תיק פרויקט

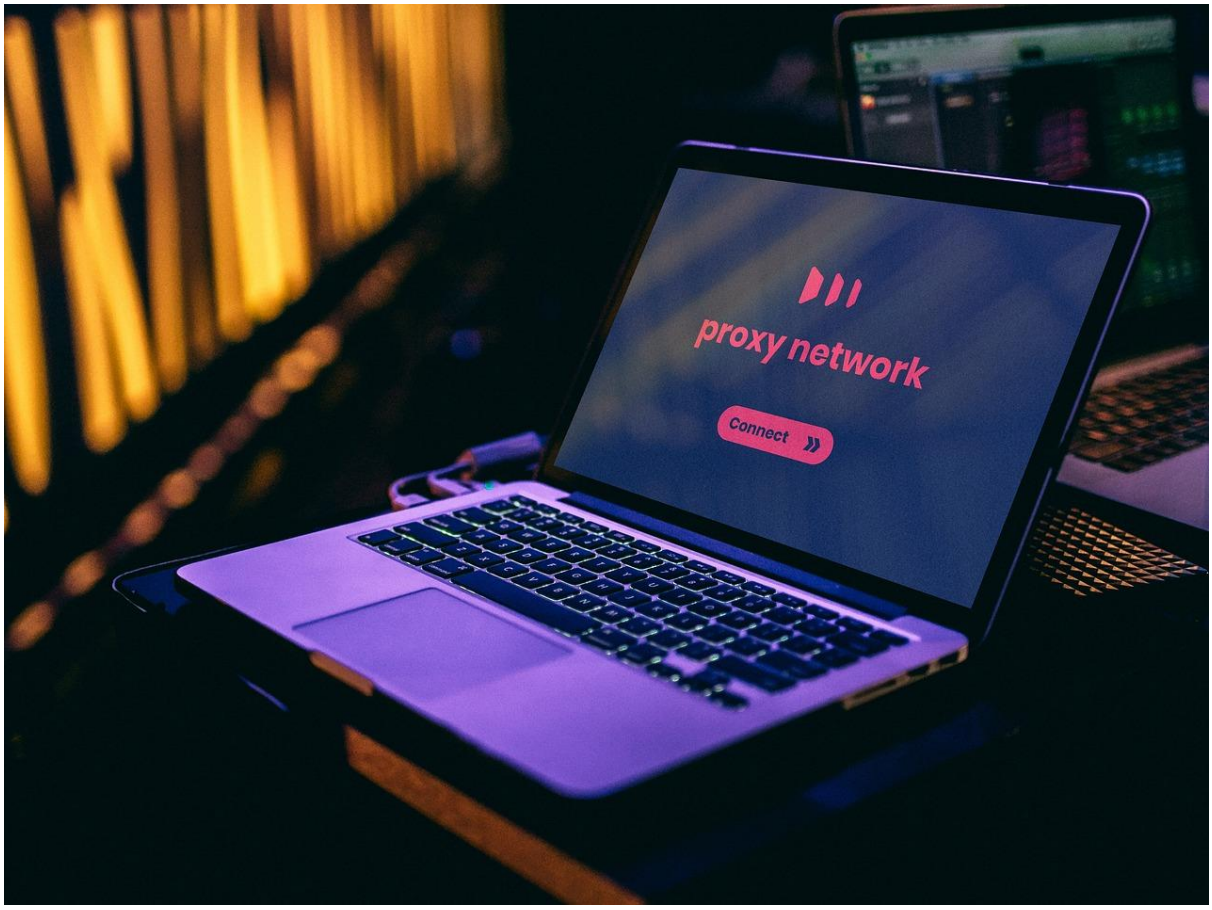


Image by Kevin Morison from Pixabay

מגיש: סער מזור

שם העבודה: MyProxy

תוכן עניינים

מבוא

תיאור ראשוני של המערכת:

המערכת תשמש בתור שער בין הלקוח ובין האינטרנט, והיא תפעל לפי פרוטוקול HTTP (וגם HTTPS). המערכת היא שרת החוצץ בין משתמשי קצה ובין האתרים שהם גולשים אליהם. שרת הפרוקסי מציע דרך תקשורת מאובטחת, המעניקה מידה מסוימת של אנונימיות למשתמשי הקצה.

בחרתי בפרויקט משום שהתפעמתי מהרעיון של מתן אפשרות לגלישה בטוחה יותר, ומהתחכום הכלול בכך.

אני צופה מספר אתגרים בפרויקט – קודם כל ניהול הלקוחות בצורה נוחה ויעילה, הפחתת זמן התקורה ככל הניתן (למשל על ידי שימוש בתכנות אסינכרוני), תמיכה במספר רב של לקוחות ועוד.

הלקוח:

הלקוח יהיה כל אדם אשר רוצה לגלוש באינטרנט בדרך קצת יותר מאובטחת ואנונימית יחסית.

יעדים:

מתן שירות מאובטח ואיכותי, תוך שמירה על יעילות מרבית ותמיכה במספר רב של לקוחות.

בעיות תועלות וחסכונות:

הבעיה נוצרת כאשר אדם רוצה לגלוש ברשת בצורה אנונימית יותר (ולכן גם מאובטחת יותר), כלומר הוא לא רוצה לדבר ישירות עם השרתים, אלא דרך מתווך. הרצון יכול להיגרם ממספר אינסופי של סיבות.

המערכת תיתן ללקוח אפשרות לשוחח עם שרתי HTTP דרכה, ובכך השרת אינו יכול לדעת מי משתמש הקצה שבאמת משוחח איתו.

סקירת פתרונות קיימים:

קיים מספר רב של שרתי פרוקסי, הפועלים לפי מגוון נרחב של פרוטוקולים.
קיימים שרתי פרוקסי הפועלים לפי פרוטוקול HTTP, כמו המערכת שאני אבנה, אך קיימים גם שרתים אחרים הפועלים לפי פרוטוקולים אחרים גם בשכבת האפליקציה (SMTP, FTP וכו') וגם בשכבת התעבורה (TCP, UDP וכו').

סקירת טכנולוגית הפרויקט:

טכנולוגיית הפרויקט תתבסס בעיקר על תקשורת TCP באמצעות Sockets, תוך שימוש במוסכמות פורטוקול HTTP, ותכנות אסינכרוני בכדי לייעל את המערכת גם מבחינת זמן וגם מבחינת משאבים.
שפת התכנות התשמש לבניית הפרויקט היא C#.

תחום הפרויקט:

הפרויקט עוסק בתקשורת TCP באמצעות Sockets, אלגוריתמים לניהול הלקוחות ולתפעול המערכת, תכנות אסינכרוני בשיטת async await, תמיכה בפרוטוקול HTTP וגם HTTPS, תכנות מונחה עצמים, טיפול בExceptions, שימוש בNullable types בLambda Expressions, וב Ternary Conditional Operator.

כמו כן, המערכת הינה Thread Safe, והיא מיישמת זאת באמצעות Locks או באמצעות מבני נתונים שהם Thread Safe.

תיאור מפורט יותר של המערכת:

המערכת אמורה לחכות שאחד ממשתמשי הקצה ינסה להתחבר אליה, או לשלוח לה הודעה. לאחר מכן היא תבין מה המשתמש רוצה ממנה (או שההודעה תהיה מוצפנת ואז היא לא תנסה להבין) ותטפל בבקשה.
אם למשל המשתמש ישלח CONNECT, המערכת תתחבר ליעד הרצוי ותודיע למשתמש שהחיבור בוצע בהצלחה.
במקרה שהמשתמש ישלח הודעת HTTP אחרת היא תעביר זאת ליעד הרצוי (אם החיבור עם היעד נותק או שהוא לא קיים, המערכת תתחבר אל היעד).

אם המערכת לא מבינה את ההודעה (כלומר לא קיימים בה פרטים החייבים להיות בהודעת HTTP), המערכת תעביר את המידע הזה ליעד המקושר עם המשתמש (כל משתמש הינו אובייקט המכיל בין היתר את מי שהוא מדבר איתו כעת, או שהשדה הזה יהיה NULL במקרה ואין עדיין משתמש יעד).
אם החיבור אל היעד אינו תקין, המערכת תנתק את שני החיבורים בצורה בטוחה (גם Thread Safe וגם תשתדל לעשות Graceful Shutdown).

משום שלא ניתן לדעת מראש כמה הודעות כל צד ישלח, ובפרט כשההודעות מוצפנות, המערכת תתייחס גם ללקוח וגם לשרת באותה צורה (כלומר באמצעות אותו אובייקט, והאלגוריתמים לניהול השיחה עם הלקוח ועם השרת יהיו זהים).

בנוסף, המערכת עושה שימוש רב בפונקציות אסינכרוניות מה שמייעל את הרצתה.

המערכת תשמור בקובץ את כל ההודעות שהיא מקבלת, מתי הן התקבלו, ממי היא מקבלת אותן ולאן הן מיועדות.

הבדיקות שמתוכננות לפרויקט:

במסגרת בדיקת המערכת, ייבדקו מספר נושאים העלולים לגרום להשבתת המערכת, או להפחתת יעילות המערכת בצורה דרסטית.

המערכת תיבדק תחת עומס, תוך כדי שהיא נותנת שירות למספר רב של משתמשים שכל אחד מהם שולח המון הודעות. בדיקה זו תהיה מקיפה ותדרוש מכל ההיבטים הבעייתיים בפרויקט לעבוד (התחברות, תקשורת באמצעות HTTP וגם HTTPS, תמיכה במספר רב של משתמשים ויעילות)

הבדיקה תיערך באמצעות התחברות דרך מספר לקוחות שכולם יהיו דפדפנים במקרה הזה, וכולם יצפו בסרטונים ביוטיוב באיכות 8K. בזמן שהמערכת מספקת את השירות הנ"ל, נבדוק דרך תוכנת Process Explorer את אחוז הניצול מהמעבד שהמערכת דורשת ואת כמות הThreads. אם נראה שהמספרים סבירים (ו/או שמרבית הThreads ישנים) נדע שהמערכת עומדת בהצלחה.

הבדיקה הזו תיבדק בהדרגה, כאשר כל feature יוסף.
קודם כל תיבדק התחברות לקוח למערכת וניהול שיחה עם שרת HTTP, לאחר מכן ניהול קישור בין לקוח לשרת HTTPS, ולבסוף מתן שירות למספר רב של לקוחות ביעילות.

תיאור תחום הידע

אוסף יכולות נדרשות:

- הרצת התוכנית בסביבה המאפשרת זאת – על מנת שהתוכנית תעבוד
- שימוש במחלקת ה-Proxy – על מנת להריץ את הפרויקט עצמו
- גישה למחלקת ה-Socket, ו/או ל-Sockets הרלוונטיים – בכדי ליצור או לעשות פעולה כלשהי של תקשורת בין שני מחשבים בשכבת הרשת
- יצירת IPEndPoint וSocket בפורמט הנכון (IPv4, Stream, TCP) ולעשות Bind, Listen, Accept – בכדי לממש את הקישור של הכתובת והפורט, ולאפשר ללקוחות להתחבר
- שימוש בפעולות (במקרה הזה האסינכרוניות) המאפשרות האזנה ושליחה של הודעות – בכדי לתקשר בין שני מחשבים
- המתנה תמידית (בצורה שלא תתקע את התכנית – ע"י פעולות אסינכרוניות במקרה הזה) להודעות שיגיעו או ללקוחות שיתחברו – במטרה לתת שירות למי שמבקש זאת
- ניהול לקוחות בצורה הולמת (העברה וקבלה של מידע רלוונטי, שמירה במבנה נתונים) – בכדי לתת מענה ראוי ונכון, בצורה מסודרת ללקוחות
- סגירת חיבורים ל-Socket שאינם רלוונטיים יותר, ולנסות לעשות זאת מבלי "לטרוק את הדלת בפרצוף" – על מנת לחסוך במשאבים, ולעשות זאת בצורה שלא תגרום ל"הרעבה" של הלקוח
- עדכון ושימוש במסד נתונים העוקב אחרי כל החיבורים (במקרה הזה מילון שהוא Thread Safe) – במטרה לגשת למידע בצורה נוחה, יעילה ואינה הרסנית
- הצגת ההתרחשויות במסך – בעיקר למטרת דיבאג
- עדכון ה-Log הרלוונטי – בעיקר למטרת דיבאג
- ניהול שגיאות – בכדי למנוע את קריסת התוכנית כשאין זה הכרחי
- שימוש ב-nullable types וב-null במידת הצורך – על מנת לעבוד בצורה נוחה, יעילה ושלא מעוררת שגיאות סתמיות

- בקיאות בפרוטוקול HTTP – במטרה להבין מה הלקוח רוצה וכיצד לפעול בהתאם

- זיהוי הודעות מוצפנות והעברתן ללא פירוש או שינוי – בכדי לאפשר ללקוחות לתקשר אחד עם השני דרך הפרוקסי בצורה מאובטחת

- המרת המידע מבתים למחרוזות, ולהפך – על מנת לשלוח את ההודעות ברשת, או לפענחן (פרוטוקול HTTP הוא מחרוזתי ולכן כל המידע חייב להיות מורכב ממחרוזות, כל עוד הוא לא מוצפן)

- מעקב אחרי (הלקוח) הנמען הרלוונטי לכל לקוח – בכדי להעביר בנוחות ובמהירות את ההודעות מלקוח אחד ללקוח אחר

- יצירת מחלקה המכילה מידע חשוב לגבי כל לקוח (ה-Socket שלו, מי הוא, מי המשתמש השני שהוא מתקשר איתו) – שימוש בתכנות מונחה עצמים בשביל נוחות, יעילות ומתוך מחשבה שבעתיד אני אולי ארצה לעשות שינויים קלים בתכונות או המתודות של הלקוח

- העמסת אופרטורים כך שניתן יהיה לבדוק שוויון בין לקוחות – בכדי לעשות פעולות על מבנה הנתונים העוקב אחרי הלקוחות, במידת הצורך (למשל במטרה למנוע כפל משתנים/משאבים ללקוחות)

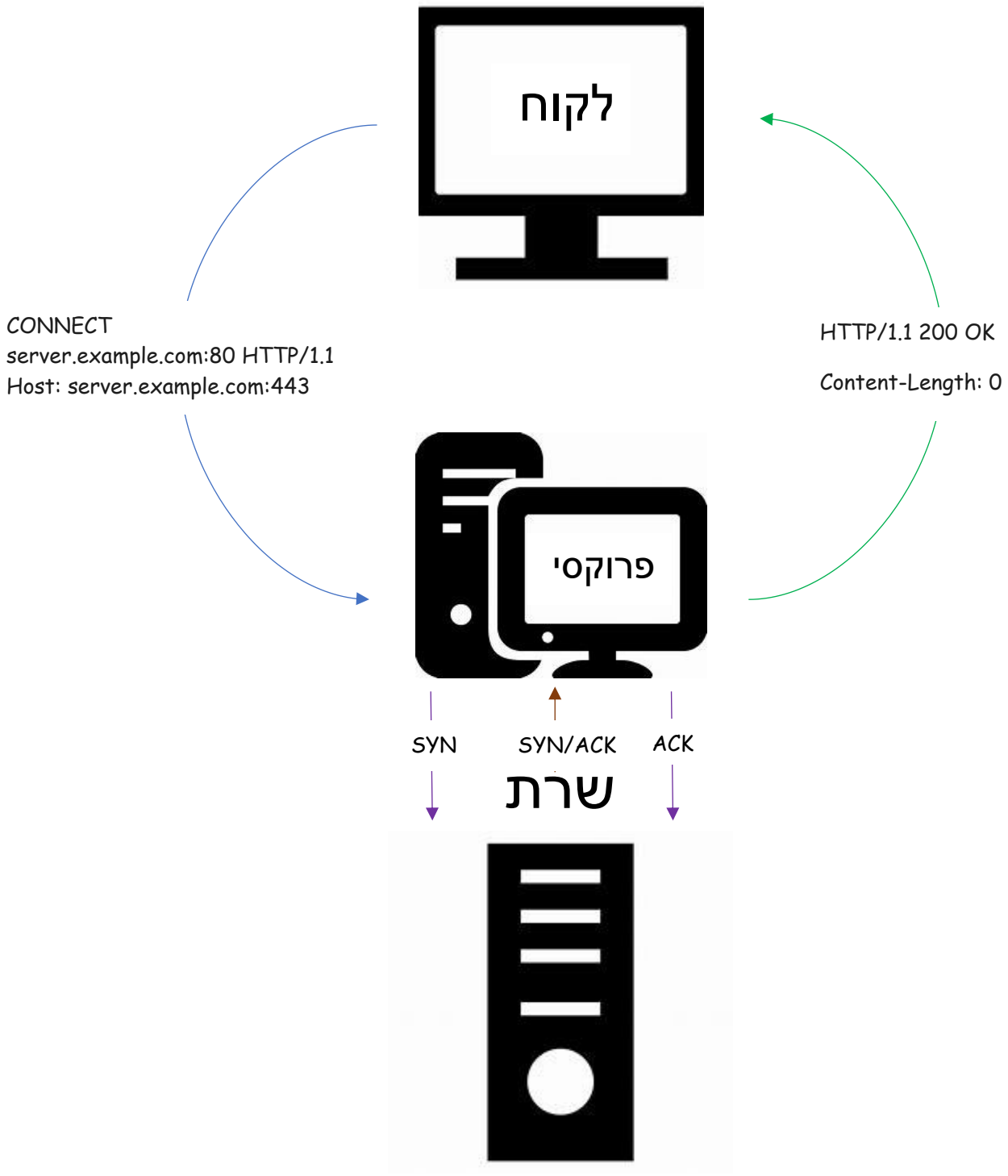
- הקמת מספר תהליכונים – בכדי לאפשר ביצוע פעולות "במקביל". ייעשה באופן אוטומטי על ידי הCLR

- הגבלת הגישה לפורטים ספציפיים (443, 80) – בכדי למנוע מתקפות שעלולות לפגוע בשרתי פרוקסי, כפי שנעשו בעבר.

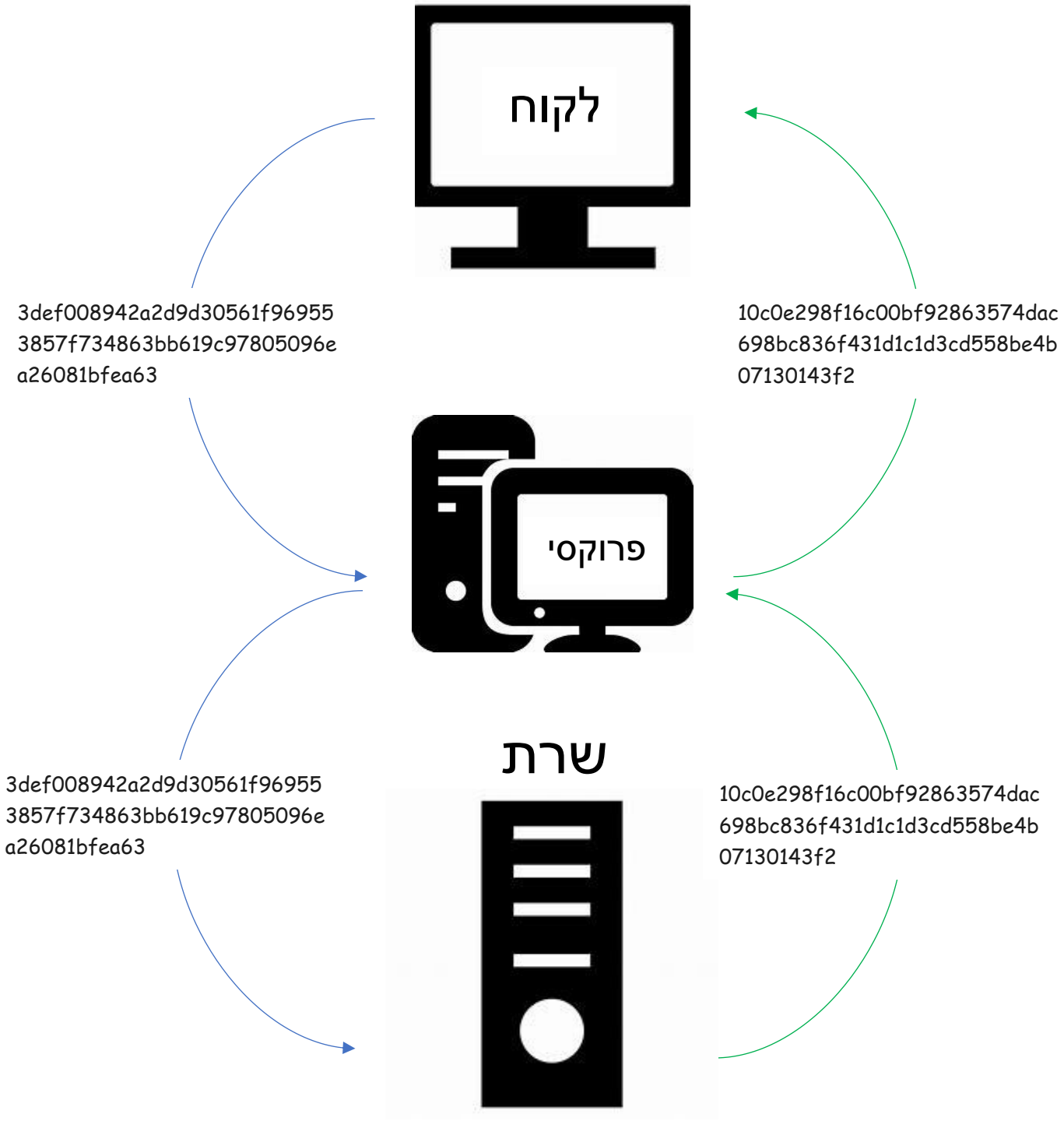
מבנה הפרויקט

תקשורת בין לקוח לשרת HTTPS, דרך הפרוקסי:

שלב א' - התחברות



שלב ב' – תקשורת בין משתמשי הקצה



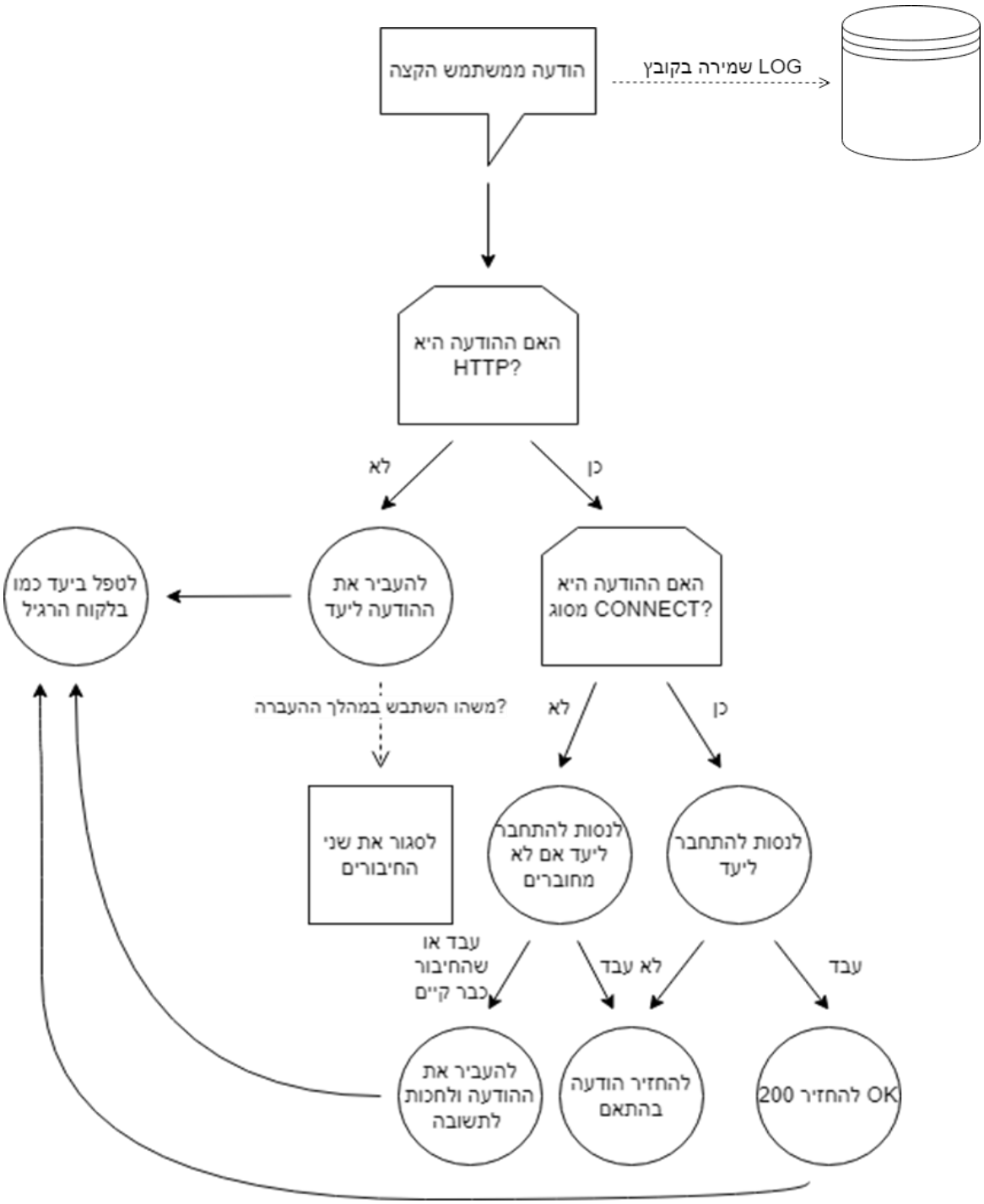
הטכנולוגיה הרלוונטית לפרויקט:

שפת התכנות: C#, תוך שימוש בNET. גרסה 7 ומעלה

מערכת הפעלה: המערכת אמורה לפעול כראוי בכל מערכת הפעלה התומכת בNET. גרסה 7 ומעלה

כחלק מהדרישות האלו, המחשב המריץ את המערכת חייב לתמוך ב-multithreading, ב-multiprocessing, ובתקשורת באמצעות Sockets.

זרימת המידע:



האלגוריתמים המרכזיים בפרויקט:

מתן מענה למספר משתמשים במקביל

השרת צריך לתת מענה למספר משתתפים בו זמנית על מנת שיהיה באמת רלוונטי למספר רב של אנשים, והוא צריך לדעת לנהל זאת בצורה נכונה (שלא יקרו טעויות כמו שליחת ההודעה לאדם הלא נכון) ויעילה (כמה שיותר מהירה ושחוסכת במשאבים).

ישנן מספר דרכים לאפשר זאת, ולהלן העיקריות:

יצירת Thread לכל לקוח וניהול כל הלקוחות באופן זהה ו"במקביל".

יצירת Process לכל לקוח וניהול כל הלקוחות באופן זהה ובמקביל.

שתי הדרכים הנ"ל מצריכות משאבים רבים, ולכן החלטתי שלא להשתמש בהן אלא בדרך השלישית שהיא ניהול הלקוחות (וכמעט כל הפונקציות בשרת) בצורה אסינכרונית.

דרך זו נחשבת ליעילה ביותר משום שאין צורך ליצור threads או processes רבים, אלא נעשה שימוש באותם threads לניהול הלקוחות השונים ובקשותיהם.
לא נוצרים threads רבים אך ה-CLR כן מוסיף עוד כאשר הוא מוצא לנכון, ובהתאם למינימום והמקסימום המוגדרים מראש (במערכת שלי הם על ברירת המחדל).

בכדי לאפשר הרצה אסינכרונית "במקביל", השרת מפעיל Task הנקרא HandlePeerAsync לכל לקוח. המטלה הזו נמצאת בלולאה אינסופית (כמובן רק כל עוד החיבור קיים), והיא מחכה לקלט ממשתמש הקצה או לניתוק החיבור.

המקור שהכי נעזרתי בו במטרה ללמוד מאפס איך לתכנת בצורה אסינכרונית ב-#C הוא הדוקומנטציה של מייקרוסופט.

התייחסות ללקוח ולשרת כאובייקט זהה

על מנת שהקוד יהיה מובן וקריא, ובכדי לא לשכפל את הקוד שלי, רציתי למצוא דרך להשתמש באותו הקוד לניהול התקשורת עם שני הצדדים.

בעיה נוספת היא שכאשר שני הצדדים מתקשרים בפרוטוקול מוצפן (HTTPS), המתווך לא יכול לדעת כמה הודעות ישלחו מכל כיוון ובכל שלב, ולכן יש לחכות להודעות באופן תמידי משני הכיוונים ולהעבירן הלאה.

בכדי לפתור את הבעיות הללו יצרתי מחלקה בשם Peer המייצגת משתמש קצה. לכל משתמש קצה יש Socket, כתובת, ומשתמש יעד. בנוסף על מנת שאני אוכל להשוות בין המשתמשים נעזרתי בהעמסת אופרטורים.
בעקבות זאת, כל הקוד שלי לא צריך להבדיל בין השרת והלקוח ואין צורך בשכפול קוד.

טיפול בהודעות HTTP ובהודעות HTTPS

המערכת צריכה לדעת לתקשר נכון עם לקוחות בפרוטוקול HTTP, אבל גם לצפות להודעות מוצפנות שהוא אינו יכול לפרש ולהעביר אותן ליעד הרלוונטי.

על מנת לפתור בעיה זו, המערכת בודקת אם ההודעה מכילה אחת מפקודות פרוטוקול HTTP ופועלת לפי זאת. אם לא, המערכת תעביר את ההודעה למשתמש היעד.

המערכת פועלת לפי מוסכמות הפרוטוקול, ויודעת לפעול כראוי לכל פקודה הרלוונטית אליה.

השתמשתי במספר מקורות על מנת לפתור בעיה זו, ביניהם: ספר רשתות במרכז לחינוך לסייבר, StackOverFlow, MDN Web Docs - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

ניהול משאבים משותפים

המערכת צריכה לנהל את המשאבים המשותפים בצורה יעילה ולא מכשילה.

ישנן מספר דרכים לנהל מצבים משותפים, אך הרבה מהן לא רלוונטיות מכיוון שהן לא Thread safe או שהן לא פועלות במהירות גבוהה מספיק.

בכדי לפתור את הבעיה הזו, המערכת מנקה את המשאבים שאין בהם שימוש יותר (כמו משתמשים במילון המשותף) והיא עושה זאת באמצעות שימוש בפונקציות שהן Thread safe.

המקור העיקרי שהשתמשתי בו בכדי למצוא פיתרון לבעיה זו הוא הדוקומנטציה של מייקרוסופט בקישורים הבאים:

https://learn.microsoft.com/en-us/dotnet/standard/collections/thread-safe/how-to-add-and-remove-items

https://learn.microsoft.com/en-us/dotnet/api/system.collections.concurrent.concurrentdictionary-2?view=net-7.0

תיאור סביבת הפיתוח:

הפיתוח נעשה בסביבת .NET 7. תוך שימוש בVisual Studio 2022.

הבדיקות נעשות באמצעות דפדפן חיצוני, אפליקציית Postman, או כל אפליקציה התומכת בחיבור לאינטרנט דרך פרוקסי מוגדר.

תיאור פרוטוקול התקשורת:

הפרוטוקול הוא פרוטוקול מחרוזתי, המורכב מ-Headers וממטען המכיל מידע נוסף.

פירוט על כלל הפקודות ניתן למצוא בקישור הבא - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

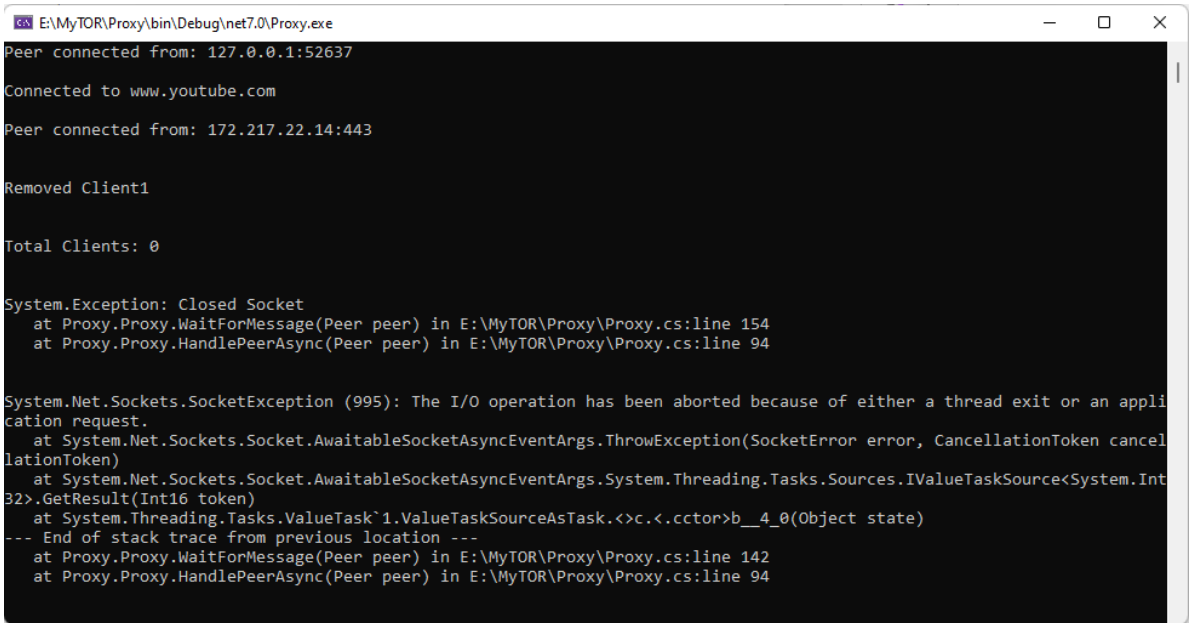
הפקודה המשמעותית ביותר שיש לשים עליה דגש רב בתור שרת פרוקסי היא הפקודה CONNECT. הפקודה מגיעה מהלקוח ובעקבותיה יש לנסות להתחבר לשרת היעד, להודיע ללקוח האם החיבור בוצע בהצלחה או שחלה טעות (ולהוסיף מה בדיוק הטעות, למשל – היעד אינו נמצא), ולהתכונן להזרמת מידע בין שני הצדדים.

תיאור מסכי המערכת:

המערכת מכילה מסך אחד (המשמש למטרות פיקוח על ריצת המערכת) המראה את:

- התחברות הלקוחות
- התחברות המערכת לשרתים
- סגירת החיבור עם הלקוחות או השרתים
- סך כל המשתמשים המחוברים
- הודעות שגיאה רלוונטיות

משום שהמערכת פועלת ללא צורך בקלט של מקלדת/עכבר/שלט, המסך הרלוונטי היחיד משמש לפיקוח על התפקוד השוטף של המערכת.



תיאור מבני הנתונים:

המערכת מכילה log המשמש למעקב אחר ההודעות העוברות דרך הפרוקסי, מתי הן התקבלו, ממי היא מקבלת אותן ולאן הן מיועדות.

סקירת חולשות ואיומים:

המערכת חשופה לחולשה הקיימת בכל פרוקסי HTTP, שהיא עמידה בתקיפות מניעת שירות בשלל גווניה (SYN Flood, Slow Loris וכדומה). החולשה אף מועצמת משום שהמערכת פועלת על מחשב סטנדרטי (בעל משאבים מוגבלים ביחס לשרתי פרוקסי ממוצעים), ובתוך רשת NAT.

בניגוד לצפוי מהרבה אנשים, המערכת עמידה בפני MITM משום שהמערכת עצמה היא כבר סוג של איש באמצע, והמשתמשים אמורים לדעת לא לסמוך עליה לחלוטין (כלומר להשתמש בHTTPS).

מימוש הפרויקט

סקירת כל המחלקות המיובאות:

שם המחלקה: System

המחלקה נחוצה על מנת לכתוב קוד בשפת #C, שיוכל לרוץ (בין אם היא בולטת ובין אם היא מאחורי הקלעים כמו ב-Top Level Statements).

שם המחלקה: System.Text

המחלקה משמשת להמרת מידע באמצעות קידודים שונים (למשל המרה מטקסט לבתים באמצעות UTF8).

שם המחלקה: System.Net

המחלקה משמשת לתקשורת בין מחשבים שונים.

שם המחלקה: System.Net.Sockets

המחלקה מיועדת לאפשר ניהול תקשורת בין מחשבים שונים ברמת הSocket (ולא בצורה מופשטת יותר, המאפשרת פחות שליטה, כמו באמצעות מחלקת TCPListener).

שם המחלקה: System.Collections.Concurrent

המחלקה מיועדת לאפשר שימוש במבני נתונים שהם Thread Safe, כמו Concurrent Dictionary.

סקירת כל המחלקות שפיתחתי:

שם המחלקה: Proxy

תפקיד המחלקה: ניהול שוטף של הפרוקסי בשלל התחומים (תקשורת, ניהול משאבים וכו').

תכונות המחלקה:

- _bufferLength – קובעת מה גודל הבאפר שיש להשתמש בו על מנת לקבל הודעות באמצעות ה-Sockets.
- _proxySock – ה-Socket של שרת הפרוקסי (באמצעותו מקבלים לקוחות חדשים וכו').
- _netLogger – ה-logger שדרכו המערכת כותבת ל-log התקשורת שלה.
- _peerDic – מילון Thread Safe שבו נשמרים משתמשי הקצה (לקוחות ושרתים כאחד) והנמענים שלהם (כל זוג מופיע רק פעם אחת).
- totalCliHistory – משתנה העוקב אחר סך כל הלקוחות שהתחברו מאז תחילת הרצת הפרוקסי.

פעולות במחלקה:

1. RunProxyAsync
   - הפונקציה מקבל את מספר הפורט (משתנה מסוג int)
   - הפונקציה מכינה את השרת לקבלת לקוחות, ונמצאת בלולאה תמידית כאשר היא תמיד קוראת לפונקציה המקבלת את הלקוחות (AcceptHandleAsync)

2. AcceptHandleAsync
   - הפונקציה לא מקבלת שום פרמטרים
   - הפונקציה מחכה (בצורה אסינכרונית) שלקוח ינסה להתחבר לשרת, היא מעדכנת את סך כל הלקוחות שהתקבלו, היא מכניסה אותם למילון הלקוחות, ולבסוף היא מעבירה שליטה לפונקציה המטפלת במשתמש הקצה (HandlePeerAsync).

3. HandlePeerAsync
   - הפונקציה מקבלת משתנה מסוג Peer (משתמש קצה – אני ארחיב על כך בהמשך)
   - הפונקציה נמצאת בלולאה אינסופית כאשר היא קוראת לפונקציה אחרת המחכה להודעה ממשתמש הקצה (WaitForMessageAsync), והיא מנהלת שגיאות שעלולות לצוץ כתוצאה משרשרת הפונקציות שיקראו ומשחררת משאבים בהתאם.

4. RemovePeerFromDic
   - הפונקציה מקבלת משתנה dic מסוג ConcurrentDictionary<Peer, Peer?> שמהווה את המילון שממנו מוחקים את הערכים.
   - הפונקציה מוחקת את הלקוח מהמילון, גם אם הוא ה"מפתח" וגם אם הוא ה"ערך" במילון (תוך ניהול שגיאות כמובן).

5. WaitForMessageAsync
   - הפונקציה מקבלת משתנה peer מסוג Peer
   - הפונקציה מחכה ומקבלת הודעות מהPeer, אם הוא נמצא במילון (אחרת היא זורקת Exception). אם היא קיבלה הודעה חסרת בתים, היא יודעת לקרוא לפונקציה הסוגרת את החיבור באופן זהיר (GracefulShutdown) ולזרוק Exception.

אחרת, היא מעבירה את המידע לפונקציה HandleRequestAsync שתפקידה לטפל בהודעה.

6. HandleRequestAsync
   - הפונקציה מקבלת משתנה peer מסוג Peer, משתנה receivedBytes מסוג byte[] (המכיל את המידע של ההודעה בבתים) ומשתנה message מסוג string (המכיל את ההודעה במחרוזת)
   - הפונקציה מעדכנת את ה-log, ואז קוראת לפונקציה הבודקת אם ההודעה היא HTTP (CheckHTTP), וקוראת לפונקציה הרלוונטית בהתאם לתוצאה (HandleHTTPMessage או HandleEncryptedMessage).

7. HandleHTTPMessage
   - הפונקציה מקבלת משתנה peer מסוג Peer, משתנה receivedBytes מסוג byte[], משתנה message מסוג string
   - הפונקציה מעבירה שליטה לפונקציות הרלוונטיות בהתאם לשאלה האם ההודעה מכילה את הפקודה "CONNECT" (AttemptConnection, ForwardHTTPMessage או HandleConnect), היא מעדכנת את המילון peerDic_ בהתאם ואם היא נתקלת בשגיאה היא סוגרת את ה-Socket.

8. HandleEncryptedMessage
   - הפונקציה מקבלת משתנה cliPeer מסוג Peer, ו-receivedBytes מסוג byte[]
   - הפונקציה מעבירה את ההודעה (בבתים) ליעד הרלוונטי, תוך טיפול בגורמי שגיאה שעלולים לצוץ (למשל Socket שלא קיימת יותר).

9. HandleConnect
   - הפונקציה מקבלת משתנה peer מסוג Peer, host מסוג string? (כתובת הIP של היעד), port (יעד) מסוג int, hostPort (בעצם חיבור של ה-IP וה-port על מנת להשתמש בפונקציות של מחלקת Socket לשליחת ההודעה) מסוג string
   - הפונקציה משיגה את היעד הנוכחי של הpeer לפי property שלו, בודקת שהוא לא מחובר ליעד החדש ומנסה לחבר אותו ליעד החדש באמצעות פונקציית AttempConnection.

10. AttemptConnection
    - הפונקציה מקבלת משתנה peer מסוג Peer, host מסוג string, port מסוג int, hostPort מסוג string, message מסוג string?
    - הפונקציה מבצעת מספר בדיקות התאמה (בגלל זה יש משתנים נפרדים לhost, port וה-hostPort, בשביל נוחות הבדיקה) ובין היתר בודקת שהגישה ניתנת רק לפורטים 80, 443. לאחר מכן הפונקציה מתחברת עם סוקט ליעד, מעדכנת את המילון, שולחת את ההודעה במידה ויש, ומחזירה הודעת 200 OK, שתגיע ללקוח לאחר טיפוס בחזרה במעלה הפונקציות. במידה והייתה שגיאה הפונקציה תחזיר הודעת 502 Bad Gateway.

<u>שם המחלקה:</u> ProxyUtils

<u>תפקיד המחלקה:</u> אחסון פונקציות נוספות לנוחיות הפרוקסי.

<u>תכונות המחלקה:</u> אין

<u>פעולות במחלקה:</u>

1. GracefulShutdown
   - הפונקציה מקבלת משתנה peer מסוג Peer
   - הפונקציה משתדלת לסגור את החיבורים עם הpeer והנמען שלו בדרך נחמדה, באמצעות שימוש ב-Socket.Shutdown וב-Socket.Close.

2. ForwardHTTPMessage
   - הפונקציה מקבלת משתנה destPeer מסוג Peer, receivedBytes מסוג byte[], hostPort מסוג string?
   - הפונקציה משיגה את Socket הנמען של ה-destPeer ושולחת לו את ההודעה.

3. GetAddressee
   - הפונקציה מקבלת משתנה dic מסוג ConcurrentDictionary<Peer, Peer?>, משתנה peer מסוג Peer
   - הפונקציה מחזירה את המשתמש הנמען של הpeer באמצעות חיפוש במילון dic.

4. Get200OK
   - הפונקציה לא מקבלת משתנים.
   - הפונקציה מחזירה מחרוזת המכילה הודעה הולמת של 200 OK לפי פרוטוקול HTTP.

5. Get502BadeGateway
   - הפונקציה לא מקבלת משתנים.
   - הפונקציה מחזירה מחרוזת המכילה הודעה הולמת של 502 Bad Gateway לפי פרוטוקול HTTP.

6. Get403Forbidden
   - הפונקציה לא מקבלת משתנים.
   - הפונקציה מחזירה מחרוזת המכילה הודעה הולמת של 403 Forbidden לפי פרוטוקול HTTP.

7. GetHostNPort
   - הפונקציה מקבלת משתנה message מסוג string
   - הפונקציה משתמשת בRegex על מנת לשלוף את כתובת ה-IP, הפורט, כתובת ה-IP והפורט ביחד, ולהחזירם.

8. CheckHTTP
   - הפונקציה מקבלת משתנה message מסוג string
   - הפונקציה מחזירה האם ההודעה היא הודעת HTTP, על ידי בדיקה האם מופיעה אחת מהפקודות המוכרות של הפרוטוקול.

<u>שם המחלקה:</u> Peer

<u>תכונות המחלקה:</u> Sock מסוג Socket, HostPort מסוג string?, Addressee מסוג Peer?

<u>תפקיד המחלקה:</u> המחלקה מהווה משתמש קצה (לקוח או שרת), ומחלקת הProxy נעזרת בה על מנת לפשט את עבודתה.

<u>פעולות המחלקה:</u>

כל התכונות מכילות פונקציות Get, Set, ובנוסף ישנן הפעולות הבאות:

1. Peer
   - הפונקציה מקבלת משתנה sock מסוג Socket, hostPort מסוג string, addressee מסוג Peer?
   - הפונקציה היא הפעולה הבונה של המחלקה, והיא קובעת את תכונות המחלקה בהתאם לקלט שלה.

2. operator ==
   - הפונקציה מקבלת משתנה b1 מסוג Peer, b2 מסוג Peer
   - הפונקציה משווה בין המשתנים ומחזירה את התוצאה.

3. operator !=
   - הפונקציה מקבלת משתנה b1 מסוג Peer, b2 מסוג Peer
   - הפונקציה בודקת אם המשתנים אינם שווים, ומחזירה את התוצאה.

4. Equals
   - הפונקציה מקבלת משתנה obj מסוג object?
   - הפונקציה עושה cast למשתנה ובודקת האם הוא שווה לאובייקט הנוכחי.

5. GetHashCode
   - הפונקציה לא מקבלת משתנים
   - הפונקצית סוכמת את ערכי התכונות של האובייקט למספר מסוג int, ללא התכונה Addressee משום שהיא תגרום ללולאה אינסופית, ומחזירה את המספר.

<u>שם המחלקה:</u> Logger

<u>תפקיד המחלקה:</u> המחלקה משמשת לכתיבת log לקובץ.

<u>תכונות המחלקה:</u> CurrentDirectory, FileName, FilePath – כולם מסוג string

<u>פעולות המחלקה:</u>

1. Logger
   - הפונקציה לא מקבלת משתנים
   - הפונקציה משמשת כפעולה הבונה של המחלקה, והיא קובעת את הערכים בכל אחד מהשדות.

2. WriteLog
   - הפונקציה מקבלת משתנה message מסוג string
   - הפונקציה כותבת לקובץ את הזמן המדויק ואת ההודעה עצמה.

הערה לגבי חלק ב':

הבעיות האלגוריתמיות שהסברתי הינן הבעיות המרכזיות ביותר והן רלוונטיות כמעט לכל קטעי.
יש התחשבות בהן (ופיתרון אליהן) או בחלקן כמעט בכל אחת מהפונקציות שפירטתי, ולכן זה בעייתי להראות מקום ספציפי שבו "הבעיה נפתרה".

הערה לגבי מסמך בדיקות מלא:

הוספתי את אופי כל הבדיקות לשלב האפיון לאחר שביצעתי אותן בגלל שחשבתי שצריך לכתוב אותן שם, ולכן אין לי את הפירוט המדויק שלהן, אך בגדול נתקלתי בבעיות במרבית השלבים של הבדיקה מסיבות שונות, וביניהן:

שימוש בפונקציה סינכרונית במקום אסינכרונית, אי אפשרות להתחבר לשרת, אי אפשרות לשרת מספר לקוחות במקביל, מספר לא קטן של טעויות בהבנת/שימוש בפרוטוקול HTTP (כמו החזרת תשובה לא נכונה ללקוח לאחר התחברות חיובית), לולאות אינסופיות, חוסר התאמה בין משתנים, חוסר טיפול נכון ב-null ועוד.

בסופו של דבר התקדמתי שלב שלב ופתרתי כל בעיה שנתקלתי בה תוך זהירות ומחשבה שנייה ושלישית, עד שהצלחתי לגרום לכל feature שרציתי להוסיף לעבוד (:

מדריך למשתמש

כלל קבצי המערכת:

- Logger.cs
- Peer.cs
- Program.cs
- Proxy.cs
- ProxyUtils.cs

התקנת המערכת:

בכדי שהמערכת תוכל לפעול חובה להריצה באמצעות מחשב המכיל NET. גרסה 7 ומעלה, ולאפשר ב-Firewall שההודעות יגיעו לשרת במידה וזה לא קורה אוטומטית.

למעט הדרישות הללו, אין דרישות נוספות למערכת. המערכת נועדה להיות כמה שיותר אבסטרקטית ונוחה לשימוש חיצוני.

משתמשי המערכת:

כל מי שרוצה להשתמש במערכת צריך רק לקרוא לפונקציה RunProxyAsync דרך המחלקה Proxy.cs בקוד שלו. מחלקת Program.cs עושה זאת מראש, אם ברצונכם רק להפעיל את המערכת וזהו פשוט תריצו אותה.

רפלקציה

הפרויקט הזה הוא ללא ספק הפרויקט המושקע ביותר שיצא לי לעבוד עליו, וממש לא רק מבחינת הקוד. אני למדתי כל כך הרבה במהלך הכנת הפרויקט על שפת #C, תקשורת, קצת WinAPI והרבה נקודות מחשבה כמו שימוש באסינכרוניות לעומת multithreading, וזה החלק שאני הכי גאה בו בפרויקט. הלמידה האינטנסיבית יחסית נמשכה כמה שבועות, וככל שהתקדמתי גיליתי עוד ועוד נושאים חדשים, מה שבסופו של דבר חיזקו אותי מאוד בתור מתכנת וללא ספק עזרו לי במקומות אחרים בחיים כמו במיונים לצבא ובראיונות עבודה.

האתגר הראשון הגדול שנתקלתי בו הוא למידת תכנות מונחה עצמים ברמה גבוהה.
רציתי להגיע לשלב הכתיבה של הפרויקט כאשר אני מצויד בכלים רבים שיעזרו לי לכתוב קוד בצורה יעילה וקריאה, ולכן החלטתי שקודם כל אני אלמד תכנות מונחה עצמים לעומק. למדתי מספרים ומצגות ונכנסתי לפרטים הקטנים על מנת לוודא שאני באמת מבין את הנושא ברמה מכובדת.

האתגר השני הגדול שנתקלתי בו הוא למידת שפת #C ברמה גבוהה יחסית על מנת להקל על תהליך הפיתוח עצמו.
נעזרתי בספרים, באתרים ובהרצאות ביוטיוב על מנת ללמוד נושאים כמו delegates, events, exception handling, design patterns וכו'. למדתי מהמקורות שהיו הכי "משעממים" משום שהרגשתי שהם באמת הסבירו לעומק את הנושאים, ובגלל שבאמת רציתי להשתפר המקורות האלה נראו לי הכי מעניינים.

האתגר השלישי הגדול שנתקלתי בו הוא שימוש בתכנות אסינכרוני (ובפרט בשפת #C).
גם פה, למדתי מכל המקורות הכי "יבשים" ובמשך דיי הרבה זמן משום שלא הסתפקתי בהבנה שטחית, אלא רציתי להבין גם מה הולך מאחורי הקלעים שם נתקלתי בקונספט של state machine ובנושאים מעניינים דומים.

האתגר השלישי הגדול שנתקלתי בו הוא הבנת השימוש בפרוטוקול HTTP דרך פרוקסי.
ישנם הרבה מקורות באינטרנט המסבירים על הפקודות של הפרוטוקול, אך לא מצאתי כמעט שום מקור שמסביר על תפעול נכון של שרת HTTP. הייתי צריך לפעול לפי תחושת בטן בחלק גדול מכתיבת הפרויקט, ולעיתים זה גרם לכך שימים שלמים של תכנות קוד הלכו לפח משום שלא הבנתי באמת את הבעיה שהייתי צריך לפתור. לאחר הרבה חיפושים, קריאה וניסויים הצלחתי להבין איך באמת לתקשר דרך פרוקסי לפי HTTP. מה שנראים כמו דברים פשוטים, למשל החזרת תשובת Bad Gateway כאשר לקוח מבקש להתחבר ליעד לא קיים, דרשו המון השקעה ואני מקווה שתוכלו להזדהות איתי.

משום שעבדתי בצורה שיטתית ותוך תכנון עקרוני מוקפד מראש, אין חלקים שהייתי מיישם אחרת בפרויקט, וכשהיו כאלה שיניתי אותם.
אם היו ברשותי משאבים נוספים, כמו מחשב חזק לשימוש המערכת שנמצא ברשת משלו, הייתי רוצה לעשות port forwarding כך שהשרת יוכל לשרת אנשים מכל רחבי העולם ולא רק בתוך רשת ה-LAN.

ביבליוגרפיה

- אתר Chilkat Software – במאמר WSAECONNABORTED – An understandable explanation.
- ערוץ Computerphile ביוטיוב
- אתר MDN Web Docs
- אתר Microsoft Learn
- ערוץ Questpond ביוטיוב
- ערוץ Rainer Stropek ביוטיוב
- אתר Stack Overflow
- רשתות מחשבים, עומר רוזנבוים; ברק גונן; שלומי הוד
- תכנות מונחה עצמים ב-#C, ארז קלר

ועוד מספר אתרים/ערוצים ביוטיוב שלצערי לא רשמתי במהלך הכנת הפרויקט.

קוד

Proxy.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;
using static Proxy.ProxyUtils;

namespace Proxy
{
internal static class Proxy
{
private static int _bufferLength = 8192;
private static Socket _proxySock;
private static Logger _netLogger = new Logger();

// The dictionary tracks clients.
// Each client has a host name, socket, connection lock and connection bool
// Example: https://www.google.com:443, Socket, Lock (object), connecting?

private static readonly ConcurrentDictionary<Peer, Peer?> _peerDic = new ConcurrentDictionary<Peer, Peer?>();
static long totalCliHistory = 0;

public static async Task RunProxyAsync(int port)
{
// Create the IpEndPoint needed for binding
IPEndPoint ipEndPoint = new IPEndPoint(IPAddress.Any, port);

StartAccepting:
try
{
// Create the socket
_proxySock = new Socket(AddressFamily.InterNetwork,
SocketType.Stream,
ProtocolType.Tcp);

// Bind the socket and configure listen amount
_proxySock.Bind(ipEndPoint);
_proxySock.Listen(100);
Console.WriteLine($"Asynchronous Proxy is running on {IPAddress.Any}:{port}");

// Accept clients and call the handler
while (true)
{
await AcceptHandleAsync();
}
}
catch (SocketException se)
{
// Make sure proxy doesnt crash with no connections
```

```
                    if (se.ErrorCode == 995)
                    {
                        goto StartAccepting;
                    }
                    }
                    catch (Exception ex)
                    {
                        Console.WriteLine($"\n{ex}\n");
                    }
                Console.WriteLine("\n\nPress ENTER to continue...");
                    Console.Read();

                // Release the socket. - from Microsoft
                _proxySock.Shutdown(SocketShutdown.Both);
                _proxySock.Close();
                }

                private static async Task AcceptHandleAsync()
                {
                // Wait for connection attempt
                Socket cliSock = await _proxySock.AcceptAsync();
                // Create a Peer object and add to dict
                totalCliHistory++;

                Peer peer = new Peer(cliSock, $"Client{totalCliHistory}", null);
                _peerDic.TryAdd(peer, null);

                // Start handling the peer
                HandlePeerAsync(peer);
                }

                private static async Task HandlePeerAsync(Peer peer)
                {
                Console.WriteLine($"\nPeer connected from:
                {peer.Sock.RemoteEndPoint}\n");

                // Wait (async) for client messages, build correct format and pass
                forward
                while (true)
                {
                try
                {
                    await WaitForMessageAsync(peer);
                }
                    catch (ObjectDisposedException e)
                    {
                    Console.WriteLine($"\n{e}\n");
                    return;
                }
                    catch (Exception ex)
                    {
                    Console.WriteLine($"\n{ex}\n");
                try
                {
                // Indicates Socket was closed in WaitForMessage()
                    if (ex.Message == "Closed Socket")
                    break;

                    GracefulShutdown(peer);
                }
                finally
                {
                    RemovePeerFromDic(_peerDic, peer);
```

```
                                                                    }
                                                                    }
                                                                    }
                                                                    }
private static void RemovePeerFromDic(ConcurrentDictionary<Peer, Peer?>
                                                        dic, Peer peer)
                                                                    {
                                        // Remove the peers from dictionary
                                                                  try
                                                                    {
            var item = dic.First(kvp => kvp.Key == peer || kvp.Value ==
                                                                 peer);
                                                     dic.TryRemove(item);
                 Console.WriteLine($"\nRemoved {item.Key.HostPort}\n");
                  Console.WriteLine($"\nTotal Clients: {dic.Count}\n");
                                                                    }
                                                            catch { }
                                                                    }

                private static async Task WaitForMessageAsync(Peer peer)
                                                                    {
                           byte[] localBuffer = new byte[_bufferLength];

                                                  // If peer is in dictionary
                                              if (_peerDic.ContainsKey(peer)
                                        || _peerDic.Values.Contains(peer))
                                                                    {
                int received = await peer.Sock.ReceiveAsync(localBuffer,
                                                        SocketFlags.None);
                                          // If received end of connection
                                                        if (received == 0)
                                                                    {
                                                                  try
                                                                    {
                                                   GracefulShutdown(peer);
                                        RemovePeerFromDic(_peerDic, peer);
                                                                    }
                                                            catch { }
                                              // Notify the calling function
                                       throw new Exception("Closed Socket");
                                                                    }
                                 // Get the message and forward control of it
                                   byte[] receivedBytes = new byte[received];
                     Array.Copy(localBuffer, 0, receivedBytes, 0, received);

                 string message = Encoding.UTF8.GetString(receivedBytes);
            //Console.WriteLine($"\n{peer.HostPort} sent: {message}\n");
                  await HandleRequestAsync(peer, receivedBytes, message);
                                                                    }
                                                                 else
                                              throw new ArgumentException();
                                                                    }

            public static async Task HandleRequestAsync(Peer peer, byte[]
                                              receivedBytes, string message)
                                                                    {
                                             // Print to console accordingly
                                             if (peer.Addressee is not null)
                           _netLogger.WriteLog($"{peer.HostPort} sent to
                                {peer.Addressee.HostPort}:\n{message}");
                                                                 else
                           _netLogger.WriteLog($"{peer.HostPort} sent to
                                             {peer.Addressee}:\n{message}");
```

```
                    // Check if data is http by looking for a method
                                    if (CheckHTTP(message))
                                                            {
            await HandleHTTPMessage(peer, receivedBytes, message);
                                                        return;
                                                            }
                    // Try sending to and receiving from the server
                    await HandleEncryptedMessage(peer, receivedBytes);
                                                            }
private static async Task HandleHTTPMessage(Peer peer, byte[] receivedBytes,
                                                    string message)
                                                            {
                                    byte[]? toReturn = null;

                                // Get the host & port from message
(string host, int port, string hostPort) = GetHostNPort(message);

        // Forward request to the addressee if CONNECT is not required
                                    if (!message.Contains("CONNECT"))
                                                            {
                    Peer? destPeer = GetAddressee(_peerDic, peer);
                                                if (destPeer is null
                                    || destPeer.HostPort != hostPort)
                                                            {
                                                // Attempt to connect
            string? answer = await AttemptConnection(peer, host, port,
                                                hostPort, message);
                                        // Encode the relevant message
                                            if (answer != string.Empty)
                                toReturn = Encoding.UTF8.GetBytes(answer);
                                  destPeer = GetAddressee(_peerDic, peer);
                                                            }

                    await ForwardHTTPMessage(destPeer, receivedBytes, null);
                                                            }
                                                        else
                                                            {
                    toReturn = await HandleConnect(peer, host, port,
                                            hostPort, receivedBytes);
                                                            }
                                                // Send response to client
                                                        try
                                                            {
                                                if (toReturn is not null)
                                        await peer.Sock.SendAsync(toReturn);
                                                            }
                                // Close connections and remove from cliDic
                                                        catch
                                                            {
                                                GracefulShutdown(peer);
                                        RemovePeerFromDic(_peerDic, peer);
                                                            }
                                                            }
        private static async Task HandleEncryptedMessage(Peer cliPeer, byte[]
                                                    receivedBytes)
                                                            {
                                // If there is no addresse, throw exception
                                            if (cliPeer.Addressee is null)
                                        throw new ArgumentNullException();

                                                // Get the addressee
                                  Socket destSock = cliPeer.Addressee.Sock;
```

```
                                    string? hostPort = cliPeer.Addressee.HostPort;
                                                                    if (hostPort is null)
                                                                     hostPort = "Client";

                                                                      // Send using Socket
                                                                 if (destSock is not null)
                                                                                         {
                                                        await destSock.SendAsync(receivedBytes);
                                  //Console.WriteLine($"\nForwarded Ecrypted Message to
                                                                            {hostPort}\n");
                                                                                         }
                                                                                         }

        private static async Task<byte[]?> HandleConnect(Peer peer, string? host,
                                                                                int port,
                                                                     string hostPort, byte[]
                                                                              receivedBytes)
                                                                                         {
                                    // Get the relevant addressee, connect if necessary,
                                                        // forward the message and the response
                                                                            string? answer;
                                                  Peer? destPeer = GetAddressee(_peerDic, peer);
                                        if ((destPeer is null || destPeer.HostPort != hostPort)
                                                                             && host is not null)
                                                                                         {
                                                                     // Attempt to connect
                                  answer = await AttemptConnection(peer, host, port, hostPort,
                                                                                        null);
                                                                   if (answer != string.Empty)
                                                          return Encoding.UTF8.GetBytes(answer);
                                                                                 return null;
                                                                                         }
                                                                                 return null;
                                                                                         }

                  private static async Task ForwardHTTPMessage(Peer destPeer, byte[]
                                                                   receivedBytes, string? hostPort)
                                                                                         {
                                                                         // Get the relevant socket
                                                                   Socket destSock = destPeer.Sock;

                                                                     if (hostPort is null)
                                                                      hostPort = "Client";
                                                        await destSock.SendAsync(receivedBytes);
                                                                                         }
                     private static async Task<string> AttemptConnection(Peer peer, string
                                                                                  host, int port,
                                                                               string hostPort,
                                                                               string? message)
                                                                                         {
                                                                    string toReturn = string.Empty;

                                                         // Check if destination is accepted by proxy
                                                           // Only http and https are allowed for now
                                                                   if (port != 443 && port != 80)
                                                                                         {
                                                                             // Return Forbidden
                                                                    toReturn = Get403Forbidden();
                                  //Console.WriteLine($"\nReplied to {peer.HostPort} with 403
                                                                                Forbidden\n");
                                                                               return toReturn;
                                                                                         }
                                                                       // Create an HTTP tunnel
```

```
                                                                    try
                                                                    {
                    Socket sock = new Socket(AddressFamily.InterNetwork,
                                                        SocketType.Stream,
                                                        ProtocolType.Tcp);
                                    await sock.ConnectAsync(host, port);

                                Console.WriteLine($"Connected to {host}");
                          Peer destPeer = new Peer(sock, hostPort, peer);
                                                peer.Addressee = destPeer;

                                                        // Add to dictionary
                                        if (!_peerDic.TryAdd(peer, destPeer))
                                                                    {
                                _peerDic.TryGetValue(peer, out Peer? val);
                                  _peerDic.TryUpdate(peer, destPeer, val);
                                                                    }
                                                    if (message is not null)
                                                                    {
                    await sock.SendAsync(Encoding.UTF8.GetBytes(message));
                                                                    }
                                  // Handle the destination like a normal peer
                                                HandlePeerAsync(destPeer);

                                            // respond to client with 200 OK
                                                        if (message is null)
                                                                    {
                                                    toReturn = Get200OK();
          //Console.WriteLine($"\nReplied to {peer.HostPort}  with
                                                          {toReturn}\n");
                                                                    }
                                                                    }
                                                                catch
                                                                    {
                                                     // Return Bad Gateway
                                              toReturn = Get502BadGateway();
 //Console.WriteLine($"\nReplied to {peer.HostPort}  with 502 Bad
                                                            Gateway\n");
                                                                    }
                                                        return toReturn;
                                                                    }
                                                                    }
                                                                    }
```

ProxyUtils.cs

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace Proxy
{
internal static class ProxyUtils
{
/// <summary>
/// Shut down and close the socket connection gracefully.
/// </summary>
public static void GracefulShutdown(Peer peer)
{
// Release the socket. - from Microsoft
peer.Sock.Shutdown(SocketShutdown.Both);
peer.Addressee?.Sock.Shutdown(SocketShutdown.Both);

// Close the sockets
peer.Sock.Close();
peer.Addressee?.Sock.Close();
}
public static async Task ForwardHTTPMessage(Peer destPeer, byte[]
receivedBytes, string? hostPort)
{
// Get the relevant socket
Socket destSock = destPeer.Sock;

if (hostPort is null)
hostPort = "Client";
await destSock.SendAsync(receivedBytes);
}
/// <summary>
/// Returns the peer's pal in the <c>_peerDic</c> dictionary.
/// </summary>
public static Peer? GetAddressee(ConcurrentDictionary<Peer, Peer?> dic,
Peer peer)
{
// find the relevant key - value pair
var item = dic.First(kvp => kvp.Key == peer || kvp.Value == peer);
// get the destination peer
Peer? destPeer = item.Key == peer ? item.Value : item.Key;
return destPeer;
}
public static string Get200OK()
{
// Return the full 200 OK HTTP message
return "HTTP/1.1 200 OK\r\nContent-Length: 0\r\n\r\n";
}

public static string Get502BadGateway()
{
// return the full 502 Bad Gateway message
string badReq = "HTTP/1.1 502 Bad Gateway\r\nContent-Length:
0\r\n\r\n";
```

```
                                                return badReq;
                                                }

                                public static string Get403Forbidden()
                                                {
                                // return the full 403 Forbidden message
string forbidden = "HTTP/1.1 403 Forbidden\r\nContent-Length:
                                                0\r\n\r\n";
                                                return forbidden;
                                                }

                public static (string? host, int port, string hostPort)
                                GetHostNPort(string message)
                                                {
                                                try
                                                {
                                // Decode the HOST header to IP address
                        string host = Regex.Match(message, @"(?<=Host:
                                ).+(?=\r\n)").ToString();
                                if (host == string.Empty)
                                host = Regex.Match(message, @"(?<=host:
                                ).+(?=\r\n)").ToString();

                                string hostPort = string.Empty;
                                int port = 0;

                                // Check if host name includes port
                                if (host.Contains(':'))
                                                {
                        // Remove the port then try converting to int
                                hostPort = host;
                host = Regex.Match(host, @".+(?=:)").ToString();
                                                try
                                                {
                        port = Convert.ToInt32(Regex.Match(hostPort,
                                @"(?<=:).+").ToString());
                                return (host, port, hostPort);
                                                }
                                                catch
                                                {
                                // Return bad request/port
                                throw new ArgumentException("Bad Port");
                                                }
                                                }
                                // Get the correct port by message content
                                if (message.Contains("https"))
                                                {
                                hostPort = host + ":443";
                                port = 443;
                                                }
                                else if (message.Contains("http"))
                                                {
                                hostPort = host + ":80";
                                port = 80;
                                                }
                                return (host, port, hostPort);
                                                }
                                catch { return (null, 0, "Client"); }
                                                }

                        public static bool CheckHTTP(string message)
                                                {
return (message.Contains("GET ") || message.Contains("HEAD ")
```

```
                                        || message.Contains("POST ") ||
                                                message.Contains("PUT ")
                                     || message.Contains("DELETE ") ||
                                           message.Contains("CONNECT ")
                                   || message.Contains("OPTIONS ") ||
                                             message.Contains("TRACE ")
                                       || message.Contains("PATH "));
                                                                    }

public static async Task<byte[]> SendReceiveAsync(Socket destination,
                                        byte[] message, string hostPort,
                                                      int bufferLength)
                                                                    {
              await destination.SendAsync(message, SocketFlags.None);
                      //Console.WriteLine($"\nSent to {hostPort}\n");

                         byte[] localBuffer = new byte[bufferLength];

     int receivedAmount = await destination.ReceiveAsync(localBuffer,
                                                       SocketFlags.None);
                          byte[] toReturn = new byte[receivedAmount];
             Array.Copy(localBuffer, 0, toReturn, 0, receivedAmount);
                     //Console.WriteLine($"\nReceived from {hostPort}:
                              {Encoding.UTF8.GetString(toReturn)}\n");

                                                        return toReturn;
                                                                    }
public static async Task<byte[]> SendReceiveAsync(Socket destination,
                                        string message, string hostPort,
                                                      int bufferLength)
                                                                    {
                      byte[] toSend = Encoding.UTF8.GetBytes(message);
               await destination.SendAsync(toSend, SocketFlags.None);
                     ////Console.WriteLine($"\nSent to {hostPort}\n");

                         byte[] localBuffer = new byte[bufferLength];

     int receivedAmount = await destination.ReceiveAsync(localBuffer,
                                                       SocketFlags.None);
                          byte[] toReturn = new byte[receivedAmount];
             Array.Copy(localBuffer, 0, toReturn, 0, receivedAmount);
                    ////Console.WriteLine($"\nReceived from {hostPort}:
                              {Encoding.UTF8.GetString(toReturn)}\n");

                                                        return toReturn;
                                                                    }
                                                                    }
                                                                    }
```

Peer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Sockets;
using System.Text;
using System.Threading.Tasks;

namespace Proxy
{
internal class Peer
{
// The peer's socket
public Socket Sock { get; set; }
// The host (with the port number) - mainly used for debug
public string? HostPort { get; set; }
// Represents the other Peer this current one is talking to
public Peer? Addressee { get; set; }
public Peer(Socket sock, string hostPort, Peer? addressee)
{
Sock = sock;
HostPort = hostPort;
Addressee = addressee;
}

public static bool operator ==(Peer b1, Peer b2)
{
if (b1 is null)
return b2 is null;

return b1.Equals(b2);
}

public static bool operator !=(Peer b1, Peer b2)
{
return !(b1 == b2);
}

public override bool Equals(object? obj)
{
if (obj == null)
return false;

// Not checking the Addressee's equality
// since it would cause Overflow Exception
return obj is Peer b2 && (Sock == b2.Sock
&& HostPort == b2.HostPort);
}
public override int GetHashCode()
{
return HashCode.Combine(Sock, HostPort);
}
}
}
```

Logger.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Proxy
{
internal class Logger
{
public string CurrentDirectory { get; set; }
public string FileName { get; set; }
public string FilePath { get; set; }
internal Logger()
{
CurrentDirectory = Directory.GetCurrentDirectory();
FileName = "NetworkLog.txt";
FilePath = CurrentDirectory + "/" + FileName;
}

public void WriteLog(string message)
{
// Write to the log file
using (StreamWriter w = File.AppendText(FilePath))
{
w.WriteLine($"{DateTime.Now.ToLongTimeString()} "
+ $"{DateTime.Now.ToLongDateString()}");
w.WriteLine(message);
w.WriteLine("--------------------------------");
}
}
}
}
```

Program.cs

```
using System.Net;
using System.Text.RegularExpressions;
using System.Net.Sockets;

namespace Proxy
{
internal class Program
{
static async Task Main(string[] args)
{
    await Proxy.RunProxyAsync(2109);
}
}
}
```