יוצאים מהלולה. לאחר מכן סגורים את הקובץ ואת הסוקט.

```
        # send the client a success message
        self.sendCommand('226 Transfer completed.\r\n')
except Exception as err:
    if err.__class__.__name__ != 'UserNotAuthenticatedException':
        logCommand("STOR function failed", err)
        self.closeSocket()
        self.sendCommand('500 Operation Failed.\r\n')
```

אם נזקפת שגיאה בזמן הפוקודה, היא נתפסת ומתרכעת בדיקה. אם השגיאה אינה מסווג כ-`class UserNotAuthenticatedException`, או אנחנו יודעים שיש לנו שגיאה בפקודה עצמה ולא בבדיקה האוטנטיקציה(שתרחשת בהתחלה וזרקת שגיאה ספציפית) וכן תודפס סוג השגיאה, ייסגר הסוקט ותישלח הודעה ללקוח שהפקודה נכשלה.

:APPE(35

```
def APPE(self, filename):
    log("APPE(" + filename + ")")
    self.isAppend = True
    self.STOR(filename)
```

פונקציה של אוחת הפקודות ב프וטוקול FTP.
פונקציה זאת מבוצעת על האובייקט FtpServerProtocol.
הfonקציה הזאת מקבלת כפרמטר שם של קובץ ומעלה את שארית הקובץ שלא
הועלה(לדוגמא כי הופסקה העברה) מהלך לשרת.
הfonקציה מדפסה את שם הקובץ, מגדרה את ה`ezmber` true ל`isAppend` data member
על מנת שבקרירה לפונקציה STOR הfonקציה תדע לעבוד במוד של append.

:HELP(36

```
def HELP(self, arg):
    logCommand('HELP', arg)
    help = """
    self.sendCommand(help + "\r\n")
```

פונקציה של אוחת הפקודות בפרוטוקול FTP.
פונקציה זאת מבוצעת על האובייקט FtpServerProtocol.
הfonקציה עוזרת ללקוח ומספקת לו מידע לגבי כל פקודה.
הfonקציה מקבלת ארגומנט(שmagiu מfonקציית הח`ru`) ומדפסה אותו.
לאחר מכן שולחת ללקוח סטרינג שמכיל את הכל המידע הנחוץ לו.

:QUIT(37)

```
def QUIT(self):
    global allThreads
    log('QUIT')
    try:
        self.sendCommand('221 Goodbye.\r\n')
        self.closeSocket()
    except Exception as err:
        log("Warning: failed to close sockets for thread: " + self.threadName + " due to error: " + str(err))
    finally:
        allThreads.pop(self.threadName)
```

פונקציה של אחת הפקודות ב프וטוקול FTP.

פונקציה זאת מבוצעת על האובייקט `.FtpServerProtocol` על `.allThreads`.

הfonקציה מקבלת גישה גלובלית למשתנה `allThreads` בעתה. `try` ו-`except`.

הfonקציה שולחת ללקוח הודעת `goodbye` וסגורת את הסוקט של הלקוח.

אם נזרקת שגיאה מודפס שם `thread` שטוף בחיבור הנוכחי ללקוח והשגיאה עצמה.

לבסוף בערך `finally`, הfonקציה מסירה את `thead` שטוף בחיבור מה`dictionary`. `allThreads` בשם.

:sendWelcome(38)

```
def sendWelcome(self):
    self.sendCommand('220 Welcome.\r\n')
```

פונקציה פשוטה ששולחת ללקוח הודעת `Welcome`.

:serverListener(39)

```
def serverListener():
    global mainServerSocket
    global isListening

    try:
        # create the server socket based on the selected protocol
        if isTCPIP:
            mainServerSocket = TCPIPSocket()
        else:
            mainServerSocket = RUDPSocket()

        # start the server main socket that listens to client incoming connections
        mainServerAddress = (SERVER_HOST, SERVER_PORT)
        mainServerSocket.listen(mainServerAddress)

        # mark the server as listening
        isListening = True
        logCommand('Server started', f'Listen on: {SERVER_HOST}, {SERVER_PORT}')
    except Exception as err:
        logCommand("Error: cannot launch server, error", err)
    # wait for clients to connect
    while True:
        try:
            # wait for incoming connections and accept socket connections from clients
            clientSocket = mainServerSocket.accept()
            newClientID = f'{clientSocket.receiverAddress[0]}:{clientSocket.receiverAddress[1]}'"

            if newClientID not in allConnectedClients:
                allConnectedClients[newClientID] = "Connected"

            # create new thread that will handle the incoming socket
            clientThread = FtpServerProtocol(clientSocket, generateUniqueThreadName())

            # start the thread so it will handle the client requests
            clientThread.start()
            logCommand('Accept', 'New client connected %s, %s' % clientSocket.receiverAddress)

        except Exception as err:
            if isListening:
                logCommand("Error: cannot accept connection, error: ", err)
            else:
                log("Warning: cannot accept any more connections, server is shutting down")
                break
```

הfonקציה הזאת פותחת את הסוקט הראשי של הרשת(mainServerSocket) ומחבה ללקוחות שיתחברו לשרת.

כאשר לוקח מתחבר נוצר thread חדש שמקבל אובייקט של `FtpServerProtocol` אשר מפעיל את פונקציית `thechnique` שלו על מנת לטפל בבקשות של הלוקה. הפונקציה מקבלת גישה גלובלית `mainServerSocket` ו`isListening`. ההפונקציה נכנסת לתבנית קוד עטופה ב`try` ו`except`.

נעשית בדיקה: אם `isTCPIP` הוא `True` אז הsocket הראשי של השירות מסוג TCP, אחרת הוא מסוג RUDP.

נוצר `tuple` אשר מחזק בכתובת הקו ופורט של השירות. הsocket הראשי של השירות מסומן במאזין וכן יקבל את `thechnique` דרך פונקציית `().listen`.

מגדירים את `data member` בשם `isListening` בדמות `True` כדי שהשירות יוכל להאזין לחיבורם, ומדפיסים הודעה שהשירות תחיל להאזין. אם נזקפת שגיאה היא מודפסת ונכתב שהשירות לא הצליח להתרום. לאחר מכן נכנסים לולאה אינסופית שבתוכה תבנית הקוד עטופה ב`try` ו`except`.

לאחר מכן נוצר סוקט חדש לлокוח אשר מתקיים מפונקציית `(accept)` שמופעלת על הsocket הראשי של השירות. בונים את השם של הקליינט שהתחבר מהקו והפורט ואם עד עכשיו הוא לא התחבר, מכניסים אותו לרשימה הקלינייטים המתחברים - `AllConnectedClient`.

לאחר מכן יוצרים thread חדש שפועל על אובייקט נוסף מסוג `Protocol`, שמקבל בפרמטרים את הsocket לлокוח ופונקציה שמייצרת שם ייחודי של `thread` באופן רנדומלי.

thread מרים את פונקציית `thechnique` שלו ומודפסת הודעה שלוקוח חדש התחבר. אם נזקפת שגיאה אז נעשית בדיקה:

אם `isListening` הוא `True` – משמע השירות עדין מאזין לлокחות, אך מודפסת הודעה שהשירות לא יכול לקבל חיבורם ואת השגיאה עצמה. אחרת, מודפס שהשירות לא יכול לקבל עוד חיבורם והוא סגור את עצמו. לאחר מכן יבוא `break` על מנת לצאת מהלולאה.

:main(40) פונקציית

```
if __name__ == "__main__":
    try:
        # start the ftp server in a separated thread so the main thread can listen to Q and Ctrl+C keys
        logCommand('Start ftp server', 'press q and Enter or Ctrl+C to stop the ftp server')
        listener = threading.Thread(target=serverListener)
        listener.start()

        # if server admin asked to stop the FTP server - quit
        if input().lower() == "q":
            isListening = False
            while True:
                log("Waiting for all clients to disconnect ...")
                time.sleep(2)
                if len(allThreads) == 0:
                    break
                time.sleep(0.5)
            mainServerSocket.close()
            sys.exit()
    except KeyboardInterrupt:
        print("Ctrl+C pressed, Shutting down FTP Server")
        os._exit(-1)
```

בפונקציית החסימה שמריצה את התוכנה, קיימת תבנית קוד עטופה בtry וexcept.

תודפס הודעה שFTP server נפתח ואם רצים לכבות אותו יש ללחוץ q ואנטר או ctrl+c.

ווזר thread חדש שמקבל את serverListener שמאזין לחיבורים על שרת ftp, והוא יתחל לרטז.

תבצע בדיקה: אם המשתמש שלח q או Q אז isListening=False וגדיר כFalse על מנת שהשרת יפסיק להאזין ללקוחות.

תתחל לולאה אינסופית שתתდפיס שהשרת ממתין שככל הלקוחות יתנתקו – שככל threads יפסיקו את הפעולה שלהם באשר ניתן sleep של 2 שניות כדי לחכות 2 שניות בין בדיקה לבדיקה שככל threads סיום.

באשר כל threads מסיים ומוסרים מהdictionary של all והגודל שלו הוא 0 אז ניתן לצאת מהלולאה.

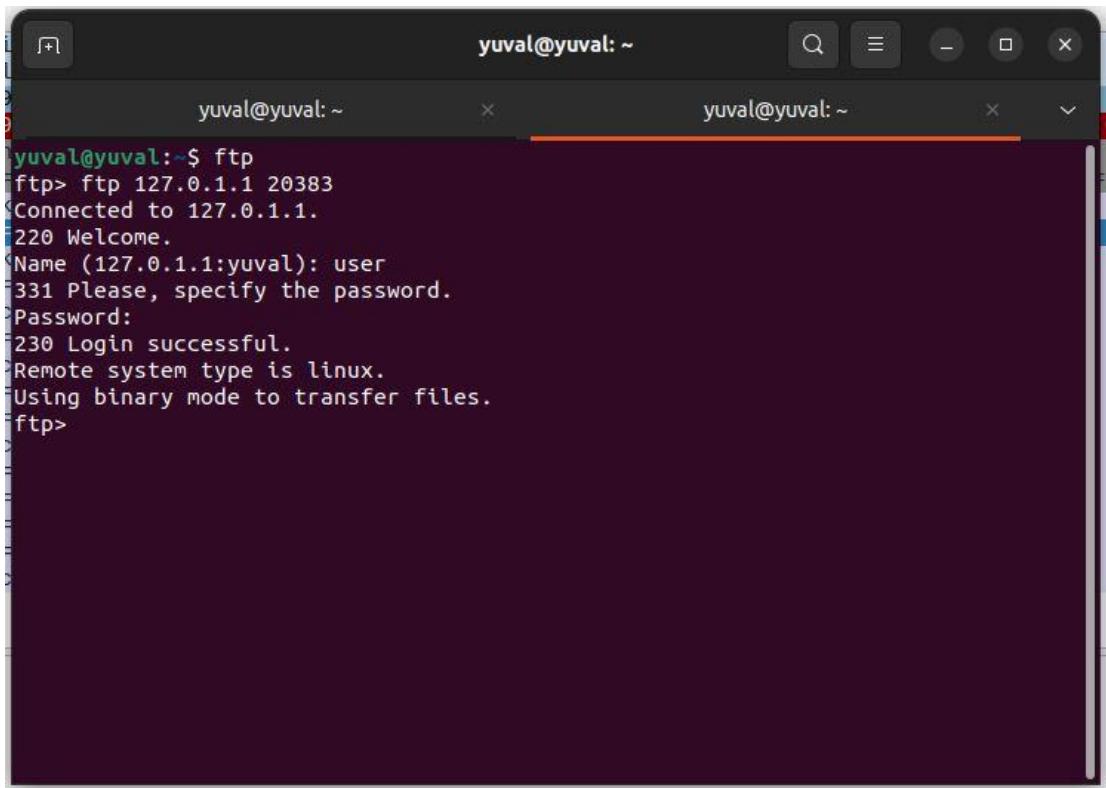
לאחר מכן ניתן עוד sleep של חצי שנייה לפני שהסוקט הראשי של השרת יסגור. לאחר מכן נמצא נזק מהתובנית על ידי sys.exit().

אם נזרקת שגיאה (צפיה לסוג KeyboardInterrupt) אז תודפס הודעה שctrl+c נלחץ ולכן נמצא נזק מהתובנית עם סטטוס קוד של 1.

תמונהות הרצה לדוגמא:

```
/home/yuval/PycharmProjects/YEAR2/venv/bin/python /home/yuval/PycharmProjects/YEAR2/ReshatotTikshoret/ftp_server.py
2023-03-12 18:12:25 [-] Start ftp server: press q and Enter or Ctrl+C to stop the ftp server
2023-03-12 18:12:25 [-] Server started: Listen on: 127.0.1.1, 20383
2023-03-12 18:12:47 [-] Accept: New client connected 127.0.0.1, 41824
2023-03-12 18:12:50 [-] Data from client: USER user
2023-03-12 18:12:50 [-] USER(user)
2023-03-12 18:12:52 [-] Data from client: PASS 1234
2023-03-12 18:12:52 [-] PASS(1234)
```

כأن ניתן לראות שהשרת מדפיס את הפקודות המתתקבלות מהלקוח, כמו שם המשתמש user והסיסמה. הפקודות USER ו-PASS נקראות בהתאם.

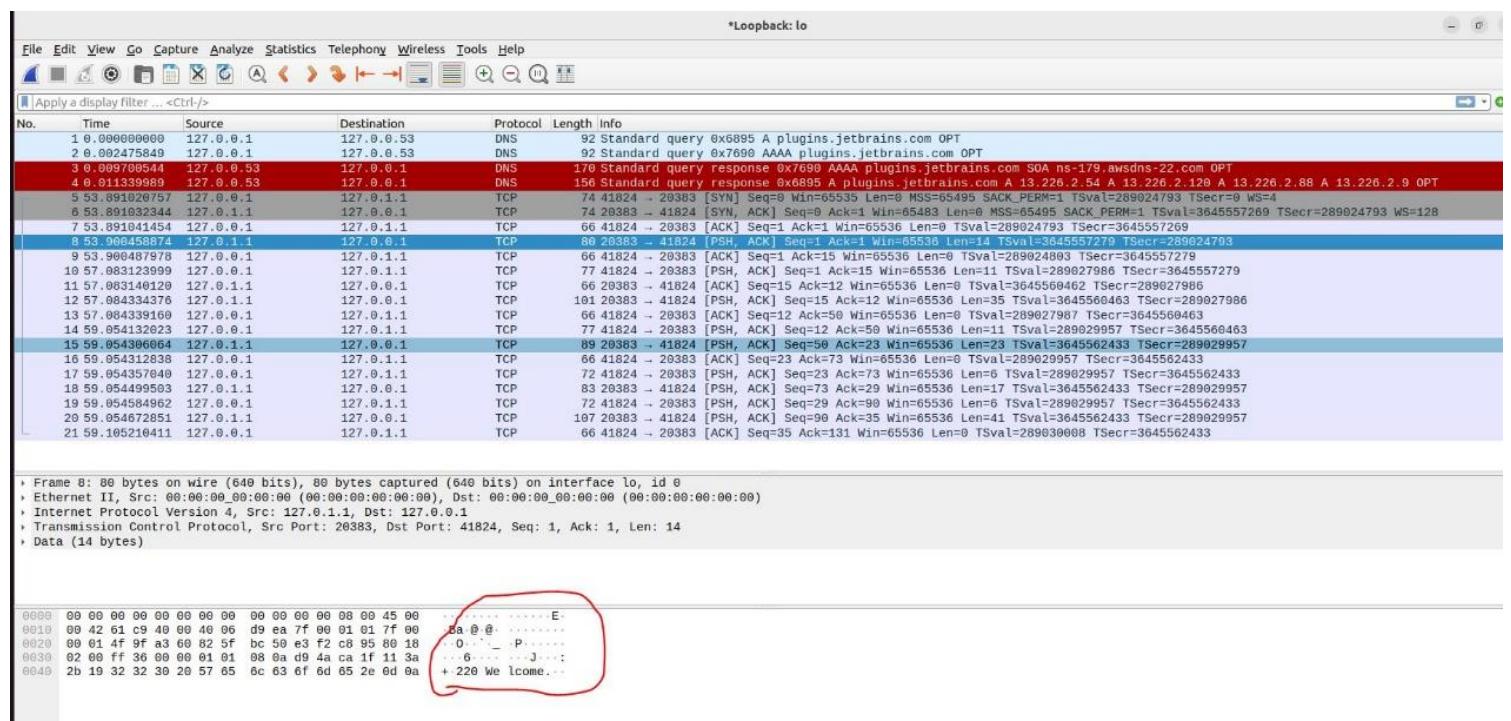


The screenshot shows a terminal window with two tabs. The left tab displays the command-line interface of the FTP server, showing log entries for the connection and the user's login attempt. The right tab shows the user's perspective, where they have logged in successfully as the 'user' account.

```
yuval@yuval:~$ ftp
ftp> ftp 127.0.1.1 20383
Connected to 127.0.1.1.
220 Welcome.
Name (127.0.1.1:yuval): user
331 Please, specify the password.
Password:
230 Login successful.
Remote system type is linux.
Using binary mode to transfer files.
ftp>
```

ניתן לראות שהפעילו את השירות FTP ואת הלקוח FTP והתחרבנו לכתובת הקין
והפורט המתאימים של השירות בהצלחה.

הצלחנו לקבל את הודעה welcome מהשירות.



כמו כן, ניתן לראות שהודעת welcome הוסנפה גם דרך הווירשאך.

הוודה מהשרת אל הלוקוט:

Terminal Screenshot showing two sessions. The left session is an FTP connection from user 'yuval' to port 20383. The right session is a terminal session where the user runs a script named 'commands'. The 'commands' session shows the server's internal log, including the removal of the 'welcome' message.

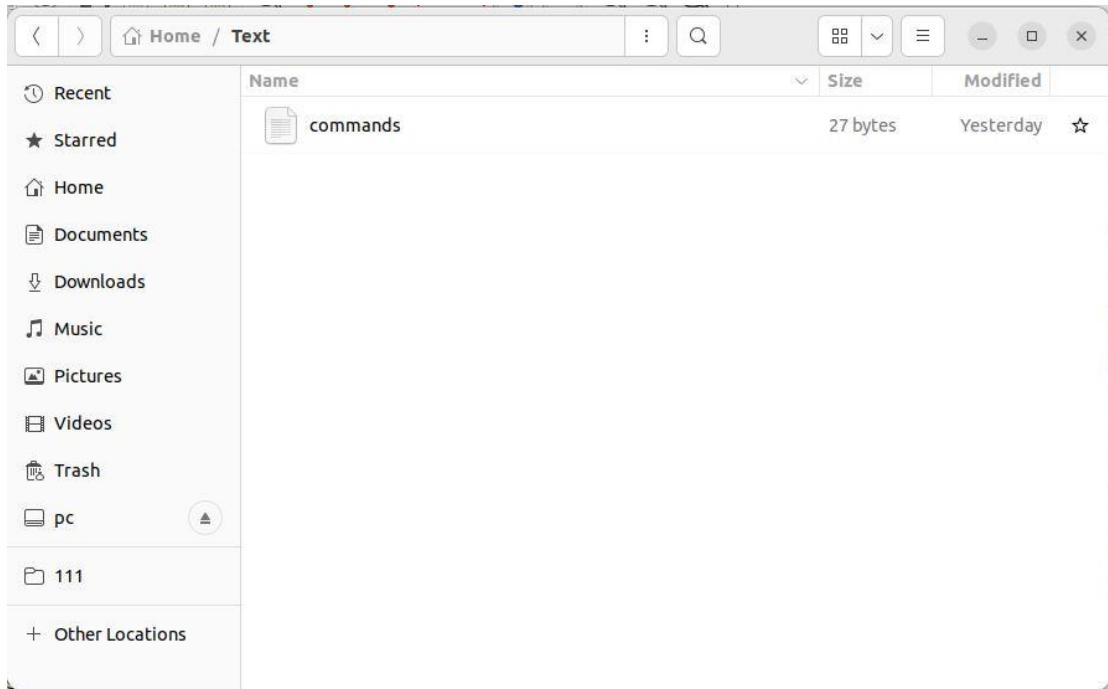
```

yuval@yuval: ~/PycharmProjects/YEAR2/ReshatotTikshoret
yuval@yuval: ~/PycharmProjects/YEAR2/ReshatotTikshoret$ ftp
ftp> open 127.0.1.1 20383
Connected to 127.0.1.1.
220 Welcome.
Name (127.0.1.1:yuval): user
331 Please, specify the password.
Password:
230 Login successful.
Remote system type is linux.
Using binary mode to transfer files.
ftp> cd home/yuval/Text
250 CWD Command successful.
ftp> get
(remote-file) commands
(local-file) /home/yuval/Music/111
local: /home/yuval/Music/111 remote: commands
227 Entering Passive Mode (127,0,1,1,117,128).
150 Opening data connection.
27 850.55 Kb/s
226 Transfer completed.
27 bytes received in 00:00 (0.65 KiB/s)
ftp> []

yuval@yuval: ~/PycharmProjects/YEAR2/ReshatotTikshoret
yuval@yuval: ~/PycharmProjects/YEAR2/ReshatotTikshoret$ 
2023-03-13 18:41:55 [-] TYPE(I)
2023-03-13 18:41:55 [-] Data from client: SIZE commands
2023-03-13 18:41:55 [-] Receive: 'FtpServerProtocol' object has no attribute 'SIZE'
-> 20383 [PS 2023-03-13 18:41:55 [-] Data from client: EPSV
-> 51362 [PS 2023-03-13 18:41:55 [-] Receive: 'FtpServerProtocol' object has no attribute 'EPSV'
-> 20383 [AC SV]
-> 20383 [PS 2023-03-13 18:41:55 [-] Data from client: PASV()
-> 51362 [PS 2023-03-13 18:41:55 [-] PASV()
-> 20383 [PS 2023-03-13 18:41:55 [-] Data from client: RETR commands
-> 51362 [PS 2023-03-13 18:41:55 [-] RETR(commands)
-> 20383 [PS 2023-03-13 18:41:55 [-] getAbsolutePath(commands)
-> 51362 [PS 2023-03-13 18:41:55 [-] getAbsolutePath() returning: /home/yuval/Text/commands
-> 30080 [SY 2023-03-13 18:41:55 [-] openSocket()
-> 44360 [SY 2023-03-13 18:41:55 [-] openSocket(): waiting for client to connect in passive mode
-> 30080 [AC code
-> 20383 [PS 2023-03-13 18:41:55 [-] openSocket(): connected to client in passive mode
-> 51362 [PS 2023-03-13 18:41:55 [-] openSocket(): dataSocketIP: 127.0.0.1 dataSocketPort: 44
-> 44360 [PS 360
-> 30080 [AC 2023-03-13 18:41:55 [-] closeSocket()
-> 44360 [FI 2023-03-13 18:41:55 [-] Data from client: MDTM commands
-> 30080 [FI 2023-03-13 18:41:55 [-] Receive: 'FtpServerProtocol' object has no attribute 'MDTM'
-> 44360 [AC TM'
-> 20383 [FBI

```

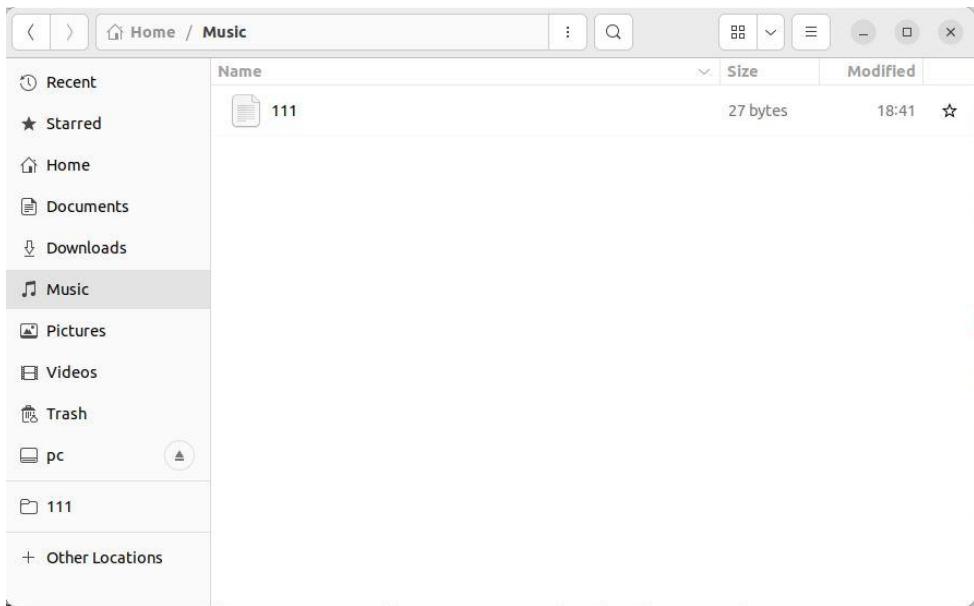
ניסינו להוריד קובץ בשם commands מתיקייה בשם Text.



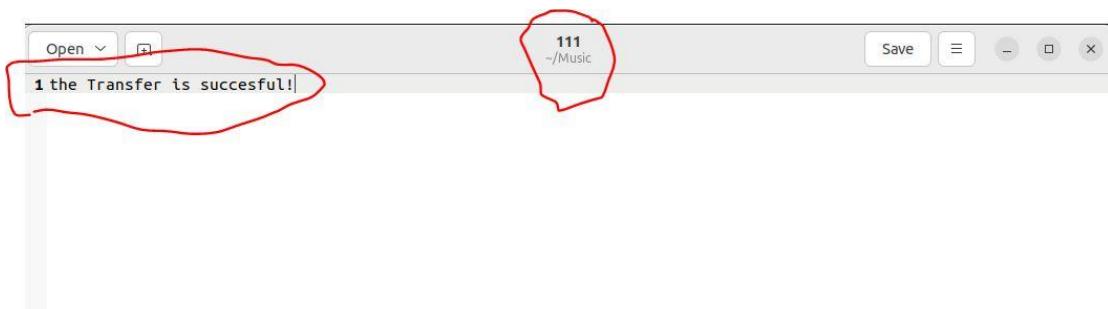
בתוך הקובץ commands כתבנו "the Transfer is successful" כדי שנוכל לבדוק אותו לאחר העברה.



עשינו get לנתיב של הקובץ זהה והעבכנו אותו לנתיב של תיקיית Music ושינוינו לו את השם ל111.



ניתן לראות שהקובץ נמצא בתיקייה Music עם השם 111 וכתובת ההודעה
ולכן ההערכה הצלילה.



Capturing from Loopback: lo

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter ... <Ctrl-/>

No.	Time	Source	Destination	Protocol	Length	Info
1	0.0000000000	127.0.0.1	127.0.0.1	TCP	74	51362 - 20383 [PSH, ACK] Seq=1 Ack=1 Win=16384 Len=8 Tsvl=3879585629 Tsecr=3820527494
2	0.000145510	127.0.0.1	127.0.0.1	TCP	84	20383 - 51362 [PSH, ACK] Seq=1 Ack=9 Win=512 Len=18 Tsvl=3879585629 Tsecr=3879585629
3	0.000153389	127.0.0.1	127.0.0.1	TCP	66	51362 - 20383 [ACK] Seq=9 Ack=19 Win=16384 Len=0 Tsvl=3879585630 Tsecr=3820527494
4	0.000211482	127.0.0.1	127.0.0.1	TCP	81	51362 - 20383 [PSH, ACK] Seq=9 Ack=19 Win=16384 Len=15 Tsvl=3879585630 Tsecr=3820527494
5	0.000265327	127.0.0.1	127.0.0.1	TCP	107	20383 - 51362 [PSH, ACK] Seq=19 Ack=24 Win=512 Len=42 Tsvl=3820527494 Tsecr=3879585630
6	0.000354907	127.0.0.1	127.0.0.1	TCP	72	51362 - 20383 [PSH, ACK] Seq=24 Ack=60 Win=16384 Len=6 Tsvl=3879585630 Tsecr=3820527494
7	0.000468283	127.0.0.1	127.0.0.1	TCP	107	20383 - 51362 [PSH, ACK] Seq=60 Ack=30 Win=512 Len=42 Tsvl=3820527494 Tsecr=3879585630
8	0.000584251	127.0.0.1	127.0.0.1	TCP	72	51362 - 20383 [PSH, ACK] Seq=30 Ack=18 Win=16384 Len=0 Tsvl=3879585630 Tsecr=3820527494
9	0.000675055	127.0.0.1	127.0.0.1	TCP	114	20383 - 51362 [PSH, ACK] Seq=101 Ack=3 Win=512 Len=48 Tsvl=3820527494 Tsecr=3879585630
10	0.000879864	127.0.0.1	127.0.0.1	TCP	74	44360 - 30689 [SYN] Seq=0 Win=65536 Len=0 MSS=65496 SACK_PERM=1 Tsvl=3879585630 Tsecr=3820527494 WS=4
11	0.000887864	127.0.0.1	127.0.0.1	TCP	74	30689 - 44360 [SYN, ACK] Seq=0 Ack=1 Win=65483 Len=0 MSS=65496 SACK_PERM=1 Tsvl=3820527494 Tsecr=3879585630 WS=128
12	0.000894974	127.0.0.1	127.0.0.1	TCP	66	44360 - 30689 [ACK] Seq=1 Ack=1 Win=65536 Len=0 Tsvl=3879585630 Tsecr=3820527494
13	0.000894186	127.0.0.1	127.0.0.1	TCP	81	51362 - 20383 [PSH, ACK] Seq=36 Ack=149 Win=16384 Len=15 Tsvl=3879585630 Tsecr=3820527494
14	0.0010181219	127.0.0.1	127.0.0.1	TCP	96	20383 - 51362 [PSH, ACK] Seq=149 Ack=51 Win=512 Len=30 Tsvl=3820527495 Tsecr=3879585630
15	0.0010184428	127.0.0.1	127.0.0.1	TCP	66	30689 - 44360 [PSH, ACK] Seq=1 Ack=1 Win=65536 Len=21 Tsvl=3820527495 Tsecr=3879585630
16	0.001139643	127.0.0.1	127.0.0.1	TCP	66	44360 - 30689 [ACK] Seq=1 Ack=20 Win=65536 Len=0 Tsvl=3879585631 Tsecr=3820527495
17	0.001203000	127.0.0.1	127.0.0.1	TCP	80	44360 - 30689 [ACK] Seq=20 Ack=21 Win=65536 Len=0 Tsvl=3879585631 Tsecr=3820527495
18	0.005684614	127.0.0.1	127.0.0.1	TCP	66	44360 - 30689 [FIN, ACK] Seq=29 Ack=29 Win=65536 Len=0 Tsvl=3879585631 Tsecr=3820527495
19	0.005687861	127.0.0.1	127.0.0.1	TCP	66	30689 - 44360 [ACK] Seq=29 Ack=29 Win=65536 Len=0 Tsvl=3820527499 Tsecr=3870585633
20	0.0141642328	127.0.0.1	127.0.0.1	TCP	66	51362 - 20383 [ACK] Seq=51 Ack=179 Win=16384 Len=0 Tsvl=3879585671 Tsecr=3820527495
21	0.0141659183	127.0.0.1	127.0.0.1	TCP	91	20383 - 51362 [PSH, ACK] Seq=179 Ack=51 Win=512 Len=25 Tsvl=3820527535 Tsecr=3879585671
22	0.0141684708	127.0.0.1	127.0.0.1	TCP	66	51362 - 20383 [ACK] Seq=51 Ack=284 Win=16384 Len=0 Tsvl=3879585671 Tsecr=3820527535
23	0.0141757280	127.0.0.1	127.0.0.1	TCP	81	51362 - 20383 [PSH, ACK] Seq=51 Ack=284 Win=16384 Len=15 Tsvl=3879585671 Tsecr=3820527535

* Frame 13: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface lo, id 9

Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src Port: 127.0.0.1, Dst Port: 127.0.0.1

Transmission Control Protocol, Src Port: 51362, Dst Port: 20383, Seq: 36, Ack: 149, Len: 15

Source Port: 51362
Destination Port: 20383
[Stream index: 0]
[Conversation completeness: Incomplete (12)]
[TCP Segment Len: 15]

0000 00 00 00 00 00 00 00 00 00 00 00 00 45 10
0010 00 43 ea e7 40 00 40 06 5b 7f 00 00 01 7f 00
0020 01 c8 a2 4f 9f a6 a3 b9 38 01 47 3a 80 18
0030 49 ff 37 00 00 01 01 00 00 e7 3d c5 06 e3 b8
0040 86 52 45 54 52 20 63 f6 6d 6d 61 6e 64 73 0d
0050 0a

במו כן ניתן לראות בהקלות וירשארק את הפקטה שקרה כי בוצעה פקודה לשרת בשם RETR שהוא בעצם פעולה get והารוגומנט שנייתן לה הוא שם הקובץ commands.

Utils

בחלק זה נסביר על מחלקות העוזר ליישום שרת ה-ftp.

קבינסטנסות:

.portNumberLock(1: אובייקט של מנעול threadsDictionary(2): Dictionary שמכיל במבנה את הפורטים הייחודיים שניתנים לשרת שנמצא במצב פאסיבי, והערכים שיכולים להיות להם הם free\occupied בהתאם למצבם.

פונקציות:

1) log(

```
def log(logMessage):
    print("%s" % (time.strftime("%Y-%m-%d %H-%M-%S [-] " + str(logMessage))))
```

פונקציה פשוטה שמקבלת כפרמטר הודעה ומדפיסה אותה הודעה בהמרא string בתוספת שנה, חודש, יום, שער, דקה ושניה באמצעות .time.strftime()

:logCommand(2

```
def logCommand(func, cmd):
    logmsg: str = time.strftime("%Y-%m-%d %H-%M-%S [-] " + func)
    print("\033[31m%s\033[0m: \033[32m%s\033[0m" % (logmsg, str(cmd)))
```

פונקציה פשוטה ויפה שמקבלת כפרמטרים את func ו cmd (شمגייעים מפונקציית החן של thread), ומדפיסה בפורמט יפה לעין שמצאנו באינטרנט את cmd בירוק ואת func באדום.(בתוספת שנה, חודש, יום, שער, דקה ושניה באמצעות (.time.strftime())

:getFTPPath(3

```
|def getFTPPath(absolutPath):
    # remove the c:\ or other drive from the absolutPath
    relativeToRootPath = os.path.relpath(absolutPath, "/")
    # replace all backslash into forward slash (windows slash to linux slash)
    relativeToRootPath = relativeToRootPath.replace('\\\\', '/')
    # add the root slash so it will make the final path as absolute ftp server path
    relativeToRootPath = "/" + relativeToRootPath
|    return relativeToRootPath
```

פונקציה שתפקידה לקבל נתיב אבסולוטי של קובץ\תיקייה ולהחזיר אותו כנתיב שמותאם לנתיב ftp(נתיב שמותאם לינוקס).

דוגמא: temp/111/my.txt/ ← c:\temp\111\my.txt

:getCurrentMilliseconds(4

```
|def getCurrentMilliseconds():
|    return round(time.time() * 1000)
```

פונקציה שמחזירה את הזמן הנוכחי במיili-שניות.

:generateUniqueThreadName(5

```
|def generateUniqueThreadName():
|    randomThreadNumber = random.randint(0, 9999)
|    currentMilli = getCurrentMilliseconds()
|    uniqueThreadName = "Th-" + str(currentMilli) + str(randomThreadNumber)
|    return uniqueThreadName
```

פונקציה שתפקידה ליצור שם ייחודי לכל thread.

השם של thread מורכב ממספר רנדומלי בין 0-9999 שומר לstring בשרשור הזמן הנוכחי במיili-שניות שומר לstring.

:getPortFromPool(6

```
def getPortFromPool():
    with portNumberLock:
        for currentPort in portsDictionary:
            if portsDictionary[currentPort] == "free":
                portsDictionary[currentPort] = "occupied"
                return currentPort
```

הfonקציה נעזרת באובייקט שנקרא מנעול שטתי שבלזמן נתון רק thread אחד ויחיד יכול לגשት יכול לגשት למערך ה포רטים.

הfonקציה עוברת בלולאה על מערך הפורטים ומחפש פורט משוחרר. כשהיא מוצאת אחד כזה היא משנה את ערכו ב-dictionary ל-false ומחזירה אותו.

:returnPortToPool(7

```
def returnPortToPool(portNumber):
    with portNumberLock:
        portsDictionary[portNumber] = "free"
```

הfonקציה מקבלת כפרמטר מספר פורט ובעזרת אובייקט שנקרא מנועל מוצאת את הפורט המופיע ב-dictionary ומשנה את ערכו ל-free.(משחררת את הפורט המשימוש).

*** פונקציות 3-8 יצרנו על סמך מתודת ה `(stat)` של `os` שמצאנו מידע עליה באינטראקטיבית ***

:getFileType(8)

```
def getFileMode(filepath):
    fileStat = os.stat(filepath)

    # init the fileModeString with empty string
    fileModeString: str = ''

    # get the file/folder stat mode
    fileMode = fileStat.st_mode

    # if this file/folder is a dir then change fileModeString to start with d (for directory)
    if (fileMode & stat.S_IFDIR) > 0:
        fileModeString = 'd'
    else:
        fileModeString = '-'

    # if USER has a read permission for this file/folder then add R to fileModeString
    if (fileMode & stat.S_IRUSR) > 0:
        fileModeString = fileModeString + 'r'
    else:
        fileModeString = fileModeString + '-'

    # if USER has write permission for this file/folder then add W to fileModeString
    if (fileMode & stat.S_IWUSR) > 0:
        fileModeString = fileModeString + 'w'
    else:
        fileModeString = fileModeString + '-'
```

תפקידו של הפונקציה הוא להחזיר את מוד ההרשות של הקובץ/תיקייה כבגרות. המוד מורכב מ10 אותיות, כאשר האות הראשונה מייצגת אם זה תיקייה או קובץ. שאר 9 האותיות האחירות מייצגות את הרשותות הקירה, כתיבה והרצה של המשתמש `.user` והutzer `.group`.

לדוגמא: "drwxr--r--"

הפונקציה מקבלת נתיב של קובץ.

מודדר להיות הסטטוס של הקובץ בתיב שמתקבל על ידי `(os.stat())`.
מאתחלים סטרינג ריק. מגדרים משתנה בשם `fileMode` שמקבל את הסטטוס מוד של `.fileStat`.

לאחר מכון מבצעים פעולה `AND` עם `fileMode` על כל אחד מה קונסטנטות שוגדרות לגבי הרשותות גישה.

אם הפעולה AND נותנת ערך גדול מ0 סימן `shlexuser` יש הרשאה ומוסיף את האות המתאימה, אם לא אז מוסיפים "-".

לכן מבצעים תנאים על כל אחד מהקונסטנטות ובסוף משרירים הכל ליטריניג שמייצג את הרשאות המשתמש על הקובץ\ספריה.

:getFilesNumber(9

```
def getFilesNumber(filepath):
    fileStat = os.stat(filepath)
    return str(fileStat.st_nlink)
```

פונקציה שמספקת את מספר הلينקים לקובץ קיים מומר `long, string`, لكن מקבלת נתיב של קובץ.

במערכת כמו לינוקס לקובץ יכולם להיות מספר שמות, וכל השמות הללו מקשרים לאותו הקובץ, אך הפונקציה הזאת מיועדת בעיקר לינוקס.

:getUser(10

```
def getUser(filepath):
    fileStat = os.stat(filepath)
    return str(fileStat.st_uid) # pathToFile.owner()
```

פונקציה שמחזירה את ה`ID user` של קובץ מומר `long, string`, אך מקבלת נתיב של קובץ.

:getGroup(11

```
def getGroup(filepath):
    fileStat = os.stat(filepath)
    return str(fileStat.st_gid) # pathToFile.group()
```

פונקציה שמחזירה את ה`ID group` של קובץ מומר `long, string`, אך מקבלת נתיב של קובץ.

:getSize(12)

```
|def getSize(filepath):
    fileStat = os.stat(filepath)
    return str(fileStat.st_size)
```

פונקציה שמחזירה את גודל הקובץ מומר לstring, לבן מקבלת נתיב של קובץ.

:getLastTime(13)

```
|def getLastTime(filepath):
    fileStat = os.stat(filepath)
    return time.strftime('%b %d %H:%M', time.gmtime(fileStat.st_mtime))
```

פונקציה שמחזירה את הזמן האחרון שבו נעשו שינויים בקובץ\תיקייה מומר לstring, בפורמט של חודש, יום, שעה ודקה.

:fileProperty(14)

```
|def fileProperty(filepath):
    return getFileMode(filepath) + ' ' + \
        getFilesNumber(filepath).rjust(4) + ' ' + \
        getUser(filepath).rjust(4) + ' ' + \
        getGroup(filepath).rjust(4) + ' ' + \
        getSize(filepath).rjust(12) + ' ' + \
        getLastTime(filepath).rjust(12) + ' ' + \
        os.path.basename(filepath)
```

פונקציה שמקבלת נתיב של קובץ\תיקייה ומחזירה שרשרת של string של פרטי הקובץ\תיקייה הבאים:

מוד הקובץ, מספר הלינקים של הקובץ, ה-ID user, ה-ID group, גודל הקובץ, שינויים אחרונים שנעשה בקובץ ואת השם עצמו של הקובץ(לא כל הנתיב).

ftpExceptions

```
class UserNotAuthenticatedException(Exception):  
    '''User is not authenticated - please login'''
```

יצרנו `Exception` משלנו שיורש ממחלקת `Exception` כדי להתאים אותו לשגיאה שקשורה לכישלון באוטנטיקציה של `user`.

השגיאה מדפיסה שה`user is not authenticated` והוא צריך להתחבר בשנית.

מגישים:

יובל דהן – 208720383, רוי ווסקר - 208957084