**מטלה 3 – רשתות תקשורת**

מגישים: 314779745,

חלק א:

בחלק זה כתבנו 2 תוכניות, Sender ו-Receiver כאשר שתיהן משתמשות ב-Socket שמבוסס על פרוטוקול TCP.

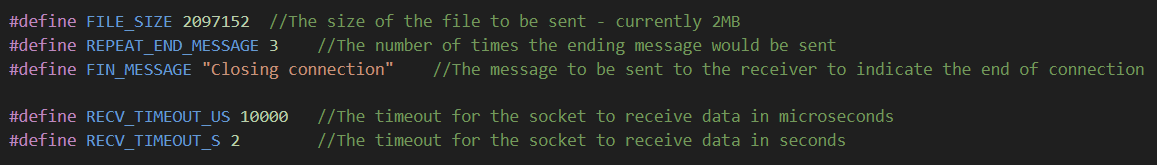
התוכנית TCP\_Sender שולחת ראשית את גודל הקובץ אותו מתכננת לשלוח (כדי שה-Receiver יוכל להקצות זיכרון ל-Buffer שלו כנדרש), לאחר מכן שולחת את הקובץ עצמו. ולבסוף שואלת את המשתמש אם לשלוח את הקובץ שוב. כאשר המשתמש בוחר שלא, התוכנית שולחת הודעת שמצהירה על סיום ההתקשרות עם ה-Receiver.

התוכנית TCP\_Receiver קולטת את הפאקטה הראשונה שמכילה את גודל הקלט ומקצה Buffer בגודל המתאים. לאחר מכן, מקבלת את הקובץ. יתכן שהקובץ ישלח במס' Segments, לכן התוכנית סופרת את מס' הבייטים שהתקבלו וממשיכה "להאזין" עד שהתקבלו מס' הבייטים שהשולח דיווח שישלח.

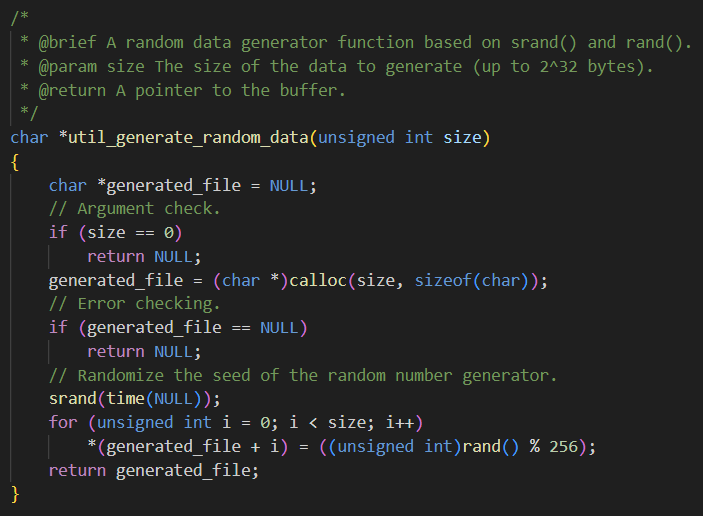
אם כל הקובץ עבר בשלמותו, או שהתקבלה הודעת סיום ה-Receiver ייסגר (וישחרר את הזיכרון שהוקצה במהלך התוכנית).

**הסבר קוד TCP\_Sender.c:**

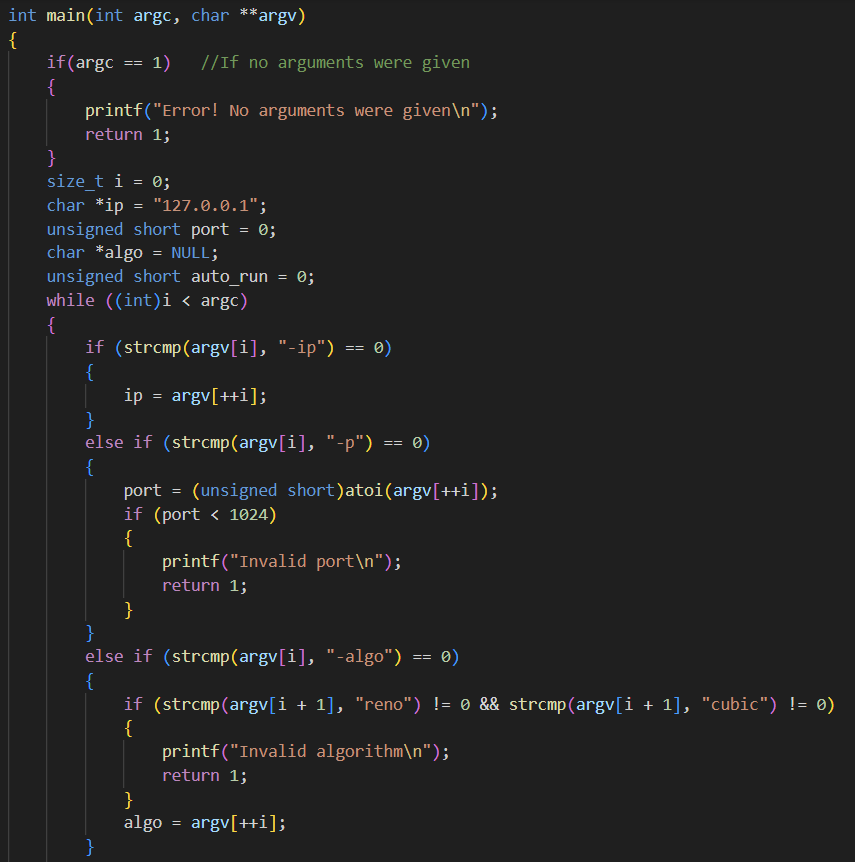
ראשית נגדיר את כל הקבועים בהם נשתמש במהלך התוכנית:



נשים לב שאת הודעת סיום ההתקשרות אנו שולחים מס' פעמים, זאת בשביל להתגבר על מקרה של Packet Loss, כך שאנו מקטינים את הסיכוי שה-Receiver לא יקבל את הודעת הסגירה וישאר פתוח.

פונקציה נתונה שמחזירה כתובת למערך שמכיל תווים רנודמליים בגודל נתון.

נשתמש בפונקציה הזו כדי לייצר את הפלט שנשלח ל-Receiver.



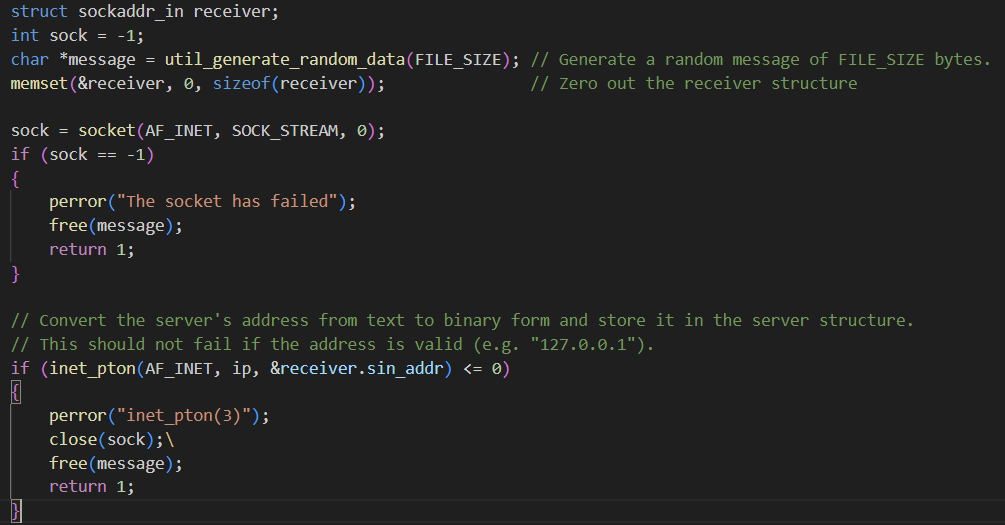
A computer screen with text

Description automatically generated

נבצע Parsing לארגומנטים שנקבל מהמשתמש ונחזיר שגיאה אם הפורט או האלגוריתם לא חוקיים או לא נתונים.

הגדרנו ערך דיפולטיבי ל-IP שיהיה כתובת ה-IP העצמית של המחשב (localhost).

הוספנו בקוד שלנו אופציה להעביר ארגומנט של מס' הפעמים שהשולח ישלח את הפלט על מנת לבצע בדיקה אוטומטית.

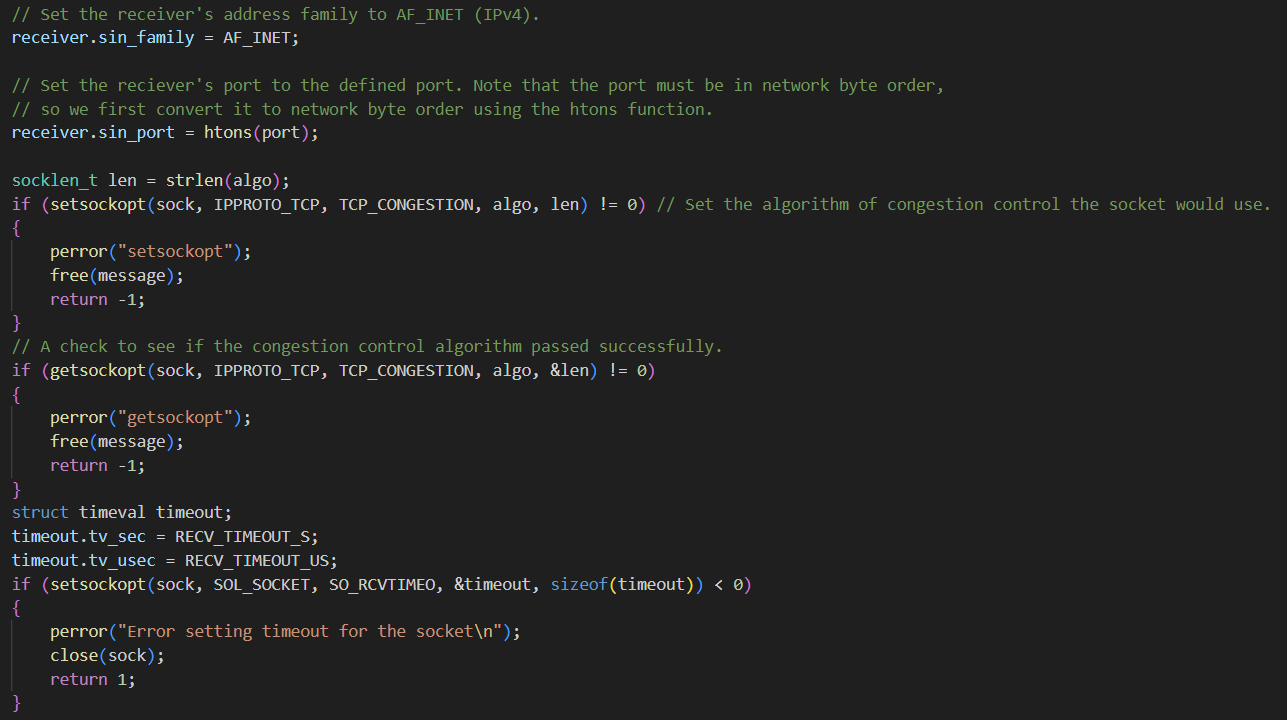


נגדיר socket חדש שמבוסס על IPv4 (AF\_INET) ופרוטוקול TCP (SOCK\_STREAM)

בנוסף נגדיר sockaddr\_in שיכיל את כתובת ה-IP של ה-Receiver.

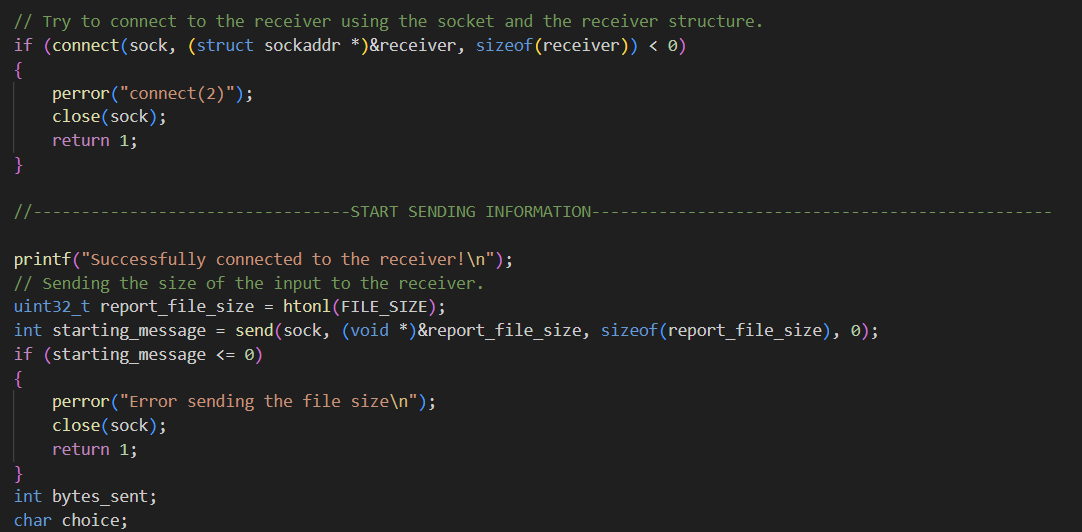
ונגדיר מצביע להודעה שייצרנו.

ניתן לראות שבכל המקומות בתוכנית בהם יתכן שתיהיה שגיאה, אנו בודקים זאת ומסיימים את הריצה במידה ואכן הייתה שגיאה.

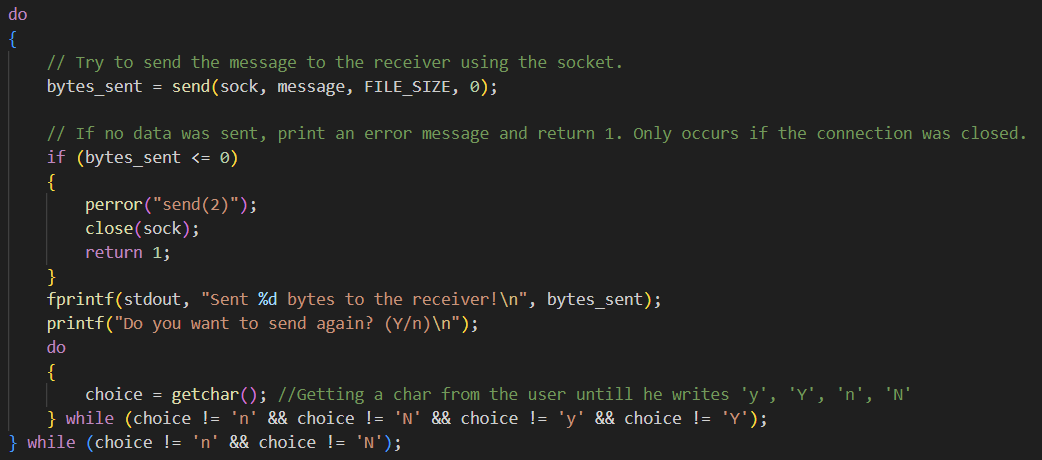


נמשיך בהגדרת ה-Receiver, נגדיר את הפורט שלו ואת סוג הכתובת שלו כ-IPv4.

באמצעות פונקציית setsockopt נגדיר את אלגוריתם ה-congestion control של ה-socket ואת זמן ה-Timeout של ה-socket.

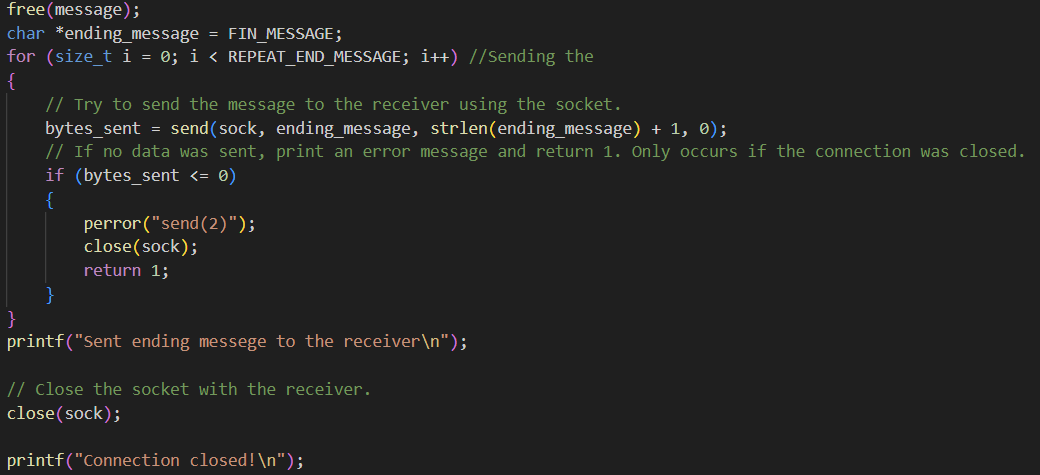


נתחבר ל- Reciver באמצעות ה-receiver sockaddr\_in שיצרנו שמכיל את כתובת ה-Receiver, ונשלח את ההודעה הראשונה שתדווח ל-Receiver על גודל הקובץ אותו נשלח.



נשלח את הקובץ ונשאל את המשתמש האם לשלוח את הקובץ שוב.

נשלח את הקובץ עד שהמשתמש ילחץ על n/N.

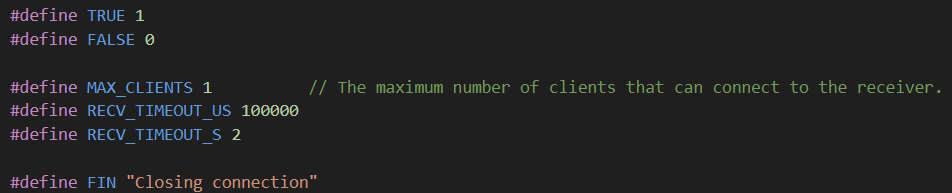
אם יצאנו מהלולאה, המשתמש ביקש לסיים את השליחה. כלומר, ניתן לסיים את התוכנית.

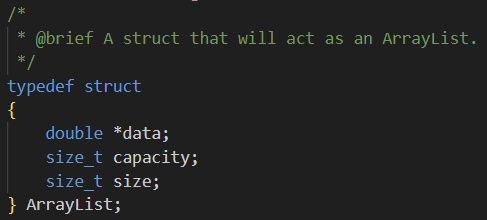
נשחרר את הזיכרון שבו השתמשנו במהלך הריצה ונשלח הודעת סיום (מס' פעמים קבוע, כדי להתגבר על Packet Loss).

לאחר מכן נסגור את ה-socket.

**הסבר קוד TCP\_Receiver.c:**

ראשית נגדיר את כל הקבועים בהם נשתמש במהלך הריצה:

ה-Receiver מכיר את הודעת הסיום כדי שידע איזה הודעה מסמנת את סיום התקשורת.

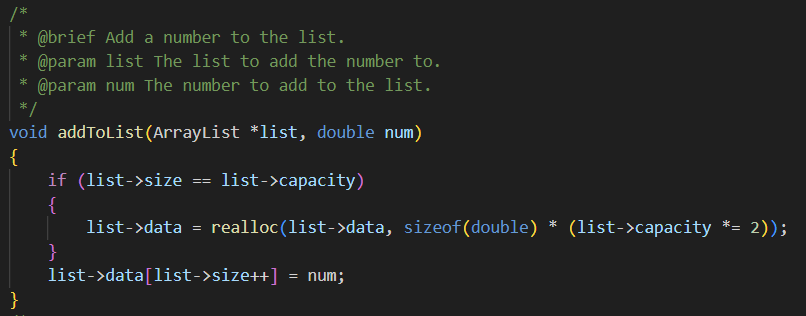


ע"מ לשמור את ערכי הזמן והמהירות של התוכנית, יצרנו struct של מערך דינאמי.

\*data – מצביע לערך המספרי ששמור במערך

Capacity – כמה איברים ניתן להכניס למערך

Size – כמה איברים שמורים כרגע במערך.

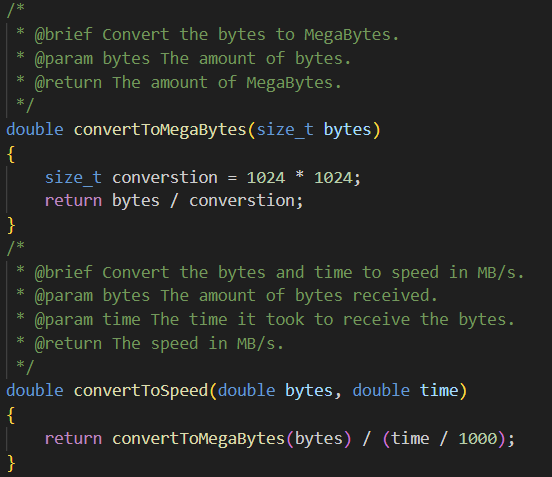


פונקציית הוספת איבר חדש למערך.

המערך מגדיל את עצמו בדיוק כמו מערך דינאמי ב-Java.

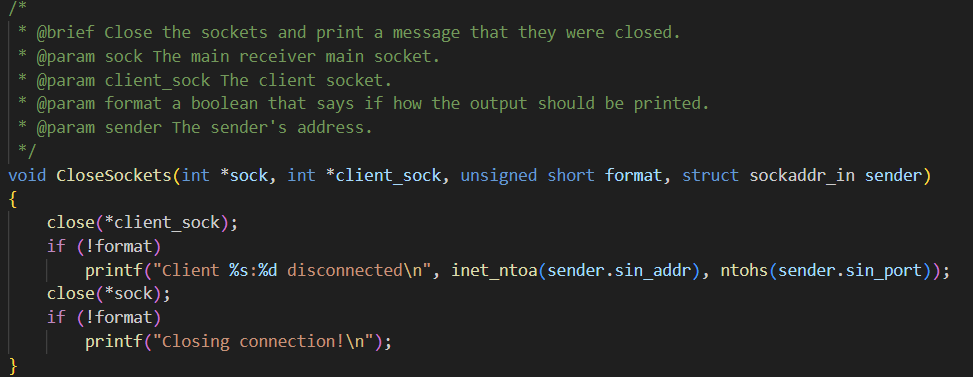
אם כמות האיברים שכרגע במערך זהה לגודל המערך, נגדיל את גודל המערך פי 2.

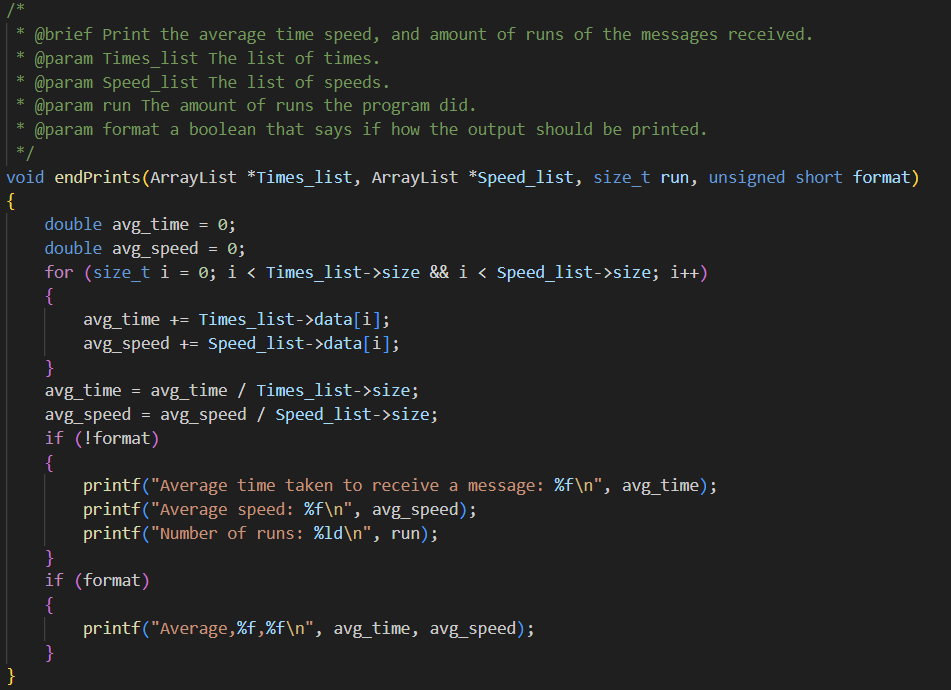
לבסוף, נוסיף את האיבר החדש למקום האחרון.



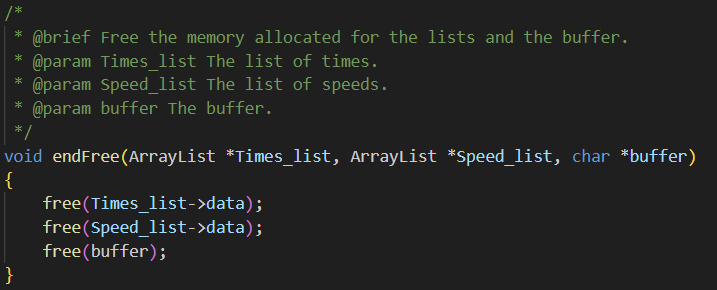
פונקציה שממירה מבייטים ל-MB

ופונקציה שמקבלת זמן וכמות בייטים ומחזירה את המהירות שבה נשלחו אותם בייטים.

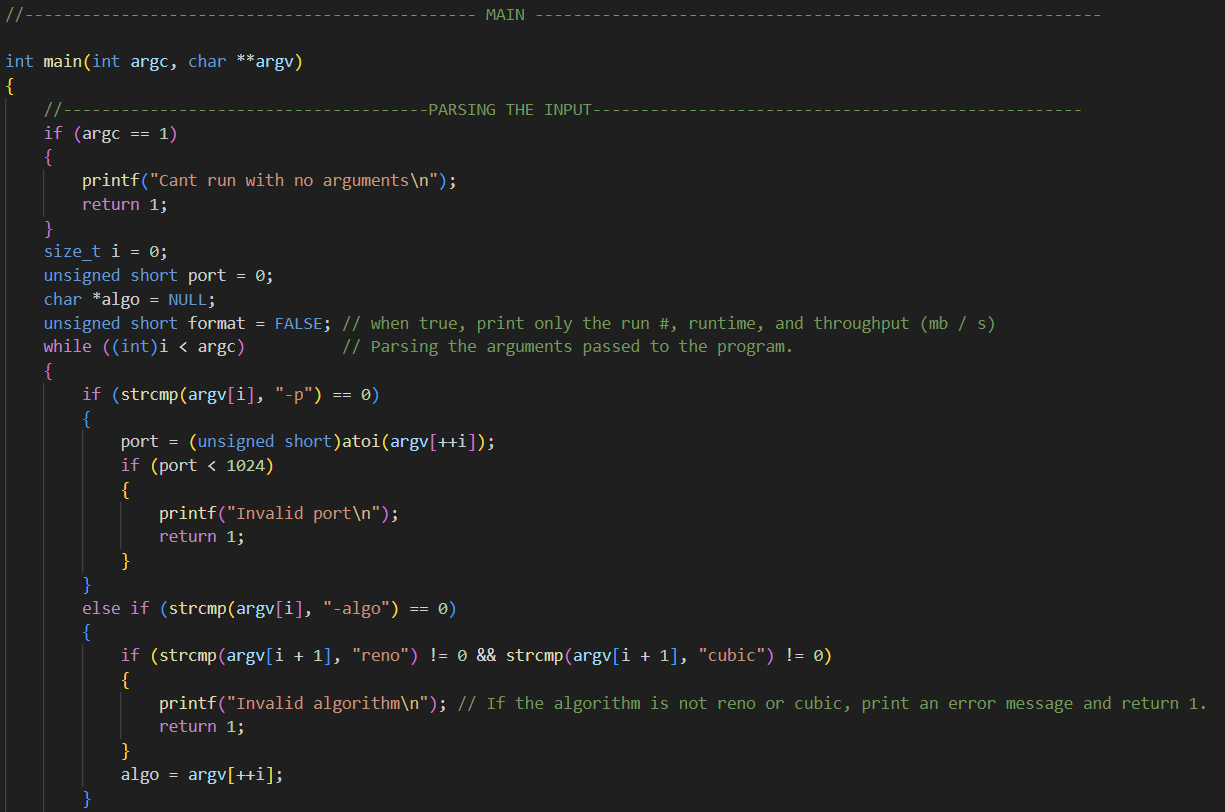
כדי לפשט את הקוד שב-main, הוצאנו את סגירת ה-sockets לפונקציה נפרדת.

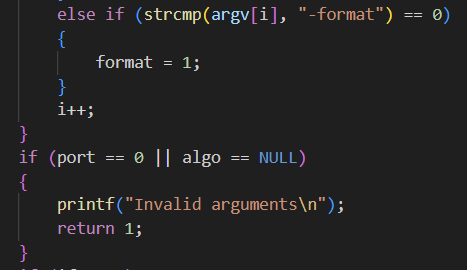


מאותה סיבה גם הוצאנו את כל חישובי והדפסות סיום הריצה לפונ' נפרדת.

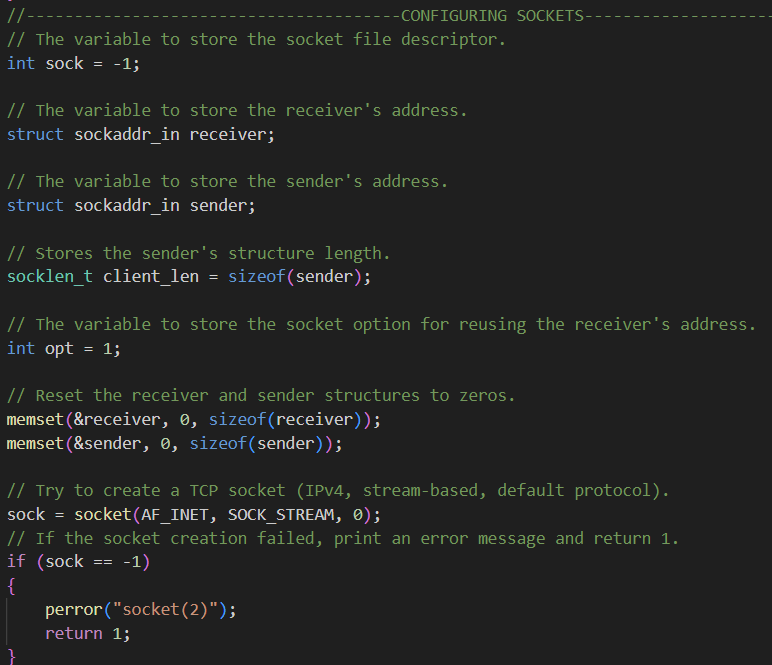
אותו דבר גם על שחרור הזיכרון במידה והייתה שגיאה בריצה או בסיום הריצה.

כעת נעבור ל-main:

כמו שביצענו ב-Sender, נבצע Parsing לארגומנטים של התוכנית.

הגדרנו ארגומנט format שגורם לתכנה להדפיס מידע מצומצם בפורמט CSV לטובת בדיקות אוטומטיות.

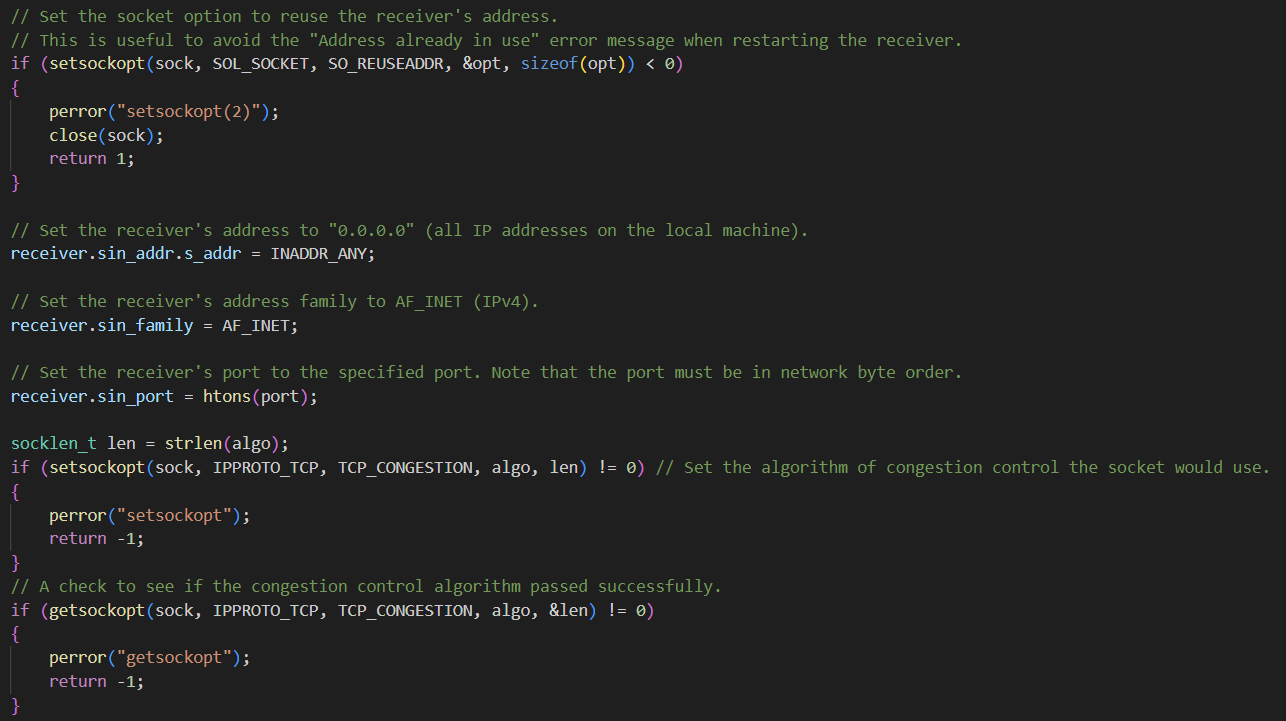
אם לא התקבל פורט או אלגוריתם TCP Congestion, נדפיס שגיאה ונסיים את הריצה.



נגדיר Socket וכתובות sender ו-receiver.

נאפס את 2 הכתובות. ונגדיר את ה-Socket שישתמש בפרוטוקול TCP ויבוסס על IPv4.

נשתמש במשתנה opt בהמשך ע"מ להגדיר שיהיה אפשר להשתמש באותה כתובת עבור ה-receiver מס' פעמים.

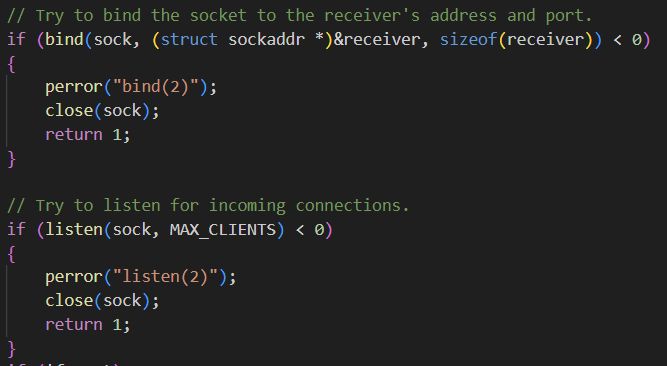


נגדיר שיהיה אפשר להשתמש בכתובת ה-IP של ה-receiver מס' פעמים, זאת כדי להימנע משגיאת

“Address already in use”.

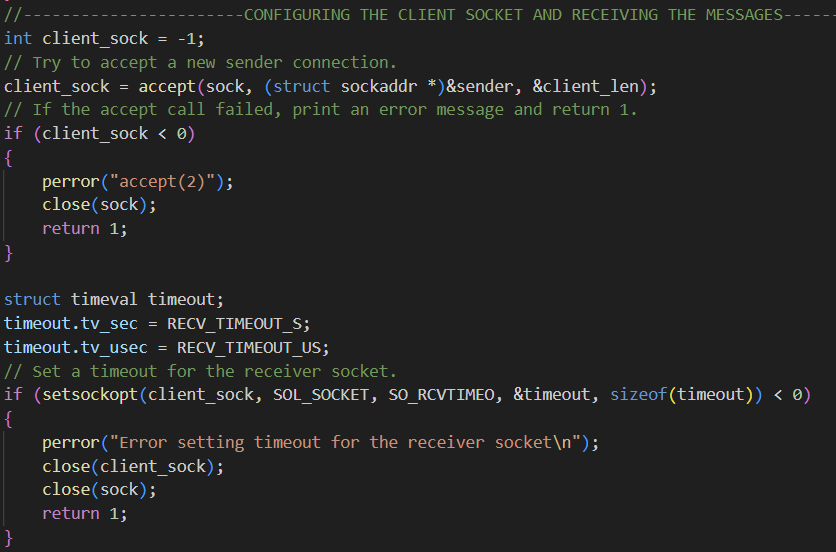
לאחר מכן נגדיר את ה-receiver שיהיה תואם לשולח (פורט תואם ו-IPv4) ונגדיר שנסכים לקבל הודעות מכל IP (INADDR\_ANY).

נגדיר את אלגוריתם ה-Congestion control של ה-socket לאלגוריתם שקיבלנו כארגומנט מהמשתנה (algo) באמצעות פונקציית setsockopt ולאחר מכן נבדוק האם ה-Socket אכן קיבל את הפקודה.



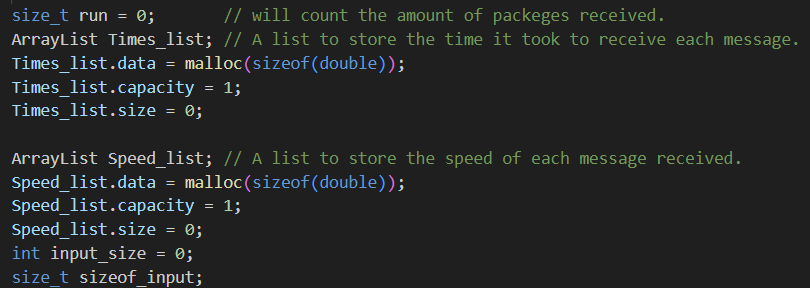
לאחר שסיימנו להגדיר את ה-socket, נבצע עליו bind עם כתובת ה-IP והפורט הנתונים (אותם הכנסנו ל-receiver בשורות מעל).

ולאחר מכן "נאזין" לחיבורים חדשים באמצעות פונקציית ה-listen.



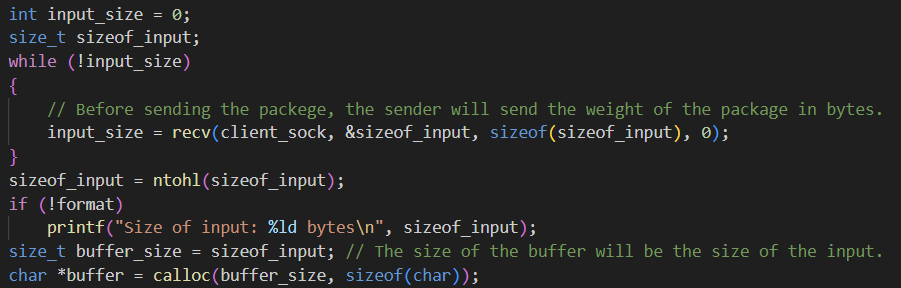
נגדיר Socket חדש של החיבור החדש (שהתקבל בפונקציית accept).

ונגדיר ל-Socket זה Timeout שהוגדר ברשימת הקבועים.



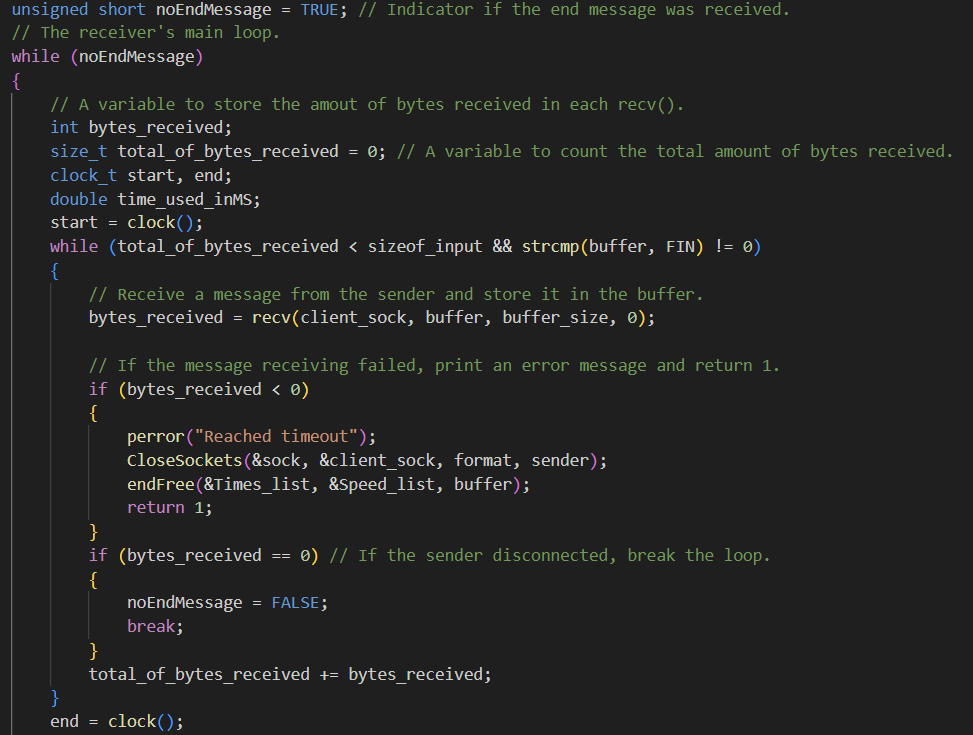
נגדיר משתנה שיכיל את כמות הקבצים שהתקבלו (רק כאשר נקבל את כל הקובץ נגדיל את המשתנה)

נגדיר גם 2 מערכים דינאמיים שיכילו את הזמנים והמהירויות של כל ריצה.

נמתין לקבל את החבילה הראשונה שהיא תכיל את גודל הקובץ שישלח.

נבצע Parsing למידע שקיבלנו, על מנת להשתמש בו בהמשך.

לאחר שקיבלנו את גודל הקובץ, נקצה מקום בזיכרון ל-Buffer בגודל buffer\_size.



נגדיר מעין משתנה "בוליאני" שיסמן אם התקבלה הודעת סיום מה-Sender.

כל עוד לא התקבלה הודעת סיום, נחכה לקבל הודעות חדשות.

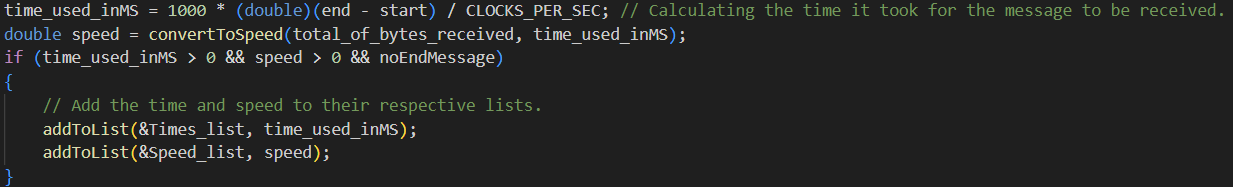
bytes\_received – מכיל את כמות הבייטים שהתקבלו ב-recv יחיד.

total\_of\_bytes\_received – מכיל את סך הבייטים שהתקבלו עד לקבלת כל הקובץ.

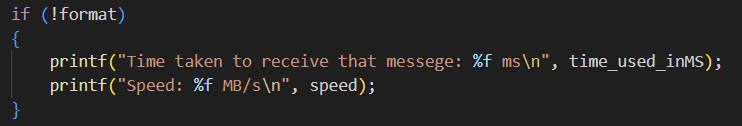
נפסיק לקבל חבילות חדשות מה-Sender אם סך הבייטים שקיבלנו גדול שווה לגודל הקובץ (שנתון לנו) או שהתקבלה הודעת סיום.

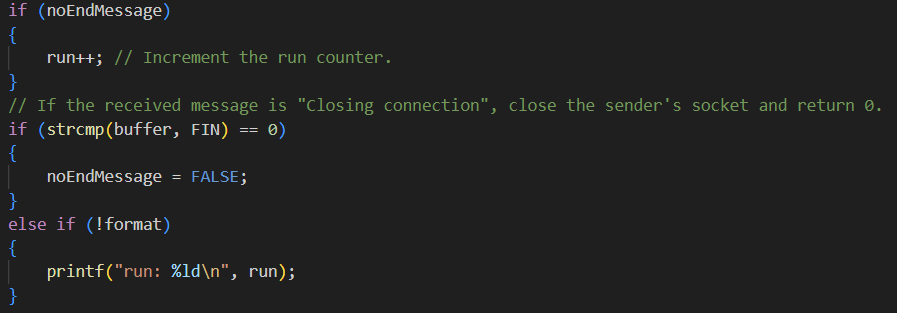
כמו כן, אם קיבלנו 0 בייטים, ה-Sender התנתק. נגדיר מצב זה בתור קבלת הודעת סיום.

נשים לב שגודל הקובץ לא משתנה, לכן נכון להשתמש ב-sizeof\_input שקלטנו בתחילת הריצה, עבור כל הקבצים שנקבל.



נחשב את הזמן שלקח ל-Receiver לקבל את הקובץ, נחשב את המהירות. ונכניס את 2 הערכים למערכים המתאימים.

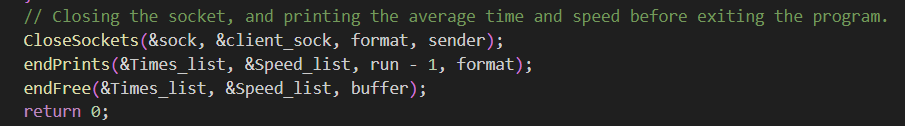
נדפיס את הזמן שלקח לקבל את הקובץ הנוכחי ואת המהירות.



לאחר שקיבלנו קובץ, נבצע את הפעולות הבאות:

נגדיל את מונה הריצות שלנו ב-1.

נבדוק אם אם התקבלה הודעת סיום.



לאחר שהתקבלה הודעת סיום, נסגור את ה-Sockets, נדפיס סטטיסטיקות ונשחרר את הזיכרון שהקצנו במהלך הריצה.

חלק ב:

בחלק זה כתבנו 3 תוכניות, RUDP\_Sender, RUDP\_Receiver, RUDP\_API

כאשר ה-Sender וה-Receiver משתמשות ב-Socket שמבוסס על פרוטוקול RUDP שממומש ב-RUDP\_API.

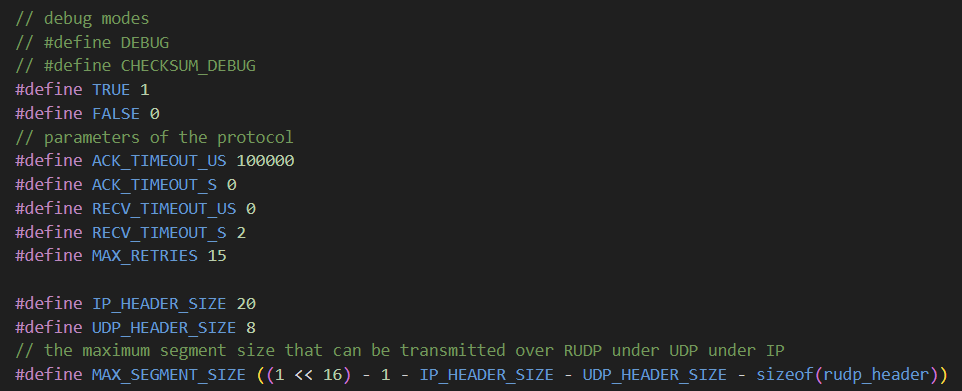
התוכנית RUDP\_Sender שולחת ראשית את גודל הקובץ אותו מתכננת לשלוח (כדי שה-Receiver יוכל להקצות זיכרון ל-Buffer שלו כנדרש), לאחר מכן שולחת את הקובץ עצמו. ולבסוף שואלת את המשתמש אם לשלוח את הקובץ שוב. כאשר המשתמש בוחר שלא, התוכנית קוראת לפונקציית סגירת ה-Socket שמהלכה נשלחת הודעת FIN ל-Receiver.

התוכנית RUDP\_Receiver קולטת את הפאקטה הראשונה שמכילה את גודל הקלט ומקצה Buffer בגודל המתאים. לאחר מכן, מקבלת את הקובץ. ב-RUDP, יתכן שהקובץ ישלח במס' מקטעים אבל פונקציית ה-rudp\_recv מעבירה את התוכן ל-buffer רק כאשר כל התוכן התקבל (מס' הבייטים שהתקבלו שווה לגודל הקובץ). התוכנית מאזינה ל-Sender עד שהוא שולח הודעת FIN.

לבסוף, ה-Receiver יסגר (וישלח הודעת FIN), ישחרר את הזיכרון שהוקצה במהלך התוכנית וידפיס סטטיסטיקות.

חשוב: לפעמים כאשר ב-Makefile מוסיפים את ה-flag -O3 שמגדיר לקומפיילר לבצע אופטימיזציה, קיימות בעיות בחישוב checksum. אם נתקלים בבעיות אפשר למחוק את ה-flag או להוריד אותו לרמת אופטימיזציה נמוכה יותר.

**הסבר קוד RUDP.c:**

ראשית נגדיר את הקבועים שבהם נשתמש בתוכנית.

בנוסף להגדרת קבועים לזמני ה-Timeout של הודעת ה-ACK והודעות אחרות, הגדרנו גם את הקבועים הבאים:

MAX\_RETRIES – כמות הפעמים שננסה לשלוח פאקטה במידה ולא קיבלנו ACK, לפני שנגדיר את השליחה ככישלון.

IP\_HEADER\_SIZE – גודל ה-Header של פאקטה שמשתמשת בפרוטוקול ה-IP.

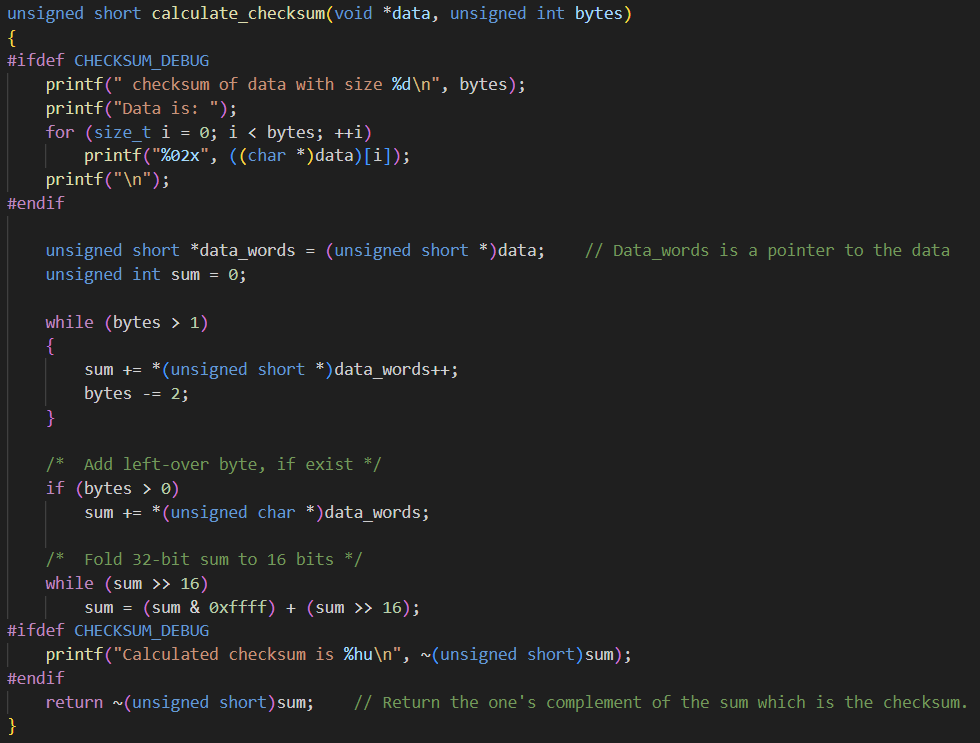
UDP\_HEADER\_SIZE – גודל ה-Header של פאקטת UDP.

MAX\_SEGMENT\_SIZE – הגודל המקסימלי של פאקטה יחידה הוא:

שכן, שדה ה-length ב-header של ה-IPv4 הוא באורך 16 ביטים, כך שהאורך המקסימלי של פאקטה יחידה הוא: .

מתוך מס' זה, נוריד את כל ה-Headerים שאנו מוסיפים להודעה – שתופסים מקום באורך ההודעה: IP Header, UDP Header, RUDP Header. שכן, RUDP משתמש בפרוטוקולי IP ו-UDP ועליהם מוסיף Header חדש מעצמו.

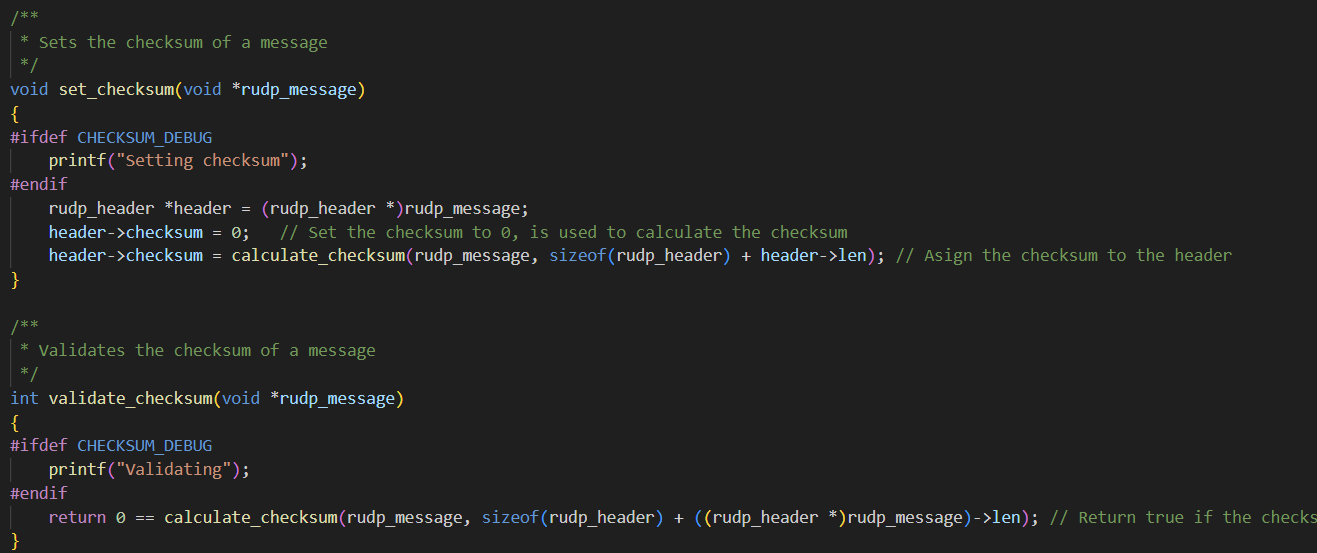
בנוסף הוספנו 2 הגדרות להודעות debug – DEBUG להודעות כלליות ו-CHECKSUM\_DEBUG להודעות בנושא checksum.

פונקציית חישוב ה-Checksum.

הפונקציה מבצעת חישוב של Checksum ה-data לפי החישוב שלמדנו בהרצאה.

bytes – גודל ה-data בבייטים.

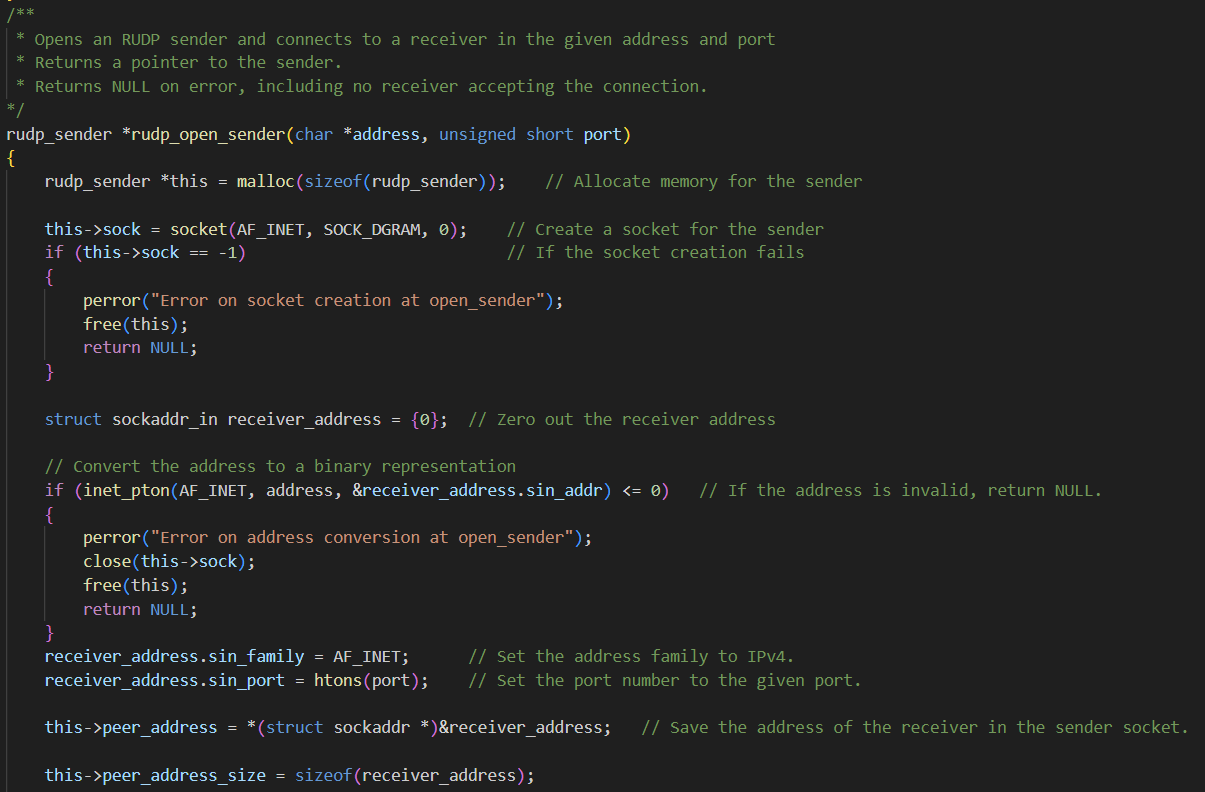
להוסיף מידע

****

set\_checksum – מקבלת הודעת RUDP ומגדירה ל-header שלה את ערך ה-checksum הנדרש.

נשים לב שהגדרנו קודם את ערך ה-checksum ל-0, על מנת שלא יפריע לחישוב.

validate\_checksum - מקבלת הודעת RUDP, מחשבת את ה-checksum שלה ומחזירה “True” אם הוא שווה ל-0 כנדרש.



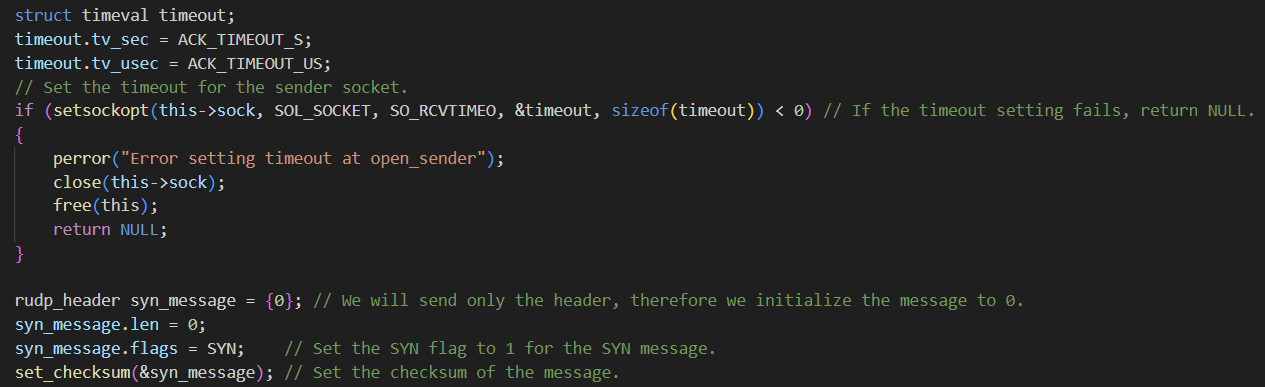
פונקציה לפתיחת RUDP Sender socket.

הפונקציה מקבלת כתובת ופורט ומחזירה מצביע ל-socket של ה-sender.

ראשית נקצה מקום ונגדיר את ה-socket כ-UDP שמבוסס על IPv4 (ועליו "נלביש" את עקרונות ה-RUDP).

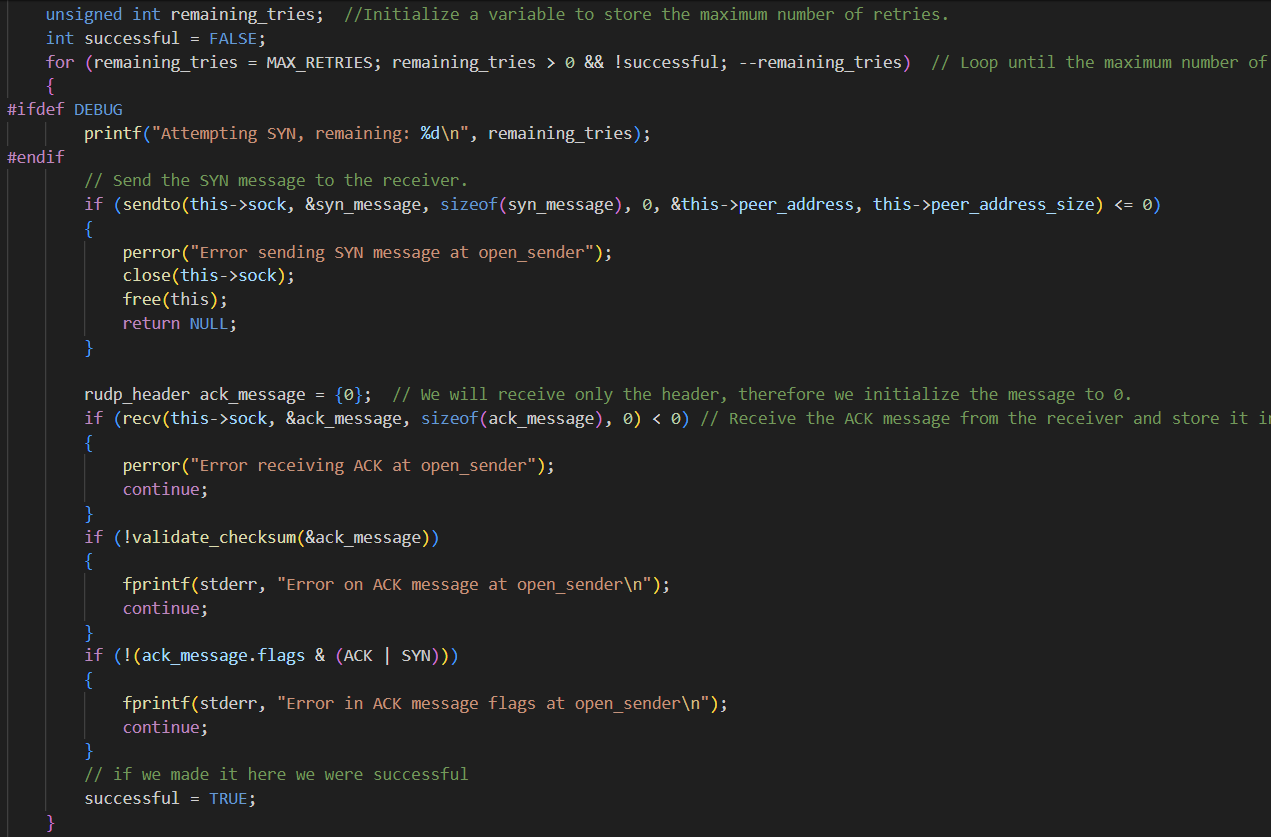
לאחר מכן נקצה את הכתובת הנתונה לתוך ערך הכתובת של ה-receiver, נגדיר כתובת זו ככתובת IPv4 ב-receiver ונקצה את הפורט הנתון ל-receiver.

בנוסף, נשמור את הכתובת ה-receiver בשדה ה-peer\_address של השולח.



נגדיר Receive timeout ל-Sender socket (זמנים שהוגדרו בתור קבועים)

נכין הודעת SYN, בה רק ה-SYN flag דלוק. הודעה זו נשלח בהמשך ל-Receiver כדי להתחיל תקשורת.

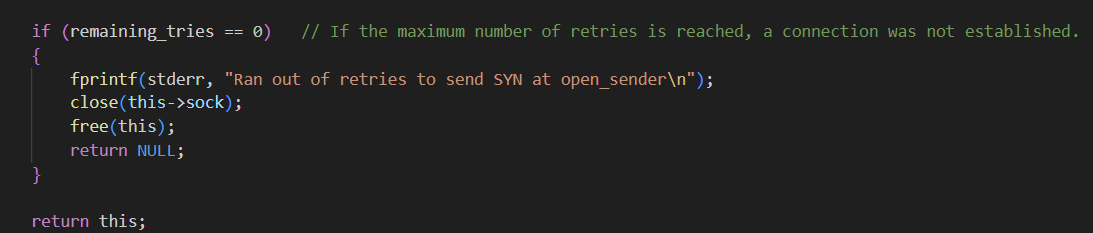


נשלח הודעת SYN ל-Receiver עד אשר נקבל הודעת SYN ACK בחזרה או עד שנגיע לכמות הפעמים המקסימלית שהגדרנו לניסיון תחילת התקשרות.

בקוד ניתן לראות שאנחנו "מכינים" הודעת ACK, מקצים את ההודעה שקיבלנו מה-Receiver לכתובת של הודעת ה-ACK, בודקים את אמיתות ה-Checksum שלה ומוודאים שזו אכן הודעת SYN ACK.

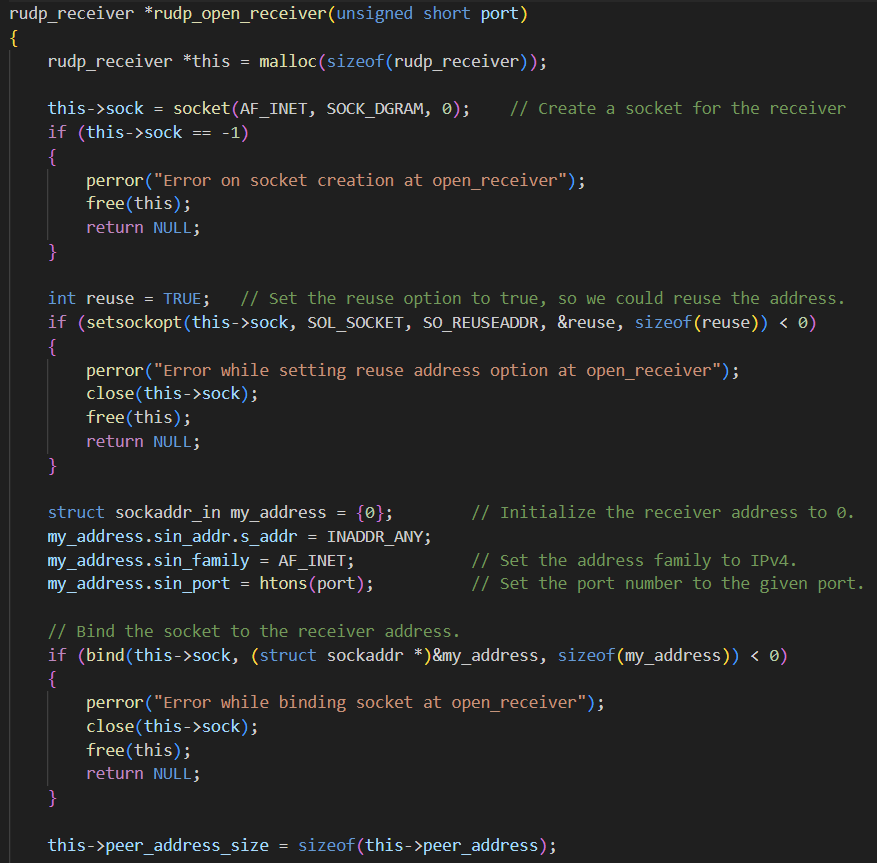
אם אחד התנאים לא מתקיים, נדפיס שגיאה ונחזור לתחילת הלולאה.

במידה וכל התנאים מתקיימים, הרי שזו הודעה תקפה, נשנה את משתנה ה-successful ל-True, מה שיסמן ללולאה לסיים את הריצה.



אם סיימנו את נסיונות שליחת הודעת ה-SYN ללא הצלחה, כלומר ניסינו MAX\_TRIES פעמים ובאף אחת מהפעמים לא קיבלנו הודעת SYN ACK (עם Checksum תקין כמובן), נדפיס שגיאה ונסגור את Socket השולח.

לבסוף נחזיר מצביע ל-Socket השולח. אם הייתה שגיאה במהלך הריצה של הפונקציה, החזרנו NULL על מנת לסמן זאת.



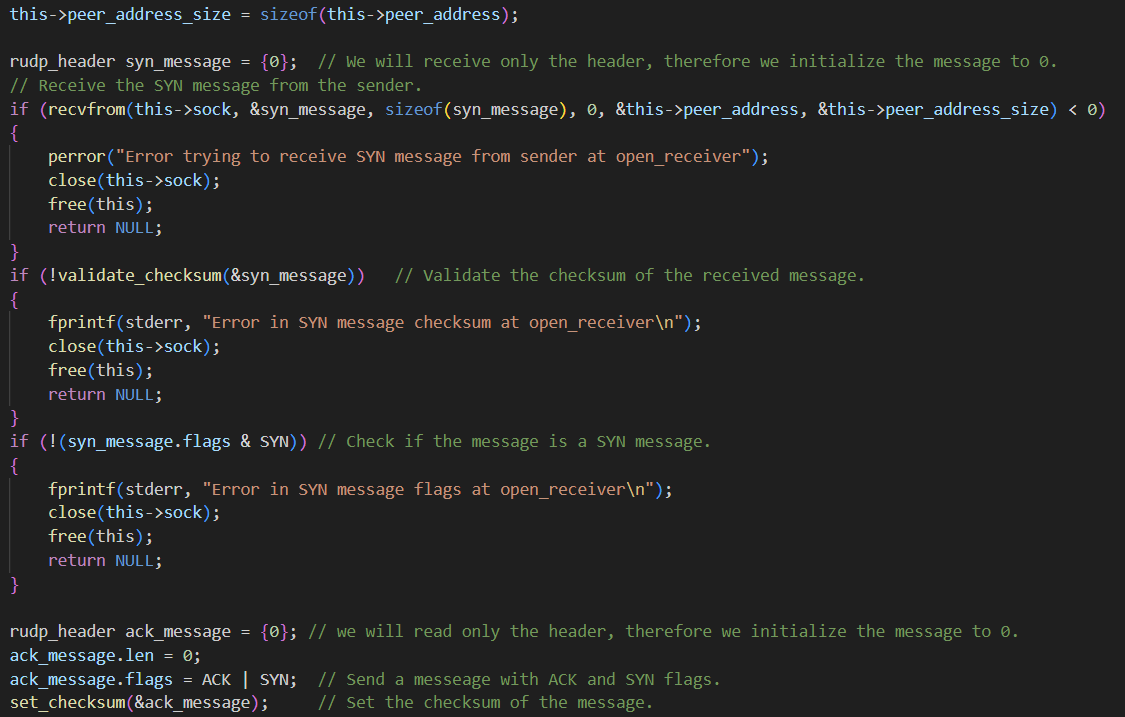
פונקציה לפתיחת RUDP Receiver socket.

הפונקציה מקבלת פורט ופותחת Socket שמוקצה לפורט הזה.

ראשית, נקצה מקום ל-Socket ונגדיר אותו בתור UDP Socket מבוסס על IPv4. לאחר מכן נגדיר שניתן להשתמש בכתובת של ה-Socket יותר מפעם אחת (בשביל למנוע שגיאות של “Address already in use”)

לאחר מכן, כמו שביצענו ב-TCP Receiver socket, נגדיר sockaddr\_in מסוג IPv4, שיכול לקבל תקשורת מכל כתובת, ומאותחל לפורט הנתון.

נבצע bind לאותו sockaddr\_in ל-socket שיצרנו, כך נגדיר ל-socket מאיזה פורט להאזין בהמשך ולאיזה סוג כתובת לצפות (IPv4).



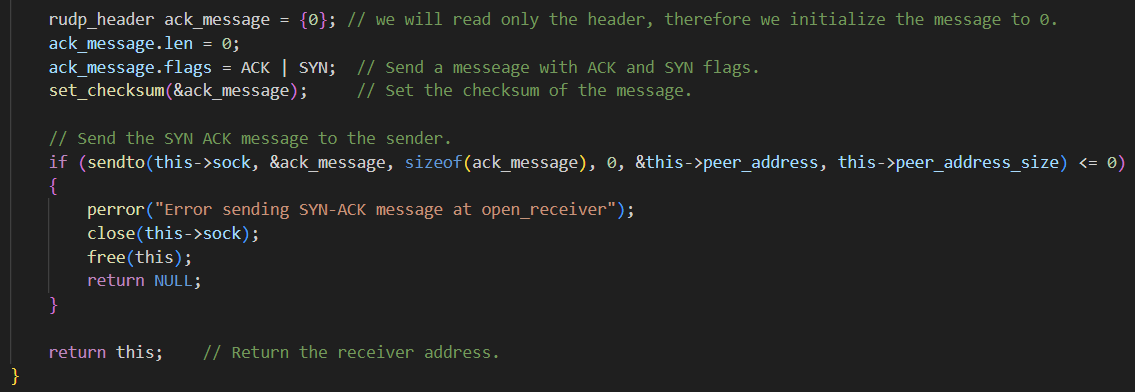
נכין הודעת SYN חדשה, הודעה זו ריקה, שכן בהודעות SYN/ACK/FIN אנו מקבלים רק את ה-Header.

נמתין לקבלת הודעה מהשולח, אותה נקצה לכתובת של ההודעה שאותה הכנו.

לאחר מכן, נבדוק האם ההודעה היא הודעת SYN תקינה:

אם ה-Checksum שלה לא תקין – נחזיר שגיאה.

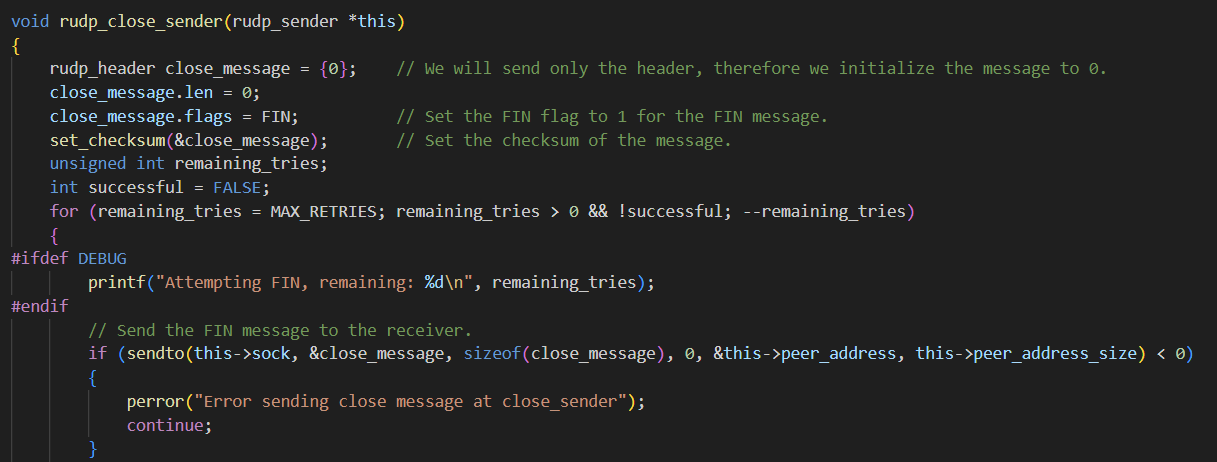
נבדוק את ערך flag ה-SYN שלה ב-header, אם אינו 1 – נחזיר שגיאה (נבצע את הבדיקה באמצעות שליפה של ערך SYN עם bitwise-and).



לאחר שהתקבלה הודעת SYN, אנחנו רוצים להחזיר הודעת SYN ACK.

נכין הודעת SYN ACK בה נדליק את ה-SYN & ACK flags (בעזרת bitwise-or ביניהם). ולהודעה זו נגדיר Checksum כדי שתהיה הודעה תקינה.

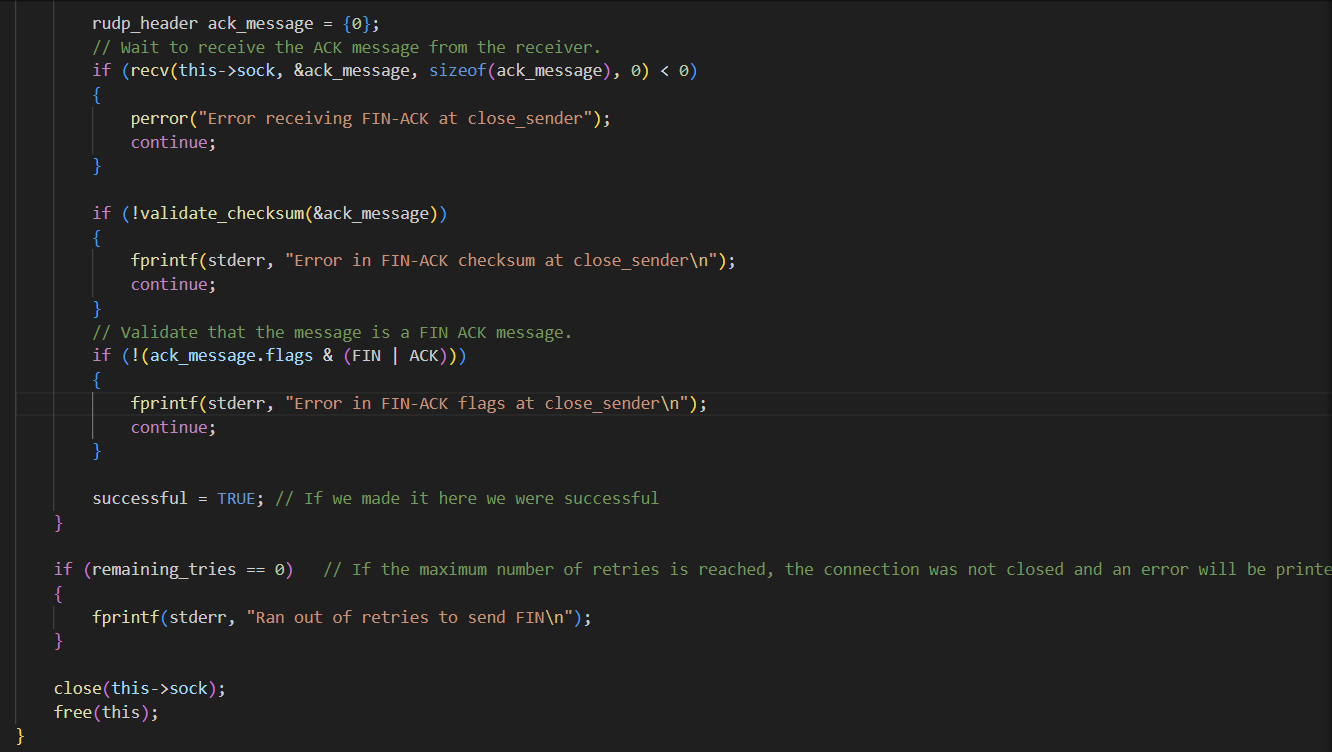
לאחר מכן נשלח את ההודעה ל-Sender ובמידה ולא הייתה שגיאה, נחזיר את מצביע ל-RUDP Receiver socket.



פונקציה זו מקבלת RUDP Sender, שולחת הודעת FIN ל-Receiver אליו הוא קשור וסוגרת אותו.

ראשית הפונקציה מכינה הודעת FIN, ע"י הגדרת RUDP Header בו ה-FIN flag דלוק. לאחר מכן נגדיר על הודעה זו Checksum כמובן.

נשלח את הודעת ה-FIN כמות הפעמים שהוגדרה (MAX\_RETRIES) או עד אשר נקבל FIN ACK ה-Receiver.



נקצה מקום להודעת ACK, אליה נכניס את ההודעה שנקבל מה-Receiver.

כאשר תתקבל הודעה, נבדוק את תקינות ה-Checksum שלה ואם היא אכן הודעת FIN ACK. במידה ושני התנאים מתקיימים, נגדיר ניסיון זה כהצלחה. במידה ולא, נדפיס שגיאה וננסה שוב לשלוח הודעת FIN.

אם יצאנו מהלולאה כאשר הגענו למקסימום ניסיונות שליחת ההודעה, נדפיס שגיאה.

לבסוף נסגור את ה-Socket ונשחרר את הקצאת המקום.

להמשיך