

CONGESTION CONTROL TCP

ליאור וינמן יועד תמר
213081763 213451818

24 בדצמבר 2022

תוכן עניינים

1	פרק א' - תיאור המערכת	2
1.1	הקדמה	2
1.2	קבצי הפרוייקט והרצתו	2
1.3	ממשקי המשתמש	3
1.3.1	ממשק ה- <i>Sender</i>	3
1.3.2	ממשק ה- <i>Receiver</i>	5
1.4	הקובץ היוצר - <i>Makefile</i>	8
1.5	קובץ ספריה - <i>myLib.h</i>	10
1.6	קובץ הטקסט - <i>msg.txt</i>	12
1.7	קובץ השולח - <i>Sender.c</i>	13
1.8	קובץ המקבל - <i>Receiver.c</i>	13
2	פרק ב' - המחקר על אלגוריתמי בקרת העומס	13
2.1	0 % איבוד חבילות	13
2.2	01 % איבוד חבילות	13
2.3	51 % איבוד חבילות	13
2.4	02 % איבוד חבילות	13
3	ביבליוגרפיה	13

1 פרק א' - תיאור המערכת

1.1 הקדמה

בפרק זה נעסוק בתיאור כלל המערכת שבנינו. נתאר אילו קבצים משתתפים בפרויקט, איך כל אחד מהם ממומש ומה הרציאות והמשמעות מאחורי כל אחד מהם. בנוסף, נסביר כיצד להריץ את התוכנית.

1.2 קבצי הפרויקט והרצתו

את הפרויקט יש להריץ על מערכת ההפעלה UBUNTU LTS גרסה 22.04 בלבד! הפרויקט מכיל את הקבצים להן:

1. *Makefile* - הקובץ היוצר של התוכנית.

2. *myLib.h* - קובץ ספריה.

3. *Sender.c* - שולח ההודעות.

4. *Receiver.c* - מקבל ההודעות.

5. *msg.txt* - ההודעה שאותה השולח ישלח למקבל.

על כל אחד מהקבצים נרחיב בהמשך(!) את כלל הקבצים האלו, יש להוריד ולשמור באותה התקיה. ישנה אפשרות להוריד ישירות מה-GITHUB, בקישור הנ"ל:

https://github.com/liorvi35/TCP_CongestionControl_C

לאחר ההורדה והשמירה של כלל קבצי הפרויקט, נפתח חלון טרמינל במיקום של התקיה שבה נמצאים כל הקבצים ונכתוב את הפקודה: *"make all"*. לאחר שכתבנו את הפקודה הנ"ל יוצרו בתקיה הקבצים הבאים:

1. *myLib.h.gch* - קובץ ספריה מהודר.

2. *Sender.o* - קובץ אובייקט בינארי של *Sender.c*, ז"א זהו הקובץ המתקבל לאחר הידור שלו.

3. *Receiver.o* - קובץ אובייקט בינארי של *Receiver.c*.

4. *Sender* - קובץ הרצת התוכנית של *Sender.c*, ז"א ההרצה מתבצעת על ידי הקובץ הנ"ל.

5. *Receiver* - קובץ הרצת התוכנית של *Receiver.c*.

כעת לאחר שסיימנו את תהליך היצירה, נפתח שני חלונות טרמינל על מיקום התקיה עם כלל הקבצים (ניתן להשתמש בטרמינל שבו יצרנו את הקבצים, אך רצוי קודם לנקות אותו עם הפקודה *"clear"*). בטרמינל הראשון נכתוב את הפקודה *"./Receiver"* ובטרמינל השני נכתוב את הפקודה *"./Sender"* (סדר הפקודות קריטי, כיוון שעלינו קודם כל להפעיל את השרת - מקבל ההודעות לפני שניתן להפעיל את הלקוח - שולח ההודעות).

לאחר שהרצנו את שתי הפקודות האלו, בחלון אחד רצה התוכנית *Receiver.c* ובחלון השני רצה התוכנית *Sender.c* (בשתייהן נפתח ממשק המשתמש וניתן להכניס קלט ולקבל פלט בהתאם). נחזור בקצרה על הדברים שנאמרו ונסכם:

• קבצי הפרויקט הם: *Makefile*, *myLib.h*, *Sender.c*, *Receiver.c*, *msg.txt* יש להוריד ולשמור אותם באותה התקיה.

• נפתח שני טרמינלים על התקיה עם הקבצים, רצף הפקודות הבא מריץ את הפרויקט:

1. בטרמינל הראשון נרשום *"make all"* ולאחר מכן *"clear"*, הדבר יגרום ליצירת הפרויקט וניקוי הפלט הסטנדרטי.

2. בטרמינל הראשון נרשום *"Receiver"*, הדבר יגרום להרצת התוכנית של מקבל הקבצים.

3. בטרמינל השני (על שניהם להיות פתוחים במקביל!) נרשום *"Sender"*, הדבר יגרום להרצת התוכנית של שולח הקבצים.

4. כעת נפתח בפנינו ממשק המשתמש, ניתן להזין קלט ולקבל פלט רצוי.

1.3 ממשקי המשתמש

לאחר שביצענו את כלל השלבים מהנושא הקודם, כעת נרצה להשתמש בתוכנית (עד כה רק הרצנו...). בשתי התוכניות נפתח בפנינו ממשק נוח למשתמש אשר מדפיס למשתמש קלט ברור על הפעולות (רק אלו הרלוונטיות לידיעת המשתמש) שקרו וכמובן גם מבקש מאיתנו קלט על מנת להמשיך את הרצת התוכנית. בנושא זה, נסקור את הממשקים של שתי התוכניות הרצות (שהן, תזכורת: *Sender* ו-*Receiver*) ונראה כיצד ניתן להגיע ולקבל את הפלט הרצוי.

1.3.1 ממשק ה-Sender

מיד בעת הפעלת התוכנית, הקובץ כבר מבצע מספר רב של פעולות (הפעם הראשונה מבוצעת אוטומטית והחל מהפעם השניה התוכנית תשאל אותנו האם ברצוננו לחזור על פעולות אלו). פלט הממשק להלן:

נעבור על הפלט שאנו רואים בתמונה:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_c$ ./Sender
file opened!
socket created!
connected to receiver!
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)

```

1. ראשית, אנו רואים כי הקובץ שאנו שולחים למקבל נפתח בהצלחה.
2. אנו רואים כי נוצר שקע התקשורת (*socket*) בהצלחה.
3. השולח התחבר בהצלחה למקבל.
4. החלק הראשון של הקובץ נשלח.
5. ממתיין לקבלת הסימון המוסכם לאישור (*authentication*) מהשרת.
6. קיבל את האישור והוא אכן נכון.
7. שונה האלגוריתם לבקרת העומס (*CC algorithm*) מ-*cubic* ל-*reno*.
8. החלק השני של הקובץ נשלח.
9. שאלה למשתמש - האם לחזור על התהליך הנ"ל ולשלוח את הקובץ מחדש?
10. המתנה של המשתמש להזנה של התו המתאים. יש להזין 'y' אם ברצוננו לשלוח מחדש ויש להזין 'n' אם ברצוננו לצאת.

נציין כי שורות 1-3 הן שורות אשר יהיו מודפסות רק בעת ההפעלה הראשונה (שהריי אם כבר הקובץ נפתח, השקע נוצר והשולח מחובר למקבל - אין צורך לחזור על פעולות אלו). כעת נעבור על שתי האפשרויות המתקבל (לאחר מכן נראה גם פלטים בהתאמה בממשק של המקבל).

אם המשתמש ביקש לצאת מהתוכנית (ז"א הזין את התו 'n') אזי נקבל את הפלט הבא:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$ ./Sender
file opened!
socket created!
connected to receiver!
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
n
exit message has been sent
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$
```

נשים לב כי קיבלנו פלט שאומר שנשלחה הודעת יציאה ולאחר מכן התוכנית הסיימה, במילים אחרות, אם המשתמש מזין 'n' אזי הלקוח שולח הודעת יציאה לשרת, סוגר את התקשורת ומסיים לרוץ.

כעת נסתכל על המקרה השני, אם המשתמש ביקש לשלוח את הקובץ מחדש (ז"א הזין את התו 'y'), נקבל את הפלט הבא:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$ ./Sender
file opened!
socket created!
connected to receiver!
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
y
CC algorithm has been changed to: "cubic"
first half has been sent!
waiting for authentication...
authentication match!
CC algorithm has been changed to: "reno"
second half has been sent!
send file again? (y/n)
```

נעבור על השורות שקיבלנו לאחר שהמשתמש הזין 'y':

1. שונה אלגוריתם בקרת העומס מ-"reno" ל-"cubic".
2. החלק הראשון של הקובץ נשלח.
3. ממתין לקבלת הסימן המוסכם מהשרת.
4. קיבל את האישור והוא אכן נכון.
5. שונה האלגוריתם לבקרת העומס מ-"cubic" ל-"reno".
6. החלק השני של הקובץ נשלח.
7. כעת, ישנה חזרה על התהליך - ישנה שוב שאלה למשתמש האם לשלוח מחדש את הקובץ או האם לצאת מהתוכנית.

ספווילר: כך התוכנית ממשיכה לעבוד בכל בחירה של המשתמש, כלומר אם המשתמש מזין כעת 'y' הלקוח שוב משנה את האלגוריתם, שולח את החלק הראשון, מחכה ומקבל אישור, משנה את האלגוריתם, שולח את החלק השני ולאחר מכן שוב שואל את המשתמש האם לחזור על התהליך ואם המשתמש מזין כעת 'n' אזי תישלח הודעת יציאה לשרת, התקשורת תיסגר והתוכנית תיעצר. בזאת סיימנו לפרט על הממשק של השולח.

1.3.2 ממשק ה-Receiver

כעת, בנושא זה, נעבור על הממשק של המקבל ונראה בהתאמה כיצד הפעולות שהשולח מבצע משפיעות על המקבל בצורה ישירה. הפלט המתקבל מהרצת השרת:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$ ./Receiver
socket created!
socket bound!
waiting for connection...
```

נשים לב כי הודפסו שלוש שורות, נעבור עליהן:

1. שקע התקשורת נוצר בהצלחה.
 2. שקע התקשורת נקשר לכתובת IP (אנו משתמשים בכתובת המארח המקומי שהיא "127.0.0.1") ונקשר לכתובת PORT (אנו משתמשים במספר "8395").
 3. השרת ממתין לחיבור מהמשתמש.
- שוב, נציין כי שלוש השורות האלו יודפסו רק פעם אחת ויחידה והיא בתחילת הרצת המקבל. כעת, נפעיל את השולח ונראה כיצד הדבר משפיע על המקבל:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
```

נעבור על השורות החדשות שהודפסו:

1. השולח התחבר בהצלחה.
2. החלק הראשון של הקובץ התקבל.
3. נשלח האישור על קבלת הקובץ.
4. שונה האלגוריתם לבקרת העומס מ-"cubic" ל-"reno".
5. החלק השני של הקובץ התקבל.
6. שונה אלגוריתם בקרת העומס מ-"reno" ל-"cubic".
7. השרת ממתין לאישור להמשך התהליך (או סיומו) מהמשתמש.

כעת, ברצוננו להסתכל מה קורה לשרת כאשר המשתמש מבקש לצאת (ז"א מזין 'n':

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
exit message has been received!

#####

collected DATASET is:

time to receive part 1 of file 1 is 47903 [us]
time to receive part 2 of file 1 is 300 [us]

#####

the times of avg part:
the average of sending first file - by "cubic" - is: 47903.000 [us]
the average of sending second file - by "reno" - is: 300.000 [us]

#####
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$
```

נעבור על השורות החדשות:

1. התקבלה הודעת יציאה מהלקוח.
 2. הודפסו הזמנים שלקח לקבל כל חלק מהלקוח.
 3. הודפסו הזמנים הממוצעים לקבל כל אחד מהחלקים.
- כלומר, אם המשתמש מבקש לצאת גם השרת מסיים לעבוד ופולט את המידע שאסף מאז תחילת תהליך ההרצה (להלן *Dataset*).
- כעת נראה מה קורה כאשר המשתמש מבקש לשלוח שוב את הקובץ:

```
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_C$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
```

נשים לב כי קורה בדיוק אותו תהליך כמו בתחילת ההרצה. כעת נניח כי המשתמש ביקש לצאת, נראה מה קורה מבחינת הדפסות הזמנים:

```

liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$ ./Receiver
socket created!
socket bound!
waiting for connection...
sender connected!
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
first message has been received!
authentication has been sent!
CC algorithm has been changed to: "reno"
second message has been received!
CC algorithm has been changed to: "cubic"
exit message has been received!

#####

collected DATASET is:

time to receive part 1 of file 1 is 43014 [us]
time to receive part 2 of file 1 is 273 [us]
time to receive part 1 of file 2 is 335 [us]
time to receive part 2 of file 2 is 249 [us]

#####

the times of avg part:
the average of sending first file - by "cubic" - is: 21674.500 [us]
the average of sending second file - by "reno" - is: 261.000 [us]

#####
liorvi35@liorvi-VirtualBox:~/Desktop/TCP_CongestionControl_CS$

```

נשים לב כי קורה כאן אותו תהליך כמו בפעם הראשונה כאשר המשתמש ביקש לצאת אבל הפעם, מודפסים יותר זמנים (שהריי הפעם, הקובץ נשלח פעם נוספת).

ספויילר נוסף: גם כאן התוכנית ממשיכה לעבוד באותה דרך כמו שראינו כעת, כלומר, אם המשתמש מזין ' n ' אזי השרת מקבל הודעת יציאה, מדפיס זמנים לקבלת כל חלק ואת הזמנים הממוצעים ומסיים לעבוד. ואם המשתמש מזין ' y ' (נניח שהמשתמש הזין ' y ' מספר $t \in \mathbb{N}$ כלשהו של פעמים) אזי יודפסו $2 \cdot t$ שורות של זמנים לקבלת החלקים של t הקבצים שנשלחו ועוד שתי שורות להדפסת הזמנים הממוצעים.

בזאת סיימנו לפרט על הממשק של המקבל.

1.4 הקובץ היוצר - Makefile

נחזור חזרה לבסיס, הקוד אשר כתבנו נכתב בשפה (העילית) C. המחשב אינו יודע ישירות להריץ את קובץ הקוד ולכן עלינו לבצע מספר תהליכים לפני שנוכל בכלל להריץ את התוכנית שכתבנו, התהליכים הללו הם הידור וקישור. נסביר, המחשב (המעבד) כלל אינו יודע מה זו השפה שאנו כותבים בה ומה המשמעות שלה, הוא מבין רק את השפה הבינארית, לכן כדי שנוכל להריץ עלינו לתת למחשב משהו שהוא יודע לעבוד איתו. ישנם שני תהליכים עיקריים אשר קובץ ה-Makefile מטפל בהם והם:

1. הידור (קומפילציה) - תהליך זה ממיר את קבצי שפת התכנות שלנו (קבצים עם סיומות ".c" ו-" .h") לקבצי אובייקטים בשפה הבינארית (קבצים עם סיומת ".o" ו-" .h.gch"), כעת יש בידינו קבצים אשר המחשב יודע לתפעל וניתן להמשיך לעבוד עימם.

2. קישור (לינקוג') - לאחר שהשגנו קבצים בשפה הבינארית, נוכל שלייצר מהם קבצי הרצה לתוכניות שלנו (שכן, כעת המחשב כן מבין אם נגיד לו להריץ קבצים בינאריים) תהליך זה ממיר את הקבצים הבינאריים שהשגנו בתהליך הקודם לקבי הרצה (הערה: במערכת ההפעלה UBUNTU LTS לא לכל קובץ חייבת להיות סיומת, אבל לדוגמה במערכת הפעלה WINDOWS קובץ הרצה חייב להיות בעל סיומת ".exe"), שאותם נוכל להריץ ולהפעיל ולממש איתם יחסי קלט-פלט.

על מנת לייצר בקלות את קבצי ההרצה ולהריץ את התוכנית, יצרנו קובץ הנקרא Makefile ועל ידי פקודת בת שורה אחת הקובץ מריץ (אם ישנו בכך צורך) את שני התהליכים שצינו מלעיל ומייצר לנו בקלות קבצי הרצה. נעבור על הקוד שלו ונסביר כיצד הוא עובד:

```
1  # macros - for more dynamic Makefile
2  CC = gcc # compiler
3  FLAGS = -Wall -g # compilation flags
4
5  # for not creating 'all' and 'clean' files
6  .PHONY: all clean
7
8  # final targets
9  all: Receiver Sender
10
11 # (linkage) making executables from objects:
12 Receiver: Receiver.o
13     $(CC) $(FLAGS) -o Receiver Receiver.o
14
15 Sender: Sender.o
16     $(CC) $(FLAGS) -o Sender Sender.o
17
18 # (compilation) making object files from '.c' and '.h' files:
19 Receiver.o: Receiver.c myLib.h
20     $(CC) $(FLAGS) -c Receiver.c myLib.h
21
22 Sender.o: Sender.c myLib.h
23     $(CC) $(FLAGS) -c Sender.c myLib.h
24
25 # delete all files that created after 'make' command (not deleting '.c' files)
26 clean:
27     rm -f *.o *.h.gch Receiver Sender
```


1. שורות 3 – 2: בשורות אלו אנו יוצרים קיצורי כתיבה ("מאקרו"). זאת על מנת להפוך את הכתיבה שלנו ליותר קריאה ודינאמית. אנו יוצרים מאקרו אחד עבור המהדר (קומפיילר) שישמש אותנו בתהליך ההמרה (אנו משתמשים ב- *Gnu Compiler Collection*, במילים אחרות זהו המהדר הסטנדרטי - *gcc*) ובנוסף, אנו יוצרים מאקרו עבור דגלי הקומפילציה, הדגלים שעזרו לנו נקמפל הם "*Wall, -g*" משמעות הדגל הראשון היא להציג בפנינו לא רק שגיאות קומפילציה אלא גם אזהרות קומפילציה שמהוות סיכון לתוכנית, הדגל השני עוזר למנפה (להלן "*debugger*") למצוא אזהרות ושגיאות. שילוב שני הדגלים האלו מהדר את הקוד שלנו בצורה יותר קפדנית ובכך אנו מקבלים תוכנית תקינה ונקייה יותר.
2. שורה 6: בשורה זו אנו מצהירים כי התוויות (*Makefile* היא שפת תוויות עם משימות לביצוע) "*all*" ו- "*clean*" הן אינן קבצים ואין לייצר קבצים הנקראים כך. הסיבה לכך כי אין לנו צורך ושימוש בהם, אילו רק תוויות לצורך נוחות השימוש (למשתמש - כי הוא מזין בכל פעם סה"כ שורה אחת כדי לבצע הרבה) ועל כן אין לנו באמת צורך, כמו כן אין.
3. שורה 9: בשורה זו אנו יוצרים תווית אשר תשמש אותנו ליצירת שני הקבצים של השולח ושל המקבל, בו זמנית. תווית זו היא למטרת נוחות, כדי שנוכל בפקודה אחת לייצר שני קבצים ולא נצטרך לייצר כל אחד ידנית בנפרד.
4. שורות 14 – 12 ו- שורות 16 – 15: בשורות אלו מתבצע תהליך הקישור, התוויות משתמשות בקובץ אובייקט בינארי של השולח או המקבל ויוצרת לו קובץ הרצה בהתאם.
5. שורות 20 – 19 ו- שורות 23 – 22: בשורות אלו מתבצע תהליך ההידור, התוויות משתמשות בקובץ קוד הכתוב בשפת *C* ובקובץ ספריה, משלבת את שניהם ומקבלת קובץ בינארי.
6. שורות 27 – 26: בשורות אלו אנו יוצרים תווית אשר בעת הרצתה תשמש אותנו למחיקת כל הקבצים אשר נצרו. תווית זו גם היא למטרת נוחות וסדר, כדי שבפקודה אחת נוכל למחוק את כל ה- "בלאגן" שנוצר לנו.

1.5 קובץ ספריה - *myLib.h*

כאשר אנו מתעסקים בתכנות בשפת C, יש לנו כמה וכמה סוגים של קבצים. ראשית, ישנם (מן הסתם) קבצי קוד הכתובים בשפה הנ"ל ומממשים את הלוגיקה ואת הפקודות השונות. עם זאת, אלה לא הקבצים היחידים, ישנם גם קבצי "כותרות"/"ספריות" (*"header"*), אשר מהווים בסיס ויסוד לקבצי הקוד. קבצי הספריות מכילים הצהרות על קבועים והצהרות על פונקציות (הצהרות בלבד) כך שאם נצרף את קובץ הכותרת לקובץ הקוד, הקוד יהיה מודע לכל הקבועים והפונקציות ויוכל להשתמש בהם (אם קיים מימוש), למעשה קבצים אלה הם עיבוד מראש (*"pre-processed"*). כיוון שכדי לממש את הלקוח והשרת, היינו צריכים להשתמש בכלילה של קבצי ספריות רבים ויצירת מספר קבועים שחשוב שיהיו גם בלקוח וגם בשרת, החלטנו ליצור ספריה משלנו על מנת לשמור על הסדר, הנוחות והארגון בקוד. נעבור על הקוד של הספריה שלנו ונסביר את הכתוב:

```
1  /* including libraries */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include <string.h>
6  #include <sys/types.h>
7  #include <sys/socket.h>
8  #include <arpa/inet.h>
9  #include <netinet/in.h>
10 #include <netinet/tcp.h>
11 #include <errno.h>
12 #include <unistd.h>
13 #include <time.h>
14 #include <sys/time.h>
15
16 /* defining constants */
17 #define IP "127.0.0.1" // server's ip address
18 #define PORT 8395 // connection port
19 #define BUFSIZE 8192 // size of the buffer
20 #define AUTH "0000000111111001" // authentication code
21 #define OK "ok" // ready message
22 #define CONNECTIONS 50 // number of clients that server can listen simultaneously
```

בלי הגבלת הכלליות, קודם כל נסביר את הקבועים שהגדרנו:

1. הגדרנו את כתובת ה-IP אשר תהיה בשימוש בעת שליחה לשרת (זוהי הכתובת של השרת והלקוח שולח את המידע לכתוב הזו).
2. הגדרנו את מספר ה-PORT שיהיה בשימוש וישמש לתקשורת בין הלקוח לשרת. המספר 8395 הוא אקראי ואין סיבה מיוחדת לבחירה הזו (במילים אחרות השתמשנו במספר PORT כרצוננו).
3. הגדרנו את גודל החוצץ (*"buffer"*) אשר יהיה בשימוש, החוצץ יחזיק בכל פעם את ההודעות המתקבלות והנשלחות מהלקוח לשרת וההפך.
4. הגדרנו את הסימן המוסכם (האישור) שהמקבל שולח לשרת. הגענו למספר הזה כך: ארבעת הספרות האחרונות של ת"ז של ליאור הן - 1763, נעביר מספר זה לבינארית ונציג בתור 16 ביטים - נקבל: 0000011011100011. ארבעת הספרות האחרונות של ת"ז של יועד הן - 1818 נמיר לבינארית - נקבל: 0100011100000101. כעת, נבצע פעולת *xor* בין שתי התוצאות, נקבל בדיוק: 0000000111111001.
5. הגדרנו הודעת אישור על התחלת התקשורת בין השולח למקבל.
6. הגדרנו הודעת יציאה אשר נשלחת מהשולח למקבל כאשר המשתמש מעוניין להפסיק לשלוח קבצים ולצאת.

7. הגדרנו את המספר המקסימלי של לקוחות שהמקבל יכול להאזין בו זמנית (ספיציפית כאן, נדרש להאזין ללקוח אחד בלבד שהוא השולח אבל באופן כללי בגלל שניתן, הגדרנו יותר).

כעת, נסביר את הספריות הספריות שהשתמשנו בהן:

1. שורות 3 – 1: ספריות קלט ופלט סטנדרטיות - שימוש חובה בכל תוכנית. הספרייה השלישית יוצרת אובייקט חדש עבור בוליאנים, לא השתמשנו בספרייה זו אבל בכל מקרה החלטנו להוסיף כיוון שזוהי ספרייה סטנדרטית ולמקרה שתוכל לבוא לעזרתנו.

2. שורה 5: ספרייה עם פעולות רבות לטיפול במחרוזות, תורמת רבות לנוחות השימוש במחרוזות.

3. שורות 10 – 6 ושורה 12: אלו ספריות עבור תכנות שקעים ("GNIMMARGORP TEKOS"). ספריות חובה בשביל המטלה.

4. שורה 11: זוהי ספרייה בעל אובייקט אשר מחזיק את שגיאת המערכת האחרונה שקרתה, שימושית מאוד כיוון שבמקרים של תקלות החזרנו למערכת ההפעלה את מספר השגיאה (פתרון אלגנטי יותר מאשר להחזיר 1 גנרי).

5. שורות 14 – 13: אלו ספריות אשר מחזיקות זמנים, השימוש שלנו איתן היה בחישוב הזמנים של קבלות החלקים של הקובץ.

1.6 קובץ הטקסט - msg.txt

הקובץ שבחרנו לשלוח מהלקוח לשרת הוא קובץ גנרי לחלוטין המכיל את הספרות 9–2. גודל הקובץ הוא 1.1MB (הקובץ עומד בדרישות כיוון שהוא גדול מ 1MB). תמונה להמחשת הקובץ (כמובן שזה לא כל הקובץ, הוא גדול מידי ולא ניתן להציג את כל כולו):

```
1 2935893598593769278873924793356643562559997779752758933838982328793292872438822644742896253675338947967845
2599546675888755269523362425499853292238586874568989734823997336485476222469564352553234766495786527438526
7967368877378366949262234345223792289386342236467265484734628887823774766595957357437275999825554426397274
5695547876672998352259539637273686264522399888445663298754975745598775825768297679823235496764296966424358
8625397865924357548882476599986266249864522348629695272836372987889327493786929438477973358986626274729638
3632339242582284337478743739923929489395583383954226329329794727565332772869463948588276683379783444663932
9449893625366947627487432946479822785996783383675897743228323387439565552647625639543445658454595426888775
6663247878854622452432654926675752773832987624252279864395995426357679656496355772265368455768947264363447
2833689747884854545246734689644379983679656945722236842369249656287523498358945323229653363753393535684988
3685562593994948323956625987988628982455558765242923786882423972936954792437964597846875992295664243932482
2483893668696538839722273674474349682894787968232686762675932285728644425326732225832895659479994645888725
2998224353868738346837927598436323966294786622948778345725398743397724456958297395357884654439346767799696
5836529386724568458226288472574493997679827375287677468973264933298432626538738337366827797424946654484925
8223468839379556326433875862356652495723662774879438749349436597975997698497826477523255336745723275376966
7938848944844827234986539558599863574636636742692249994347297629854523986998829664939379383484962394897674
8646555593663999858459447737599866495242728882482385259353292963854598558734272934467693438793329526784548
7253649688323459472363225672657755824274586995739342483236524995295749956948338563264682427664293258266744
9434929948848382269268973487699458564772669729856793363383555655368473233662772367567954867557452828822882
7232969668859799945295682858327826466286283495578737329894724467574272963566395674868644378597958425488553
459736353465796666699967476577357847985489432544827632488858646322957826633998552895325923658759625392395
2772857837924544537383447727724855758728234356437648353868869647942445736652384263994336637695629233688653
9426382933582984352877556574354978578849786462488825537737926793573873745594836536346949227728446649482375
4829289458288735387879949946287395292823887838998664397626937697537438837337686627772449936678556575767582
7686587278884286378426568443727923358844399598224536399454576527786556226742662866395538994592983979739794
387368885599688363782576546846552685585846327663526769248826589864643384485484999963644345383286927865259
```

אין יותר מידי מה להסביר בחלק זה, לכן נעבור לחלק הבא.

1.7 קובץ השולח - *Sender.c*

1.8 קובץ המקבל - *Receiver.c*

2 פרק ב' - המחקר על אלגוריתמי בקרת העומס

2.1 0 % איבוד חבילות

2.2 01 % איבוד חבילות

2.3 51 % איבוד חבילות

2.4 02 % איבוד חבילות

3 ביבליוגרפיה