

# פרויקט גמר – מדעי המחשב

מימוש ומחקר של פרוטוקול BitTorrent בשפת C++

דולב פרנקו

סייבר ומערכות הפעלה



### **פרטי התלמיד:**

שם: דולב פרנקו

תעודת זהות:

נייד

תאריך לידה:

כתובת מייל: [dolevfranco@gmail.com](mailto:dolevfranco@gmail.com)

מקום מגורים:

### **פרטי בית הספר:**

מוסד לימודי:

סמל המוסד הלימודי:

כתובת:

טלפון בית הספר:

### **פרטי המנחה:**

שם המנחה:

תעודת זהות:

תחום עבודה (מקצוע):

פרטי תואר המנחה:

טלפון:

דוא"ל:

# תוכן עניינים

5.....	מבוא
6.....	חלק תיאורטי
7.....	מבוא לרשתות
9.....	מבוא לשפת C++
10.....	פונקציית HASH
11.....	היכרות לעומק עם הפרוטוקולים UDP & HTTP & TCP
11.....	UDP
11.....	TCP
15.....	HTTP
16.....	The BitTorrent Protocol
16.....	Torrent Files
17.....	Bencoding
17.....	Tracker
19.....	Peers
21.....	שרת torrent
23.....	חלק מעשי
24.....	הצגת תכנון ומבנה הפרויקט
25.....	הצגת כלי עבודה למימוש הפרויקט
26.....	תיאור שלבי העבודה על הפרויקט – לוח זמנים
27.....	המחשה ויזואלית של המוצר העובד
30.....	רפלקציה - בעיות ותאגרים בתהליך המימוש והמחקר של הלקוח
33.....	מדריך למשתמש

33	תיאור הקבצים בקוד
35	איך להפעיל את המערכת שלי מ0
36	שלב ראשון – פרסור קובץ torrent
38	שלב שני – התחברות לTracker וקבלת רשימת הפירים
40	שלב שלישי – הורדת הקובץ מהpeers
40	פונקציית update
44	ספריות אשר השתמשתי בהם בהכנת הלקוח
45	ביבליוגרפיה
46	הקוד

## מבוא

מטרת העבודה: פיתוח לקוח שיכול לקבל קובץ torrent ולהוריד את הקבצים אשר הוא מייצג.

עבודת הגמר תעסוק במימוש הפרוטוקול BitTorrent.

הגעתי לפיתוח התוכנה משום שבחופש הגדול ראיתי הרבה סרטים והורדתי אותם דרך תוכנה הנקראת uTorrent. צריך ללכת לאתר נקרא thePiratBay ומשם ניתן להוריד קבצי torrent אשר מייצגים את הסרטים. תוכנה זו הייתה מעין מסך שחור עבוירי והחלטתי שאני רוצה להבין מה קורה בפנים ואיך היא מצליחה לתקשר עם מאוד מחשבים אשר מכילים את אותו קובץ.

העבודה על פיתוח ומחקר התוכנה חשפה בפניי לתחומים מגוונים במדעי המחשב שלא הכרתי וגרמה לי להתפתח הן בחשיבה האלגוריתמית שלי והן במחשיבה היצירתית שלי. העבודה עזרה לי להבין כיצד באמת עובדים נושאים חשובים ובסיסיים במדעי המחשב ובעולם כולו.

בעבודה זו יש סקירה עמוקה של נושאים שונים במחשבים, בניהם: רשתות, C++, תקשורת בין לקוח לשרת ועוד, כאשר התוצר הסופי יהיה תוכנה אשר מקבלת קבצי torrent ומורידה את הקובץ אשר הם מייצגים, בדומה ל-uTorrent.

העבודה מומשה בשפת C++ מכיוון ששפה זו היא שפת בסיס לכל שפות התכנות כיום והיא מאפשרת שליטה על הזיכרון של המחשב, מהירות, יעילות, OOP ועוד.

על מנת להבין את הפרויקט יש תחילה יש להבין היטב את כל הנושא של רשתות ואופן פעולתם ברשת. אתחיל במבוא קצר לרשתות, לאחר מכן מבוא לשפת C++ ולאחר מכן אסביר על הפרוטוקול, על המחלוקות השונות שיצרתי ואופן המימוש.



# חלק תיאורטי



## מבוא לרשתות

מתחילת המהפכה התעשייתית, במאה ה-19, התקשורת החשמלית החלה להתפתח. בעבר (לפני כמאה שנה) כשאנשים רצו לתקשר אחד עם השני הם היו צריכים להעביר מכתבים, שלקח להם הרבה זמן להגיע ולא היה ניתן להבטיח את הגעתם ליעד. כיום, אנחנו רק צריכים לפתוח את מכשירנו האישי ונוכל לשלוח ולהעביר מידע לכל האנשים בעולם בשניות. איך תהליך זה קורה?

בזכות הרשתות שאנחנו מכירים כיום, רשת האינטרנט בעיקר, אנחנו יכולים להעביר מידע ממחשב אחד למחשב אחר. הרשתות פרוסות בכל רחבי העולם והם מבוססת על חבילת פרוטוקולי התקשורת.

### מה הוא האינטרנט?

לרוב האנשים האינטרנט היא ענן אשר מאפשר להעביר קבצים ממקום אחד לשני. במציאות, האינטרנט הוא עשרות אלפי כבלים אופטיים מתחת לאדמה וליים אשר מחוברים זה לזה במיליוני קילומטרים ובניהם עובר המידע הדיגיטלי. פרויקט זה משלב בתוכו נושאים המתקשרים לרשתות ולכן כדי להבין כיצד פועלות הרשתות.

### מושגים חשובים להבנת הרשתות:

- **כתובת IP** – כותבת המורכבת מארבעה בתים, המייצגת נקודת קצה ברשת. לכל נקודת קצה ברשת יש כתובת IP וכך אפשר לשלוח אליה או לקבל ממנה מידע.
- **מודל חמשת השכבות** – המודל מספק הסבר והדרכה כללית על מרכיביה ותפקדיה השונים של הרשת. המודל נוצר על ידי ארגון התקינה הבינלאומי (ISO) המראה ומסביר כיצד צריכה להיראות תקשורת בין מערכת מחשב אחת לשנייה, ללא תלות בייצרן של אותה מערכת. כל שכבה במודל מספקת שירות רק לרמה שמעליה, מבלי לחשוף אותה לאופן בו השירות שלה ממומש. להלן שכבת המודל ותפקידיהם העיקריים:
  1. **השכבה הפיזית** – השכבה אחראית על העברת ביטים (Bits) ממקום אחד למקום אחר על ידי כבלי רשת, סיבים אופטיים, גלים אלקטרומגנטיים ועוד.
  2. **שכבת הקו** – מטרת השכבה היא להעביר מידע בצורה פיזית בין שני ישויות סמוכות.
  3. **שכבת הרשת** – תפקידה של שכבה זו היא למצוא את המסלול הטוב ביותר מנקודת קצה אחת אל השנייה בעזרת נתיבים אשר מנתבים את הפקטות בין הרשתות השונות. (פרוטוקול IP)
  4. **שכבת התעבורה** – שכבה האחראית על שליחת הפקטה אל יעדה ובדיקת אמינות נקודות הקצה. בנוסף השכבה משתמשת בפורטים העוזרים לנתב כל שירות ליעודו. (פרוטוקול TCP)
  5. **שכבת האפליקציה** – משמשת לשימושים שונים לצורכי האפליקציה ומעבירה את המידע בהתאם לפרוטוקולים השונים.

חשוב מאוד להבין את תהליך ואופן פעולת שליחת פקטה באינטרנט גם בשביל הבנת אופן הפעולה של אפליקציית משותפת וגם זהו ידע בסיסי לכל אדם העוסק בתכנות בכל צורה שהיא.

על מנת להבין טוב יותר המושגים הנ"ל וכיצד עובד האינטרנט, אראה דוגמא ואסביר מה התהליך שקורה מרגע בו רושמים בדפדפן את הכתובת [www.facebook.com](http://www.facebook.com), ועד שמופיע לנו על הדפדפן העמוד של פייסבוק.

## רשימת השלבים:

1. המשתמש כותב את הכתובת בדפדפן.
2. האינטרנט בנוי על כתובות IP בלבד, אין לדפדפן דרך לדעת איפה נמצאים השרתים של פייסבוק לפי כתובת ה-URL. לכן הדפדפן מתחיל בתהליך הנקרא DNS Query:
  - הדפדפן מוציא את כתובת הדומיין מתוך ה-URL (למשך מתוך <https://www.facebook.com/hakfar.hayarok> את הדומיין facebook.com) ומחפש במקום ב-DNS Cache האם הכתובת קיימת. ה-DNS Cache – הוא מקום בו מאוחסנים צמדי דומיין ו-IP ששמורים במחשב.
  - אם דומיין קיים והדפדפן מוצא את ה-IP, הדפדפן עובר לשלב הבא.
  - אם הדומיין לא קיים הוא יפנה ל-DNS Server, שיעביר בצורה רקורסיבית לעוד שרתים כדי שימצא או לא את כתובת ה-IP.
  - במידה והדומיין לא נמצא, תחזור הודעת שגיאה.
3. HTTP – בהנחה שהוחזר משרתי ה-DNS כתובת IP נכונה שמייצגת מחשב ברחבי הרשת, בשלב הבא נבנית פקטת HTTP בתוך הדפדפן. במקרה שלנו הבקשה בקבצי ה-Header תכיל את ה- name host ותבקש מפייסבוק להחזיר לנו את דף הבית.
4. בניית נתיב לפקטה עד הגעתה ליעד.
- שימוש ב: Table NAT כל מחשב ברשת הפנימית מקבל כתובת פרטית משלו על ידי הראוטר. לראוטר יש כתובת חיצונית אחת שלרוב היא קבועה. כאשר הראוטר מקבל מידע מאחד המחשבים שמחוברים אליו, נעשה שימוש ב- Table Nat. מטרתה היא להעביר פקטות ממחשבים בתוך הרשת למחוץ לה. הראוטר מקבל מידע מסוים מהמחשבים אליו הוא מחובר, כתובת ה-IP של המחשב מוחלפת בכתובת הראוטר וכן הלאה. הכתובות וה- post המוחלפים יאוחסנו בטבלה הנקראת Table NAT כדי שנדע לאן צריך להחזיר את המידע שמתקבל.
5. כך, באמצעות Tables Routing ומושגים שהוגדרו קודם לכן הפקטה מגיעה עד לשרת המבוקש עם הכתובת ששייכת ל- facebook.com. שם היא נבנית, ומוחזרת תשובה.
6. התשובה מגיעה אלינו והדפדפן מנתח את התשובה ומציג אותה.



## מבוא לשפת C++

שפת Cpp היא שפת תכנות המבוססת על שפת C שהתפתחה בסביבות שנות ה-80. השפה מיישמת תכנות מונחה עצמים, תכנות גנרי ותכנות פרוצדורלי. Cpp הינה עד היום אחת השפות הכי פופולריות בקרב מתכנתים בעולם, למרות שיצאו מאז שפות חדשות. שפות רבות כמו JAVA או C# אשר הושפעו ממנה רבות. בשביל להבין את העקרונות של השפה, להימנע משגיאות מיותרות ובשביל לעשות דברים בדרך הכי יעילה שרק אפשר חשוב לדעת מה עומד "מאחורי הקלעים", כלומר מה באמת קורה שאנחנו מריצים תכנית בשפת Cpp. חשוב להבין את השפה לעומק. בפרק זה, אסביר על תהליך הקימפול של תוכנת cpp.

### תהליך הקימפול של cpp:

- בשלב הראשון, pre-processor, עבור כל קובץ cpp, נוצר קובץ זמני שהוא קובץ עם תוכן של כל הקבצים להם עשינו include. בנוסף הוא מחליף את כל הערכים שעשינו להם define בערכם האמיתי.
- בשלב השני קורה תהליך של קומפילציה, עבור כל הקובץ שנוצר לו pre-processor, הוא בודק שגיאות קומפילציה (שגיאות של משתנים לפונקציות לא קיימות, אי דיוק ב type של המשתנה). לאחר הבדיקה, בהנחה שלא היו שגיאות, נוצר עבור כל קובץ cpp קובץ obj אשר מכיל את הקוד בשפת מכונה.
- השלב השלישי והאחרון – שלב הלינקר. הלינקר מחפש באיזה קובץ obj יהיה פונקציית main (אם יהיה 0 או 2 כאילו, נקבל שגיאת לינקר). עבור כל פקודה ב main, הוא מחפש אש המימוש שלה באחד מקבצי obj. במידה ונמצאו כל המימושים לפונקציות, הלינקר יוצר קובץ הרצה exe.

עכשיו, כשמבינים את התהליך הקימפול של תוכנה ב Cpp, יהיה קל יותר לתכנת, להבין שגיאות מסוימות ולהחליט האם נרצה להשתמש בדרך אחת על פני השנייה כאשר מתכנתים בשפה הזו.

## פונקציית HASH

פונקציית HASH היא פונקציה שממירה קלט באורך משתנה לפלט באורך קבוע, בדרך כלל קצר בהרבה. באופן כללי פונקציית האש תיתן את אותו הפלט בעבור קלטים שונים, אבל פונקציית האש טובה היא פונקציה שבהסתברות גבוהה תפיק פלט שונה עבור כל קלט שונה. לפונקציות כאלה נעשה שימוש רב בחיפוש, מיון והצפנה.

בפרוטוקול ה BitTorrent השימוש ב-HASH נעשה פעמים רבות. בקובץ הטורנט יש את ה infohash שלמעשה הוא HASH של כל החלק של ה Info של הקובץ טורנט המקורי. הוא מחושב על ידי SHA1. בפרוטוקול יש גם Table HASH שמעבירה SHA1 על כל חתיכה ברגע שהיא מסתיימת להיות מורדת ובודקת אם הוא מתאים ל HASH הראשוני, ובמידה ולא המידע נזרק.

כאן ניתן לראות את שדה ה Pieces :

```
6:pieces760:y"Šžù*pf"0£-Ê(ETX, 1F(EMjÇ÷ 'S1NAK5÷|ÜBELÂîâ?î"<(EOTIEM-k-î=ZYDLE-c'BDLE; jÂ!EOTIFSBS~îDUSP³kpÃ
æ.GSËiDLE
a_§-zCH"»$ESCS"-_JS0Âò
îÊ/ÊYÜB«$S0`aNUL0SYNfßVT:;%0(ENOîs}+DC2VTEx...n•zÊze%, rp3´²ÓFFSYNWªÃeÝîsESCw,Ó,,ß&!¿¿<R9SOHîâ:œSOs!RSDC4\
^f´ââ†<šéäGS(SYN†FS"9SYNâ,,DC4
ÑAV£-E(DC1MÜK
.US;j (ETBµÉNîÜ(ENO0NAK¶bî¿ÆðSYNB USÉ{MR(ENO}9MÂ£|>šp=BELÆ€ko°$+µST™
ÜDLE,,KE(EOT05bbL-BXÂî¿DC1æIè†...
ACK~îKäîä` öK-RS(FSpEä
'ESCDLEÑACK¶î~*DC4(EOTI$ETXL¿ *DÝU¥š$×"(ec»Âð> dî¶c ÆIA6,âhVTR;9@ð
Ž`±¿îUÆr&p±ÃóXÜ÷£¹FS¢+Nâ±Â(ETX1î
Ñ³¿fI/|î€DC3IÉ+µjVîÂEZ(ETB<xÜæäý\
K2ääÝ+ú_j{#xî)¶-·ÃIŠ¿ w#I×,ðZð6YLGSUÓZ
(ETB^,,´BÊ?kJpSO¢ñ\}ñ&IÝ23¿<´ÉÝý¶"--4NNUlqOnCANDLEDtë-SOH0áw²=1òÜDLEIWÄ5`QST¹|?ÉBFO
...´}°û¥Êx>€Üs;-G¿¿¿bCAN40!00!¶ùRR&µ,,-0ÄIwUunFSw³ÚyÄß1})bDC3/t...Êëüàšpàv~€US,qnoÜt {6(EOTL3¿<ræèðÎ
b²ëüfð°WóÃ[æš
ÀœSOP7.ÄzA¿ü3¿Ü0ZDîî~ÝÝòY•>]GS,)æaI_)VNñ@-ŽY(EMùqæeß~Wóóó~ÜYtLVTFN¥(EOTm...MŠ(ESCÑSôb$;dòSUB]~îÜ,,èðšsJÑ
DC3t×T$;ör"çÜ
´ÜEVª"0RS:â0«[STX_wî"îŠHD27aBÜß<¶îVTîë_je6:locale2:en5:title28:Adasport - Eagles. New SPORTe
```

## היכרות לעומק עם הפרוטוקולים UDP & HTTP & TCP

במהלך מימוש ה BitTorrent יש צורך בהמון שימוש בתקשורת. אני בפרויקט השתמשתי ב udp בשביל לתקשר עם trackern ובפרוטוקול TCP בשביל לתקשר עם ה peers ולקבל מהם מידע בצורה טובה ואמינה. לכן, חשוב תחילה להבין את שני המושגים האלו לעומק. ישנו גם עוד פרוטוקול שימושי מאוד שבחרתי להסביר כאן והוא ה HTTP, אמנם הקליינט שאני מימשתי לא תומך בו, אבל הביטורנט פרוטוקול תומך בו, בחלק מהמקרים לפי מה שכתוב בקובץ הטורנט.

### UDP

UDP או User Datagram Protocol נועד לתחליף ל TCP, הפרוטוקול נמצא בשכבה הרביעית (שכבת התעבורה) ומתבסס על שכבת הנתוורק (IP). הוא מספק שני דברים ששכבת ה IP לא מספקת: פורטים וצ'קסאם על מנת לוודא שהמידע הגיע נכון. ב UDP החיבור לא אמין, ואין הבטחה שכל המידע יגיע, או שיגיע בסדר הנכון: פקטות יכולות לאבד בדרך או להגיע בסדר הלא נכון. אפליקציות שהאמינות

של המידע לא חשובה להם יכולים להשתמש ב UDP שמשמש פחות במאשבים וצורך פחות זמן.

Source Port Number(16 bits)	Destination Port Number(16 bits)
Length(UDP Header + Data)16 bits	UDP Checksum(16 bits)
Application Data (Message)	

### TCP

TCP או Transmission Control Protocol הינו אחד הפרוטוקולים השימושיים ביותר ברשת כיום. הוא נמצא בשכבה הרביעית ומתבסס על השירות של שכבת הנתוורק (IP). פרוטוקול ה TCP מספק מעבר מידע אמין ומסודר בין נקודות קצה ברשת. הוא אף דואג לטפל בשגיאות. תוכנות רבות משתמשות בפרוטוקול זה כמו: Wide Web, FTP, email World ועוד המון.

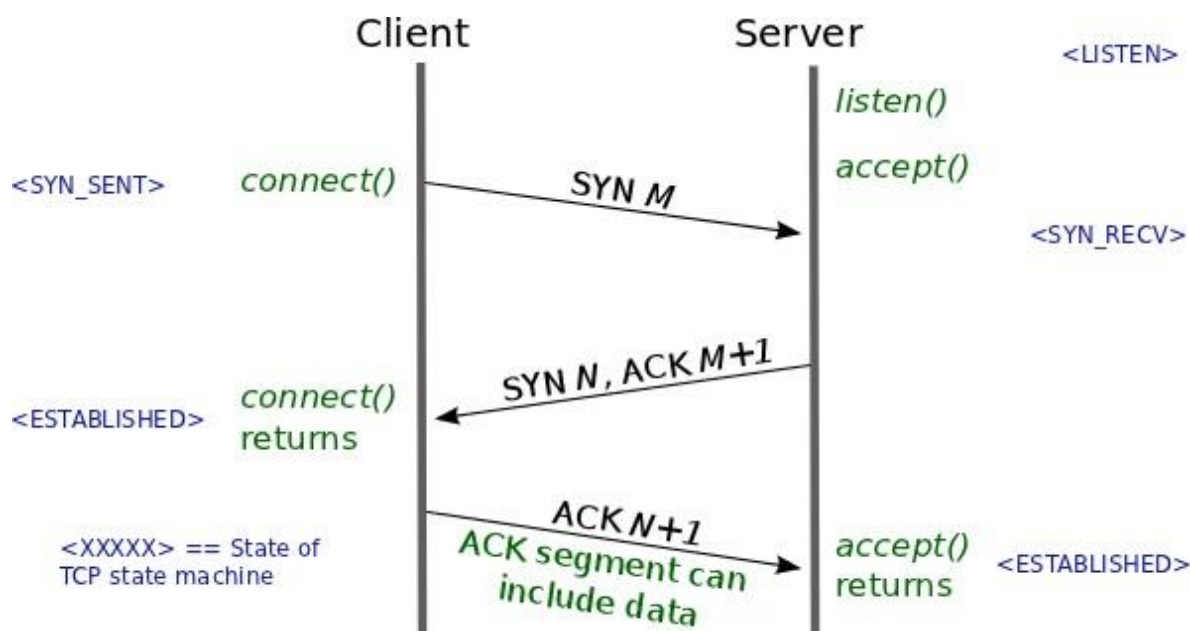
נחלק את הפרוטוקול לשלושה שלבים מרכזיים: תחילת התקשורת, מהלך התקשורת, וסיום התקשורת.

## תחילת תקשורת

יצירת חיבור "Three-Way Handshake": בשביל ליצור חיבור, TCP משתמש ב-Three-Way Handshake.

בשביל ליצור חיבור שלוש לחיצות יד חייבות להתרחש:

- **SYN**: הלקוח מבקש לעשות Synchronize לשרת. "מספר רצף" אקראי לחלוטין מוגרל ומוכנס לפקטה. ("מספר רצף" - יוסבר בהמשך).
- **ACK-SYN**: השרת מאשר את ה SYN של הלקוח, ומבקש בחזרה לעשות synchronize למי שפנה אליו.
- **ACK**: הלקוח מאשר את ה SYN של השרת, ובשלב זה לאחר שהשרת קיבל את האישור, נוצר חיבור דו-כיווני אמין בין שני ה hosts.



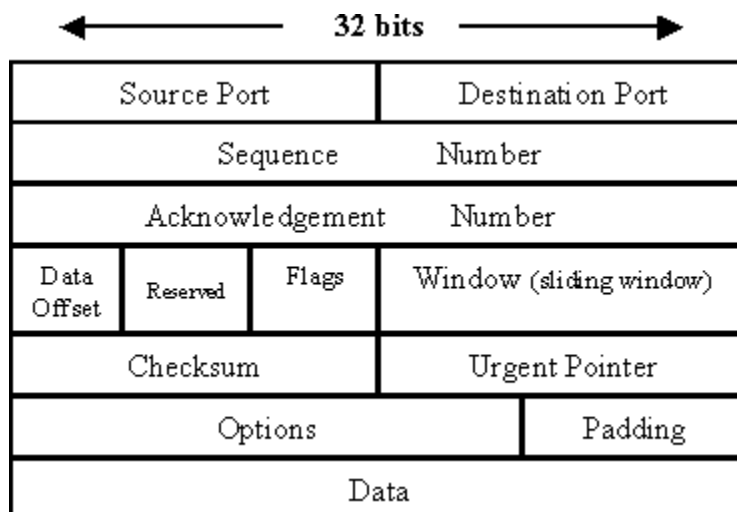
## במהלך התקשורת

כפי שצוין מקודם, חיבור TCP הוא אמין, בטוח ונותן מענה לטיפול בשגיאות. בחלק זה אספר מעט איך הוא מטפל ודואג לכל דברים אלו. TCP עושה זאת בעזרת tcp sequence number. בתחילת החיבור בין שתי נקודות קצה, כל צד מגריל מספר רצף אקראי לחלוטין ושולח אותו ב SYN Packet. מרגע זה והלאה בכל פקטה ישלח ACK עם מספר הבתים שיתקבלו עד עתה וגם sequence number שמציג את ההתחלה של המשך המידע. באופן זה ניתן לוודא שכל המידע יתקבל, ושרצף הפקטות לא השתנה. אם משהו השתנה מהסדר התקין הפרוטוקול מבקש בקשה חוזרת או שולח את הבקשה מחדש ומטפל בבעיה.

## בסיס התקשורת

צד הלקוח מחליט שהוא רוצה לסיים את החיבור, שולח ack אחרון על המידע. אחרי, הוא שולח בקשה עם דגל ה FIN דולק, דבר המראה כל כך שהוא רוצה לסיים את החיבור עם השרת. אחרי זה השרת שולח לו שהוא קיבל את הרצון של הקליינט לנתק עימו קשר וגם שולח פקטה בחזרה עם דגל ה FIN דולק. הקליינט שולח ACK שהוא קיבל, number sequence אחרון, ובסוף השלב הזה שני הצדדים מנותקים.

להלן המבנה של פקטה על פרוטוקול ה TCP:



## הסבר על המבנה:

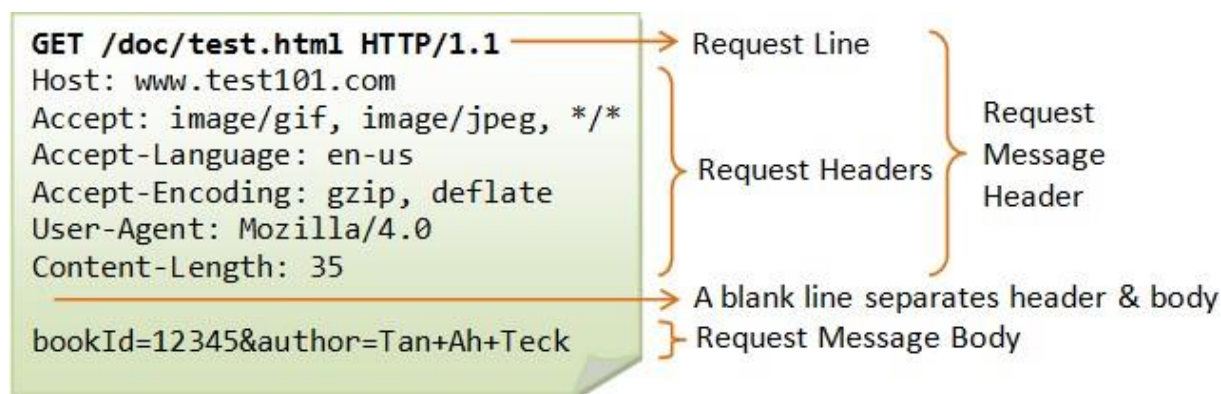
- Source port: מזהה את פורט המקור ממנו נשלחה הפקטה.
- Destination Port: מזהה את פורט היעד אליו נשלחת הפקטה.
- Acknowledgement Number field: מכיל את הערך של ה"מספר רצף" הבא של הבייט שהשולח מצפה לקבל (אם דגל ה ACK דלוק).
- Sequence Number: מספר מזהה של הבייט הנוכחי של השולח.
- Data Offset (a.k.a. Header Length) field: מגדיר את גודלו של ה TCP header בגדלים של 30 ביט וורד. מספר בתים מקסימלי הוא 02 והמקסימום הוא .
- Reserved: שמור להרחבות עתידיות.
- Flags: מכיל 3 דגלים של ביט אחד כל אחד מהם (כאן אסביר על 6 דגלים שימושיים):
  - URG – מראה על כך שיש מידע "דחוף" בפקטה.
  - ACK – מראה על כך שיש מספר Acknowledgment.
  - PSH – מראה על כך שמידע צריך לעבור לאפליקציה הכי מהר שאפשר.
  - RST – עושה ריסט לחיבור.

- SYN – מסנכרן שני צדדים על מנת להתחיל חיבור.
- FIN – מראה שהשולח סיים להעביר מידע.

## HTTP

קיצור של Hypertext transfer protocol. פרוטוקול משכבת האפליקציה מעל שכבת TCP/IP. הוא מתפקד כפרוטוקול בין צד לקוח לשרת עם בקשה - תגובה. Http Servers לרוב יגישו דפי HTML ללקוחות (לדוגמה דפדפנים וכו').

להלן דוגמה לפקטת GET של לקוח בפרוטוקול HTTP:



### שדות ב Header של פקטת HTTP:

- **Accept:** מגדיר סוגי תשובות שיכולות להתקבל כתשובה, אם יש יותר מאחד אפשר להפריד אותן בנקפס.

```
Accept: text/plain; q=0.5, text/html, text/x-dvi; q=0.8, text/x-c
```

- **Accept-Charset:** מגדיר את ה Chars Set שיכולים להתקבל בתור תשובה.

```
Accept-Charset: iso-8859-5, unicode-1-1; q=0.8
```

- **Authorization:** מגדיר מזהה של משתמש, ל resource המבוקש

```
Authorization: BASIC Z3Vlc3Q6Z3Vlc3QxMjM=
```

- **Cookie:** מכיל זוג של שם משתנה וערך שיהיה שמור במחשב לפי URL.

```
Cookie: name1=value1;name2=value2;name3=value3
```

- **Host:** מגדיר את ההוסט באינטרנט. מגדיר גם פורט. אם לא נותנים פורט, הפורט 82

```
Host : "Host" ":" host [ ":" port ] ;
```

- **Referer:** מגדיר את ה URI שהקליינט בא ממנו. אם הקליינט בא מהכתובת הרשומה אזי שדה referer יהיה כך:

```
Referer: http://www.tutorialspoint.org/http/index.htm
```

## The BitTorrent Protocol

BitTorrent הינו פרוטוקול תקשורת P2P שנכתב על ידי בראם כהן. הפרוטוקול מיועד להעברה והורדה של קבצים. בתקשורת client - server רגילה, הכל עובר דרך השרת בעל רוחב פס מוגבל. במקרה של ריבוי משתמשים, קצב ההורדה יקטן וייווצרו בעיות. הפרוטוקול BitTorrent יכול לשמש להקלה על שרתים שכאלה. הוא עושה זאת מכיוון שלקוח מוריד קובץ מסוים, השרת שומר את המזהה של המחשב ומשתמש בו לאחר מכן גם כשרת הורדה ולקוח ממנו חלקים מהקובץ שהוא הוריד וכך מחלק את העומס על כמות גדולה של מחשבים.

התוכנה עובדת עם קבצים בעלי הסיומת torrent. קבצים אלו הם למעשה מכילים תוכן המצביע על הפניות למידע הדרוש על מנת להוריד את הקובץ המבוקש. קובץ ה-torrent מפנה את התוכנה לTracker, שהוא למעשה שרת המכוון את כל התנועה בין המשתמשים המורידים את אותו קובץ. ביטורנט שולחת ומקבלת מספר חלקים מקובץ ההורדה בו-זמנית, על פי הוראות ה-Tracker. היכולת לשלוח ולקבל חלקים שונים של הקובץ בו זמנית, ממשתמשים שונים, מאפשרת לביטורנט לשמור על קצב העברה גבוה מאוד.

משלב זה של פרק זה, אסביר באופן טכני את הפרוטוקול. תחילה אסביר את המבנה ואת הרקע על Torrent Files. לאחר מכן אסביר מה הוא אופן הפעולה של הפרוטוקול: אסביר מי הוא ה-Tracker, ואת התהליך של קבלת קובץ ושליחה / העברה של קובץ. יש להדגיש שבפרק זה בחרתי לכתוב אך ורק את הקטעים הטכניים של הפרוטוקול. שלבים אלגוריתמיים, דרך מימוש וטיפול בשגיאות אסביר בהמשך, בפרק בו אסביר על המימוש של הפרוטוקול.

## Torrent Files

קבצי טורנט הם קבצים בינאריים, בעלי סיומת torrent אשר מכילים metadata על הקבצים חיצוניים שנרצה להוריד אותם, הם לא מכילים את התוכן של הקובץ המורד / מועלה או כל דבר בסגנון זה. יש לציין שייצוג המידע בקבצי הטורנט הינו מקודד בBenconding עליו אסביר בהמשך.

### מבנה:

- Announce – כתובת ה-Tracker
- Info – מילון המכיל מידע על הקבצים וקובץ:
  - Name – שם הקובץ, איך הוא צריך להישמר. אם יש כמה קבצים זה יהיה השם של התיקיה
  - Piece length – אורך של כל piece בבתים
  - Pieces – רשימת hash שיש על כל piece שמתקבל
  - Files – אם יש קבצים מרובים העמודה הזאת מופיעה. רשימה של קבצים שלכל קובץ יש את התכונות הבאות:
- Path – שם הקובץ
- Length – אורך הקובץ בבתים



דוגמא לקובץ torrent :

[illegible]

## Bencoding

בינקאודינג זו דרל לייצוג מידע בפורמט מסוים, יש תמיכה במחרוזות, מילונים, מספרים ורשימות. נעשה בו שימוש אך ורק בקבצי torrent.

## :Strings

- <string length encoded in base ten ASCII>:<string data>
- דוגמא - hello:5 מייצג את המחרוזת hello

## :Integers

- `i<integer encoded in base ten ASCII>e`
- דוגמא – `i24e` מייצג את המספר 24, `i-2e` מייצג את המספר -2.

## :Lists

- `l<bencoded values>e`
- לדוגמה - `l4:spam4:eggse` מייצג את הרשימה `["spam", "eggs"]`

## :Dictionaries

- `d<bencoded string><bencoded element>e`, Key חייב להיות מסוג מחרוזת
- דוגמא - `d3:cow3:moo4:spam4:eggse` מייצג את המילון `{ "cow" => "moo", "spam" => "eggs" }`

## Tracker

ה - Tracker הוא סוג של שרת HTTP שעוזר בקיום ובהתחלת התקשורת בין peers בפרוטוקול ה BitTorrent. ה Tracker נשאר מעודכן בנוגע למיקום של קבצים שונים על מחשבי ה Peers. בתחילת תהליך ההורדה, הבקשה הראשונה של הקליינט שמממש את פרוטוקול ה BitTorrent תהיה מופנית ל Tracker, שיחזיר רשימה של peers

שלחם יש חלקים או את כל הקובץ המבוקש. ותפקיד ה Client יהיה ליצור איתם חיבור עד למצב בו הוא מקבל קובץ שלם מכולם. כתובת ה Tracker נמצאת בשדה ה Announce בתוך ה torrent file.

### תקשורת עם Tracker

אנחנו נצטרך לדבר עם ה tracker בפרוטוקול udp, כל הערבים שנשלחים בבקשה צריכים להיות ב big Indian (הספרה האחרונה היא הספרה בעלת הכי פחות משמעות).

ראשית, לפני שאנחנו מבקשים את המחשבים המכילים את הקובץ אנחנו צריכים לשלוח בקשת התחברות ל tracker.

#### מבנה בקשת ההתחברות:

- protocol\_id (64-bit integer) - מספר מיוחד קבוע, 0x41727101980
- action (32-bit integer) - 0 (הפעולה של התחברות)
- transaction\_id (32-bit integer) - מזהה קישור רנודמלי שהלקוח יבחר

#### מבנה התשובה של tracker אם הוא חי:

- action (32-bit integer) - 0 (הפעולה של התחברות)
- transaction\_id (32-bit integer) - מזהה רנודמלי שהלקוח יבחר
- connection\_id (64-bit integer) - מזהה של החיבור, נשמור אותו בצד ונעזר כל בקשה שנרצה לשלוח ל tracker תשלח עם המזהה הנוכחי.

לאחר שהתחברנו ל tracker, נרצה להשיג את רשימת המחשבים אשר מכילים את הקובץ שלנו, הבקשה שנרצה לשלוח היא Announce request.

#### מבנה בקשה ה announce:

- connection\_id (64-bit integer) - המזהה אשר קיבלנו בבקשת ההתחברות
- action (32-bit integer) - 1 (הפעולה של announce)
- transaction\_id (32-bit integer) - מזהה קישור רנודמלי שהלקוח יבחר
- Info hash (20-byte string) - עשרים בתי האש (sha1 במקרה הזה) של torrent file.
- Peer Id (20-byte string) - מזהה ייחודי של כל לקוח
- downloaded (64-bit integer) - מה שהורד עד עכשיו.
- left (64-bit integer) - מה שנשאר ללקוח להוריד בבתים.
- uploaded (64-bit integer) - מה שהועלה עד רגע מסוים.
- event (32-bit integer)
- completed - נשלח כאשר ההודעה הסתיימה. ○

- Started – נשלח כאשר ההורדה מתחילה.
- Stopped – נשלח כאשר מספיקה ההורדה, אם אפשרי.
- Ip (32-bit integer) – לא חובה, דיפולט – 0. כתובת הIP או הDNS של הpeer עצמו.
- key (32-bit integer) – מפתח רנדומלי
- numwant (32-bit integer) – כמה כתובות של peers אחרים הלקוח רוצה שיחזירו לו, המספר הדיפולטיבי הינו 1-.
- port (16-bit integer) – הפורט שהpeer מאזין, המספרים שיכולים להיות הם בין 6881-6889

### מבנה התשובה של Tracker (נציין רק את השדות אשר מעניינים אותנו):

- peers (32-bit integer\*n):
  - ip – כתובת ip של הpeer
  - Port – הפורט שעליו הpeer מאזין

לאחר שהלקוח מקבל את התשובה שהוא צריך (רשימה של peers), הוא מתחיל ליצור קישור עם כל peer, ואם אחד מהם מוכנים להורדה, הוא מתחיל להוריד עד שיש לו החבילה המלאה.

## Peers

קיבלנו מהתשובה של tracker רשימה של כתובות IP וport. החיבור לכל peer יהיה באמצעות tcp, ב-multi-threads, באמצעות פרוטוקול BitTorrent. אתאר כעת את החיבור לpeer יחיד כתקשורת p2p.

### תקשורת עם peer יחיד:

ראשית, נצטרך ליצור חיבור עם הpeer ולבדוק אם הוא פעיל.

### בקשת handshake:

- Pstrlen – אורך של pstr
- Pstr – שם המזהה של הפרוטוקול, "BitTorrent Protocol"
- Reserved – שמורים להרחבות, כל ההרחבות כרגע משתמשות ב0
- Info hash (20-byte string) – עשרים בתי האש (sha1 במקרה הזה) של הtorrent file.
- Peer\_id – עשרים בתיים של ID ייחודי למשתמש.

## סוגי בקשות שונות עם peers:

כעת אציג את הבקשות השונות שאפשר לשלוח לpeer, אציג אותם כך: <length prefix><message ID>, כל <> מייצג חלק אחר בתוך ההודעה כאשר אני אציג את השם שלו ומה הערך שהוא צריך להיות.

- Keep-alive (<len=0000>) – peers עלולים לסגור את החיבור אחרי זמן מה מבלי הודעות, לכן נצטרך לשלוח הודעה מסוג זו כדי להשאיר את החיבור פעיל. ההודעה ריקה.
- Choke (<len=0001><id=0>) – הלקוח המוריד לא יקבל הודעות מהפיר עד שנשלח Unchoke
- Unchoke (<len=0001><id=1>) – מבטל את ההשפעה של choke
- Interested (<len=0001><id=2>) – הלקוח מעוניין להוריד מידע מהפיר
- Not interested (<len=0001><id=3>) – הלקוח אינו מעוניין להוריד מידע מהפיר
- Have (<len=0005><id=4><piece index>) – מכיל את ערך piece שהורד בהצלחה ואומת על ידי hash
- Bitfield (<len=0001+X><id=5><bitfield>) – יכול להישלח רק אחרי ה"לחיצת יד", הbitfield מייצג את piceses שהורדו בהצלחה
- Request (<len=0013><id=6><index><begin><length>) – משומש כדי לקבל block.
  - Index – מספר מזהה של החתיכה
  - Begin – אופסט בתוך החבילה
  - Length – אורך הבקשה
- Piece (<len=0009+X><id=7><index><begin><block>) – X הוא האורך של block.
  - Index – מספר מזהה של החתיכה
  - Begin – אופסט בתוך החבילה
  - Block – מספר החלק בחתיכה
- Cancel (<len=0013><id=8><index><begin><length>) – לבטל בקשות של block.

## תהליך הבקשות בחיבור עם peer

ראשית, אנחנו נשלח לpeer אליו הגענו בקשה להדנשקה. לאחר שקיבלנו את לחיצת היד שלנו והפיר מוכן לחיבור שלנו, אנחנו נרצה לדעת איזה חתיכות יש לאותו פיר.

יש לנו שתי אפשרויות:

1. הלקוח ישלח על כל חתיכה שהוא צריך לקבל, בקשת have והוא יראה האם ללקוח יש אותה
2. נשלח בקשת bitfield, ועל כל ביט שדולק, לפיר יש את החתיכה המבוקשת

שנית, על מנת לקבוע אם הלקוח המוריד יקבל חבילות מהפיר הוא לשלוח ולקבל מספר בקשות. נשלח בקשת interested בהתחלה ונחכה לקבלת בקשת unchoked, ברגע שקיבלנו אותה, הפיר מוכן להעביר לנו מידע ונתחיל

בבקשת חתיכות עד שיהיה לנו את הקובץ המילה. אם לא קיבלו unchoked, נסגור את החיבור עם הפיר ולא נמשיך לדבר איתו.

שלישית, לאחר שאנחנו יודעים איזה בקשות אנחנו צריכים לבקש מהפיר, אנחנו מתחילים בבקשת החתיכות. נבקש בבקשה את המיקום של החתיכה, האופסט שלה ואורכה. לעיתים, ה-piece length המצוין בקובץ torrent, ארוך יותר ממה שנוכל לבקש בבקשה אחת לכן נוכל לציין אותו ב-blocks, וכל אחד מה-blocks יכול כמה חתיכות. רביעית, כאשר אנו מקבלים את החתיכה, עלינו לעשות עליה פקודת sha1 ולהשוות אותה לתוכן שמופיע בתוכן קובץ torrent, אם הם שווים, תוכן הקובץ לא נערך והכל טוב.

## שרת torrent

נחזור לשלבים הראשונים של הורדת קובץ מטורנט – קבלת רשימת הפירים. אנחנו מבקשים מ-tracker מסוים להביא לנו רשימה של מחשבים שהורידו את הקובץ הזה פעם.

כל לקוח שמוריד את הקובץ, השרת שומר את ה-IP של אותו מחשב ושומר אותו ב-DHT. DHT היא מערכת מבוססת המספקת שירות חיפוש בדומה ל-hash table. צמדי מפתח-ערך מאוחסנים ב-DHT, וכל צומת משתתף יכול לאחזר ביעילות את הערך המשוך למפתח נתון. היתרון העיקרי של DHT הוא שניתן להוסיף או להסיר צמתים עם מינימום עבודה סביב הפצה מחדש של מפתחות. מפתחות הם מזהים ייחודיים אשר ממפים לערכים מסוימים, אשר בתורם יכולים להיות כל דבר, החל מכתובות, למסמכים ועד לנתונים שרירותיים. האחריות לתחזוקת המיפוי ממפתחות לערכים מתחלקת בין הצמתים, באופן ששינוי במערכת המשתתפים גורם לכמות מינימלית של הפרעה. זה מאפשר ל-DHT להתאים למספרים גדולים במיוחד של צמתים ולטפל בהגעות, יציאות וכשלים מתמשכים של צמתים.

בשימוש ב-DHT ניתן לא לסמוך על אף יישות ברשת ולכן יש שימוש רב ב-DHT במערכות אשר משתמשות בפרוטוקול P2P.

יש לציין שאני לא אכין שרת BitTorrent משום שזוהי עבודה קשה מאוד, וזה לא המימוש שרציתי להתרכז בו באפליקציה. כדי להדגים שאני יודע לבנות שרתים מרובי לקוחות, בניתי אפליקציה בשם Roulette for Whatsapp.

## Roulette for WhatsApp

Roulette for WhatsApp, אתה מתחרה עם החברים שלך כדי לנחש במהירות מי שלח את ההודעה המוצגת. ייצאו כל צ'אט שתראה מ-WhatsApp ושחקו עם הודעות צ'אט אקראיות.

ייצא צ'אט מ-WhatsApp (עקוב אחר מדריך קל), ערוך אותו והתחל משחק. קוד אקראי למשחק שלך יופיע על המסך והחברים שלך יצטרכו להעתיק אותו ולהצטרף למשחק. התחרו עם החברים שלכם במשך 10 סיבובים

וקבלו ציונים (אם צדקתם) לפי המהירות שלכם. בכל סיבוב, מוצגות על המסך 5 הודעות אקראיות של אותו אדם ואתם צריכים לנסות ולנחש של מי.

כתבתי את האפליקציה בשפת Flutter, dart ואת השרת בשפת javascript עם webSockets. כאשר משתמש יוצר משחק, הוא יוצר בעצם "חדר" חדש של socket. כדי להצטרף לחדר, שאר החברים יזינו את הקוד וישלחו בקשת joinGame לשרת. לאחר שמצטרף שחקן, נשלח הודעה לכל שאר המשתמשים על הצטרפות משתמש חדש.

```
socket.on('joinGame', (username, gameId) => {
  if (!isRoomExist(gameId)) return socket.emit('joinGame', generateJoinGameMsg(false, 'Game not found'))
  if (isGameStarted(gameId)) return socket.emit('joinGame', generateJoinGameMsg(false, 'Game already started'))
  if (getUsersInRoom(gameId).length >= 7) return socket.emit('joinGame', generateJoinGameMsg(false, 'Game is full'))
  const user = addUser({id: socket.id, username, room: gameId})
  if (user.error) return socket.emit('joinGame', generateJoinGameMsg(false, user.error))
  socket.join(user.room)
  io.to(user.room).emit('users', getUsersInRoom(user.room))
  socket.emit('joinGame', generateJoinGameMsg(true, ''))
  socket.emit('gameData', getGame(gameId))
})
```

לאורך כל המשחק נשלח כל הזמן מידע על תוצאות המשתמשים והמיקום שלהם. ניתן להוריד את המשחק בגוגל פליי בקישור הבא: <https://play.google.com/store/apps/details?id=com.roulette.forwhatsapp>

קוד המקור לאפליקציה נמצא בGitHub בקישורים הבאים: <https://github.com/dtkdt100/Roulette-for-Whatsapp-server>

הקוד נמצא בprivate משום שאני רוצה שיעתיקו את האפליקציה, ניתן לשלוח לי אפשרות לגישה ואני אאפשר.

# חלק מעשי

```
document.getElementById(div).innerHTML = errorMessage;
else if (i==2)
{
    var atpos=inputs[i].indexOf("@");
    var dotpos=inputs[i].lastIndexOf(".");
    if (atpos<1 || dotpos<atpos+2 || dotpos>inputs[i].length-1)
        document.getElementById('errEmail').innerHTML = "Email is not valid";
    else
        document.getElementById(div).innerHTML = "OK";
}
else if (i==5)
{
    var atpos=inputs[i].indexOf("@");
    var dotpos=inputs[i].lastIndexOf(".");
    if (atpos<1 || dotpos<atpos+2 || dotpos>inputs[i].length-1)
        document.getElementById('errPass').innerHTML = "Password is not valid";
    else
        document.getElementById(div).innerHTML = "OK";
}
```

## הצגת תכנון ומבנה הפרויקט

הפרויקט כתוב בשפת ++c. לפני תחילת כתיבת הקוד תכננתי את הפרויקט וקבעתי באופן כללי את היעדים שאליהם ארצה להגיע. את הפרויקט תכננתי כך שיהיה נוח להוסיף לו עוד דברים ולשנות פונקציונליות בקלות. בנוסף על כך, הקפדתי על קוד נקי ומסודר.

בתכנות עבדתי בשלבים – קודם כל הבנה עמוקה של הפרוטוקול BitTorrent ושל לקוחות ידועים כגון uTorrent ורק לאחר מכן מימוש התוכנה שלי.

בכל שלב עבדתי מלמטה למעלה, לדוגמה: אם הייתי צריך לבצע תקשורת עם tracker או אחד הpeers, בניתי קודם כל מחלקת socket ומחלקת מעטפת אליה (כדי שהsocket יסגר שלא משתמשים בו), כך כל מחלקה תוכל להשתמש במחלקת socket ולא לוודא שהוא נסגר או נפתח. בנוסף לכך, כחלק מהפרמטרים המועברים למחלקת socket אפשרתי את החיבור או בUDP או בTCP כך שלא אצטרך לממש מחלקת socket נפרדת לכל סוג חיבור.

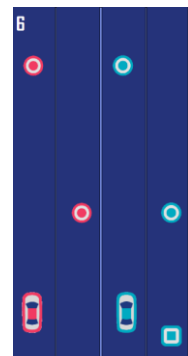
נצטרך לשים בתוכנה נתיב לקובץ torrent, והתוכנה תוריד את החבילה לתוך הנתבי c://downloads (נוכל לשנות אותו). התוכנה תראה כמה אחוזים מחבילה כבר הורדה וכמה נשאר.

יצרתי לתוכנה קובץ של בדיקות בשימוש בgTest של גוגל על קבצים בcpp.

התוכנה עובדת וכל מי שרוצה יכול להשתמש בה, בנוסף ניתן להשתמש בקוד גם כן, הפרויקט הינו עם קוד פתוח ב[GitHub](https://github.com).

### הערות

- משום שלא הייתי צריך ליצור שרת מרובה לקוחות, או שרת בכללי (יצרתי רק לקוח), יצרתי את אפליקציה WhatsApp Roulitte אשר היא מממשת שרת מרובה לקוחות עם webSockets בJS. ניתן לראות הסבר עליה בעמוד מספר 21.
- הממשק האינטראקטיבי בתוכנה שמימשתי אינו מספיק משלב את תגובת המשתמש חוץ מהעלאת קובץ torrent לתוכנה לכן יצרתי פרויקט בcpp בשימוש בWinApi שהוא משחק הנקרא two cars. המשחק מאפשר למשתמש לשלוט בשני מכוניות היכולות לזוז ימינה או שמאלה, על המכוניות נופלים או ריבועים או עיגולים, צריך לאסוף את העיגולים ולהתחמק מהריבועים. בפרויקט ב[GitHub](https://github.com)





## הצגת כלי עבודה למימוש הפרויקט

- Visual Studio – משמש כ-IDE לcpp. התוכנה מאוד נוחה ולא מאוד כבדה, בנוסף לכך ניתן לדבג איתה בצורה מאוד נוחה והיא מאוד מומלצת לקוד בcpp.
  - uTorrent – התוכנה המוכרת ביותר לשימוש בטורנטים. אשתמש בה כדי להוריד אליה קבצי טורנט ולבדוק מה היא בדיוק עושה, זאת אומרת מה היא שולחת לtracker ולpeers.
  - Wireshark – הינה תוכנה אשר מאפשרת לעשות sniff לרשתות במחשב שלך. בנוסף לכך, wireshark נותנת מגוון רחב של פילטרים אשר יאפשרו לנו להסתכל בדיוק על הפקטות שאנחנו נרצה לראות. לדוגמא, נוכל לפלטר על פרוטוקול BitTorrent, ולעשות follow tcp stream וכך לראות בדיוק את הבקשות והתשובות עם peer מסוים.
- נתקלתי בהרבה בעיות עם השליחות של פקטות לpeer (משום ששולחים את בצורה בינארית ולא בקשת get וpost רגילה) ולכן נעזרתי המון בשימוש במקביל של uTorrent ושל Wireshark, גררתי קובץ torrent לתוך uTorrent והקלטתי את כל התעבורה ש-uTorrent מבצעת. כך יכולתי לדעת בדיוק איפה הבעיה בקוד שלי ובפקטות שאני שולח.

## תיאור שלבי העבודה על הפרויקט – לוח זמנים

1. הבנה עמוקה של הפרוטוקול BitTorrent.
  - קריאה והבנה עמוקה של הפרוטוקול ואופן הפעולה שלו, קריאה בכמה אתרים, גם באתרים רשמיים וגם בלא רשמיים.
  - קריאה על מימושים שונים של לקוחות BitTorrent והבנה מה ההבדל בניהם
  - מימוש אחד מהשיטות לפרוטוקול שלי
2. הבנה של קבצי טורנט.
  - קריאה על bencode ואיך הוא עובד, איזה מבני נתונים הוא תומך וכו'
  - מימוש bencode בעצמי תוך כדי שימוש באלגוריתמים יעילים ורקורסיבים.
3. הבנה של רשתות והתקשורת של peers ו tracker.
  - קריאה והבנה של פרוטוקולים ידועים ברשת: tcp, udp, http.
  - הבנה של התקשורת וחקירה של uTorrent בכדי להבין שהבנו נכון את התקשורת ואנחנו יודעים כל בקשה אשר קורת.
  - ניסיון ראשוני עם רשתות בעזרת הספר של גבהים, מימוש של שרת-לקוח עם sockets בpython.
4. הבנה עמוקה של שפת התכנות cpp ועקרונות oop.
  - נלמד את השפה מספרים וסרטונים ביוטיוב
  - מימוש פרסור הקובץ טורנט
  - מימוש מחלקות מעטפת (guard classes) socket ול handle כדי שיהיה ניתן לא לדאוג לסגור אותן כאשר המחלקות יוצאות מן הscope. לדוגמא המחלקת מעטפת תעשה CloseHandle.
  - תקשורת עם tracker: שליחת בקשת התחברות, קבלת התשובה ופרסור מתאים.
  - תקשורת עם peers: יצירת מחלקה אשר מאפשרת תקשורת **במקביל** של peer, נשלח בקשה מתאימה לכל peer ונקבל תשובה נכונה.
  - סגירת החיבור והורדת הקובץ למקום המתאים במחשב.

## המחשה ויזואלית של המוצר העובד

1. הורדה של קובץ טורנט מהאינטרנט: אני כתבתי בגוגל, "sample torrent file download", והורדתי כמה קבצי טורנט. בנוסף ניתן להוריד את הקבצי הtorrent מאתר הנקרא The pirate bay. להלן תמונה מהאינטרנט לשיר shape of you:

The screenshot shows a torrent file listing on a website. The title is "Various Artists - Shape of You - 10's Best (2023) Mp3 320kbps [PMEDIA]". The file type is "Audio > Music". It contains 101 files and is 819.97 MiB (859798706 Bytes) in size. It was uploaded on 2023-05-29 11:00 GMT by user 34, with 19 seeders and 19 leechers. The info hash is C13D3FB2F77BA4F35091912ACF3D7879B37087F1. There are two "Download" buttons, one with a download icon. Below the buttons is a large empty text box. At the bottom, there is a "MAGNET" link and another "Download" button. At the very bottom, there are links for "Login", "Register", "About", and "PirateBay".

2. **כתיבת הנתיב של הקובץ לתוכנה:** לאחר שמורידים את הקובץ, יש לשים את הנתיב שלו בתוך התוכנה שכתבתי. התוכנה מייד תבצע את כל הפעולות הנדרשות על מנת להוריד את כל הקובץ בשלמותו (רק בתנאי שהיא מצאה לפחות קישור אחד אשר ממנו היא יכולה להוריד) והיא תאחסן את הקבצים במקום אשר שמוגדר מראש על ידי המשתמש.
3. **התחלת ההורדה:** לאחר השמת הקובץ בתוכנה, התוכנה מייד תתחיל מהתחברות לTracker, היא מקבלת ממנו peer ומתחילה לבקש חבילות מהם. להלן צילום מסך של תהליך ההורדה של הקובץ:

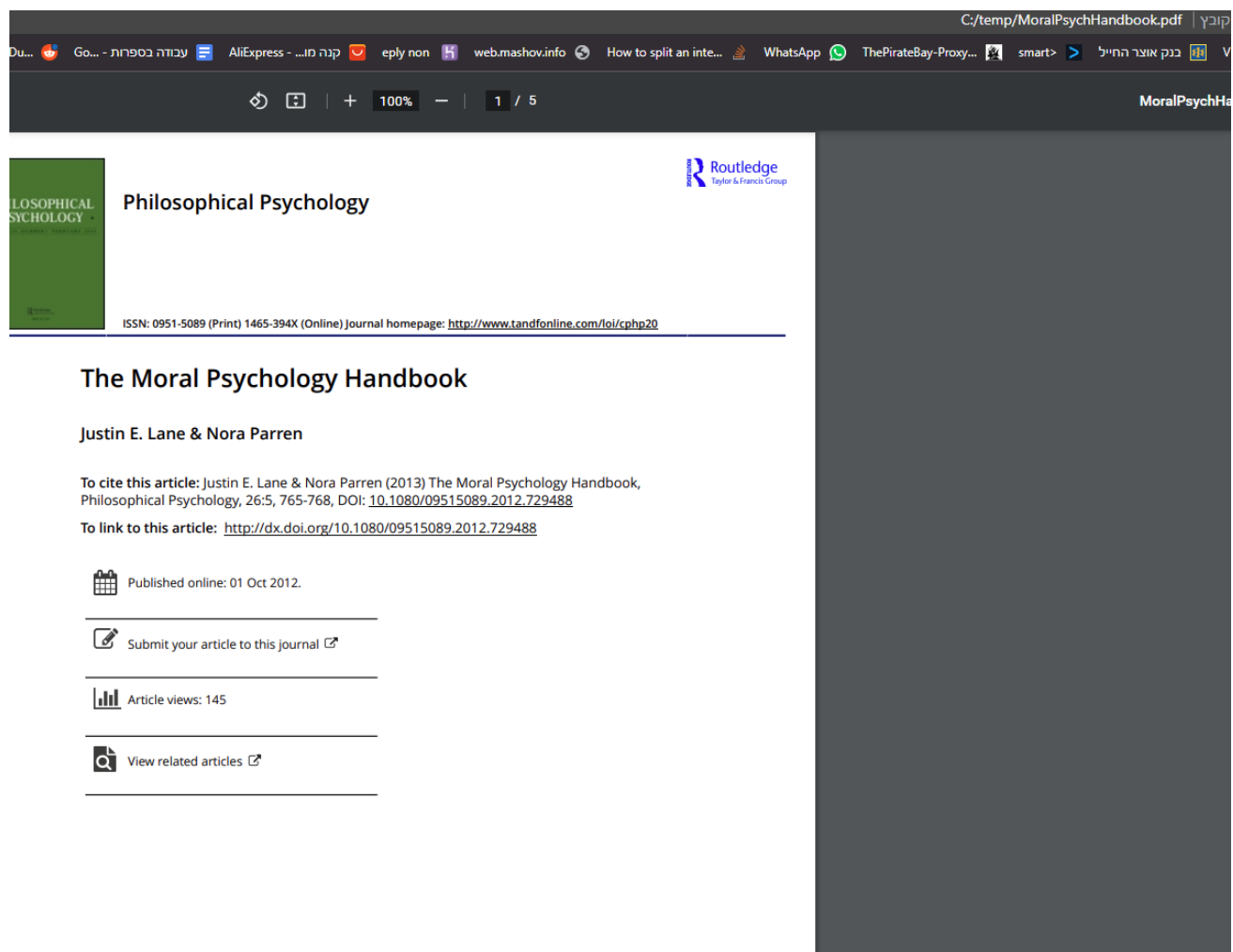
```
Microsoft Visual Studio Debug Console
Checking tracker: tracker.coppersurfer.tk
Checking tracker: tracker.opentracker.org
failed to connect 87.70.172.75
connected 72.21.17.17
connected 46.232.211.91
Downloaded: 10.86%
Downloaded: 16384 / 150932
Downloaded: 21.71%
Downloaded: 32768 / 150932
Downloaded: 32.57%
Downloaded: 49152 / 150932
Downloaded: 43.42%
Downloaded: 65536 / 150932
Downloaded: 54.28%
Downloaded: 81920 / 150932
Downloaded: 65.13%
Downloaded: 98304 / 150932
Downloaded: 75.99%
Downloaded: 114688 / 150932
Downloaded: 86.84%
Downloaded: 131072 / 150932
Downloaded: 97.7%
Downloaded: 147456 / 150932
Downloaded: 100%
Downloaded: 150932 / 150932
Downloading to disk...
Done: C:\temp\MoralPsychHandbook.pdf

C:\Users\dolev\source\repos\bittorrent\x64\Debug\BitTorrentMain.exe (process 18508) exited with code 0.
Press any key to close this window . . .
```

ניתן לראות כאן מספר דברים:

- ראשית ניתן לראות התוכנה בודקת את ה-tracker הדיפולטיבי (tracker.coppersurfer.tk) אך היא לא מצליחה להתחבר אליו (כנראה timeout), היא עוברת ל-backup trackers, ומוצאת tracker פעיל.
- שנית התוכנה מקבלת מה-tracker רשימה של peers ומתחילה לנסות להתחבר לכל אחד מהם.
- את הפיר הראשון היא לא מצליחה להתחבר אליו (כנראה לא פעיל או אין בידו את הקובץ).
- לאחר מכן, היא הצליחה למצוא שני peers אשר עובדים והיא מתחילה להוריד חתיכות מכל אחד מהם.
- לבסוף, לאחר ההורדה השלמה של הקובץ, התוכנה מורידה את כל הקובץ ל-disk של המחשב וכותבת באיזה מיקום הוא.

4. **סיום ההורדה:** לאחר זמן מסוים, התוכנה תסיים את ההורדה והקובץ יקבצים ישמרו במקום המתאים ויהיה ניתן לצפות בתוכן המורד. למשל אנחנו כעת יכולים לראות את הקובץ pdf שהורדנו. להלן הקובץ בכרום:



C:/temp/MoralPsychHandbook.pdf | קובץ

Go... AliExpress - ... קנה מ... eply non web.mashov.info How to split an inte... WhatsApp ThePirateBay-Proxy... smart> בנק אוצר החייל V

100% 1 / 5 MoralPsychHa

**Philosophical Psychology**

ISSN: 0951-5089 (Print) 1465-394X (Online) Journal homepage: <http://www.tandfonline.com/loi/cphp20>

**The Moral Psychology Handbook**

Justin E. Lane & Nora Parren

To cite this article: Justin E. Lane & Nora Parren (2013) The Moral Psychology Handbook, Philosophical Psychology, 26:5, 765-768, DOI: [10.1080/09515089.2012.729488](https://doi.org/10.1080/09515089.2012.729488)

To link to this article: <http://dx.doi.org/10.1080/09515089.2012.729488>

Published online: 01 Oct 2012.

Submit your article to this journal

Article views: 145

View related articles

## רפלקציה - בעיות ותאגרים בתהליך המימוש והמחקר של הלקוח

במהלך עבודתי על עבודת הגמר נתקלתי בהרבה אתגרים, הן תכנותיים והן מחשבתיים אשר הקדשתי להן מחשבה רבה. בחלק זה של הפרויקט אתאר את האתגרים שהיו לי במהלך התהליך וכיצד פתרתי אותן.

**אתגר 1:** בהתחלה כאשר ניגשתי לפרויקט, קראתי על המושגים והתהליכים שרציתי לדעת – רשתות, BitTorrent ו-cpp מהאתרים הרשמיים. נוכחתי לדעת שההסברים אינם נוחים לקריאה ואינם מובנים. למרות קריאה חוזרת ונישנית לא הבנתי מאיפה להתחיל ולא ידעתי מה אני אמור לעשות, מתי וכיצד.

**פתרון:** לאחר זמן מה של ניסיון קריאה והבנה מן ההסברים הרשמיים, עברתי להסברים הלא רשמיים ולמידה אינטראקטיבית (סרטונים ביוטיוב) ונוכחתי לדעת כי אני מבין יותר כיצד כל התהליך קורה. בנוסף, הורדתי את הכלים הנדרשים בעצמי והתנסיתי בעצמי בתפעולם.

**אתגר 2:** כאשר לראשונה התחלתי לכתוב בשפת cpp, כתבתי פונקציית התחברות ל-tracker אך היא לא עבדה כלל והחזירה לי שגיאה. עברתי מספר פעמים על מה שאני שולח ואיך אני שולח, אך הפעם רק בעזרת הדפסות המשתנים ודיבוג של התוכנה. אמנעם כל הבקשות היו על פי הצפוי, בדומה למה שכתוב בהסברים על הפרוטוקול, אך ידעתי שאם הן היו באמת אותו דבר, הפונקציה היית עובדת ויש בעיה.

**פתרון:** הרצתי את התוכנה והסנפתי אותה בעזרת Wireshark. הבנתי כי גוף הבקשה ששלחתי בבקשתה התחברות אינו בדיוק אותו דבר כמו ש-uTorrent שולח אותו לשרת (לפני כן, פתחתי uTorrent עם אותו קובץ טורנט שאני משתמש בו ושמרתי כל בקשה שקורת בכדי שאוכל להשוות בין הלקוח שלי לבין הלקוח המוכר uTorrent). כשהגדרתי את הבקשה לשרת ב-socket עשיתי עם מחזורת והוספתי אליה את המידע הנדרש. הבנתי על פי ההשוואה שעשיתי עם uTorrent שבטעות שלחתי גם את הגרשיים ולכן tracker לא הבין את הבקשות שלי ולא החזיר תשובה ראויה. מיד תיקנתי והכנתי מחלקה אשר כותבת לתוך וקטור מה שצריך, ולאחר מכן, הכל עבד כצפוי.

389	58.689544	192.168.1.103	72.21.17.17	BitTorrent	122 Handshake
393	58.760784	192.168.1.103	46.232.211.91	BitTorrent	122 Handshake
396	58.824669	72.21.17.17	192.168.1.103	BitTorrent	127 Handshake
397	58.824944	192.168.1.103	72.21.17.17	BitTorrent	59 Interested
399	58.830815	46.232.211.91	192.168.1.103	BitTorrent	127 Handshake
400	58.831047	192.168.1.103	46.232.211.91	BitTorrent	59 Interested
405	58.901073	46.232.211.91	192.168.1.103	BitTorrent	56 Bitfield, Len:0x2
407	58.957302	72.21.17.17	192.168.1.103	BitTorrent	56 Bitfield, Len:0x2
409	59.014892	46.232.211.91	192.168.1.103	BitTorrent	59 Unchoke
410	59.015134	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x0,Begin:0x0,Len:0x4000)
430	59.139898	72.21.17.17	192.168.1.103	BitTorrent	59 Unchoke
431	59.140255	192.168.1.103	72.21.17.17	BitTorrent	71 Request, Piece (Idx:0x1,Begin:0x0,Len:0x4000)
432	59.165650	46.232.211.91	192.168.1.103	BitTorrent	118 Piece, Idx:0x0,Begin:0x0,Len:0x4000
433	59.191483	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x2,Begin:0x0,Len:0x4000)
469	59.330847	46.232.211.91	192.168.1.103	BitTorrent	118 Piece, Idx:0x2,Begin:0x0,Len:0x4000
470	59.336998	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x3,Begin:0x0,Len:0x4000)
485	59.419954	72.21.17.17	192.168.1.103	BitTorrent	118 Piece, Idx:0x1,Begin:0x0,Len:0x4000
492	59.428349	192.168.1.103	72.21.17.17	BitTorrent	71 Request, Piece (Idx:0x4,Begin:0x0,Len:0x4000)
500	59.476547	46.232.211.91	192.168.1.103	BitTorrent	118 Piece, Idx:0x3,Begin:0x0,Len:0x4000
501	59.485122	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x5,Begin:0x0,Len:0x4000)
540	59.624709	46.232.211.91	192.168.1.103	BitTorrent	118 Piece, Idx:0x5,Begin:0x0,Len:0x4000
541	59.636759	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x6,Begin:0x0,Len:0x4000)
546	59.693963	72.21.17.17	192.168.1.103	BitTorrent	118 Piece, Idx:0x4,Begin:0x0,Len:0x4000
547	59.699782	192.168.1.103	72.21.17.17	BitTorrent	71 Request, Piece (Idx:0x7,Begin:0x0,Len:0x4000)
568	59.789567	46.232.211.91	192.168.1.103	BitTorrent	118 Piece, Idx:0x6,Begin:0x0,Len:0x4000
569	59.797940	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x8,Begin:0x0,Len:0x4000)
608	59.950774	46.232.211.91	192.168.1.103	BitTorrent	118 Piece, Idx:0x8,Begin:0x0,Len:0x4000
609	59.962493	192.168.1.103	46.232.211.91	BitTorrent	71 Request, Piece (Idx:0x9,Begin:0x0,Len:0xd94)
610	59.964955	72.21.17.17	192.168.1.103	BitTorrent	118 Piece, Idx:0x7,Begin:0x0,Len:0x4000
616	60.101655	46.232.211.91	192.168.1.103	BitTorrent	810 Piece, Idx:0x9,Begin:0x0,Len:0xd94

```

> Frame 410: 71 bytes on wire (568 bits), 71 bytes captured (568 bits)
> Ethernet II, Src: IntelCor_14:7e:c2 (f8:ac:65:14:7e:c2), Dst: Techni
> Internet Protocol Version 4, Src: 192.168.1.103, Dst: 46.232.211.91
> Transmission Control Protocol, Src Port: 61604, Dst Port: 63493, Seq
▼ BitTorrent
  ▼ Message: Len:13, Request, Piece (Idx:0x0,Begin:0x0,Len:0x4000)
    Message Length: 13
    Message Type: Request (6)
    Piece index: 0x00000000
    Begin offset of piece: 0x00000000
    Piece Length: 0x00004000

```

**אתגר 3:** פירסור קובץ הטורנט היה חלק מאוד מאתגר בעבודה. אני מקבל מבנה מאוד מסובך, של מילונים בתוך רשימות בתוך מילונים. היה נדרש אלגוריתם אשר יודע להסתדר עם קבצי טורנט ולמפה אותם למבנה נתונים מוכר שבניתי.

**פתרון:** חשבתי על שלב שלב באלגוריתם, חשבתי עליו לעומק וחשבתי כיצד ידע להתמודד עם כל סוגי הקבצים שיקבל. בסופו של דבר לאחר חשיבה מרובה והמון טיפולים בשגיאות ספציפיות וטבלאות מעקב הגעתי לאלגוריתם שיוצר סוג של עץ רקורסיבי ומכניס את כל המידע של הקבצי טורנט למבנה נתונים מוכר. הסבר מורחב על האלגוריתם יכול להימצא בעמודים שלמטה.

**אתגר 4:** ישנם מספר דרכים למימוש לקוח של טורנט והיית לי התלבטות: האם לשמור את המידע שאני מקבל ב-RAM, ורק אחראי שאני מאמת את כל המידע להכניסו לדיסק הקשיח. או לחלופין לשמור כל פיסת מידע קטנה שאני מבקש בדיסק ואם אחת מזוהמת אז למחוק הכל. לא ידעתי באיזה אלגוריתם כאדי לבחור, ולכל אחד מהם יתרונות וחסרונות משלו.

**פתרון:** הייתי צריך לבחור מבין הרבה אפשרויות קיימות לאלגוריתמים למימוש ביטורנט. בסופו של דבר החלטתי לפתוח קוד פתוח אל קליינט עובד, ולחקור כיצד או פועל ולחקות את הפעולה שלו. בסופו של דבר, מהסתכלות על קוד של קליינטים אחרים הבנתי שהדרך הכי טובה לעשות את זה היא לשמור את המידע ב-RAM ורק אז להעביר לדיסק (כמובן שלא העתקתי את הקוד מהקליינט המפורסם הקיים אלא כתבתי הכל בעצמי).

**בעיה 1:** כשאתה מוריד קובץ אתה מוריד אותו מאנשים ולא משרת בטוח שאתה יודע שאפשר לסמוך עליו. אתה לא יודע מה תוכן הקובץ, יכולים להעביר לך וירוסים ואתה לא תדע. באופן זה המחשב של המשתמש יכול להזדהם. כדי להתגבר על בעיה חשובה כזאת חשוב תמיד לבדוק כל חתיכה (עם sha1) ולא להוריד לדיסק חתיכות שאינם נבדקו.

**בעיה 2:** כל הקליינטים שאתה מחובר אליהם ב"Swarm" חשופים לכתובת ה IP שלך. הדבר יכול להוות סכנה במקרים מסויימים, ישנה חשיפה ברשת.

לסיכום, במשך שנה עבדתי על מחקר ומימוש לקוח של BitTorrent, נתקלתי בעשרות אתגרים (לא את כולם כתבתי) קטנים וגדולים, הייתי עף סף ייאוש מספר פעמים אך תמיד הצלחתי לקחת צעד קדימה ולהתקדם. זו היא הפעם הראשונה שאני עושה פרויקט גדול המשלב הן מחקר והן פיתוח של תוכנה ובעת המחקר הבנתי שאני נפתח כעת לתחום ענק של אפשרויות ומקורות מידע אשר לא ידעתי על קיומם לפני תחילת הפרויקט.

בהמשך אני ממשיך להשתמש בכל הכלים אשר למדתי – wireshark, cpp ועוד כלים שימושיים. אני שמח על ההזדמנות הגדולה אשר ניתנה לי ועל איך שניצלתי אותה.



## מדריך למשתמש

### תיאור הקבצים בקוד

אתאר כעת את כל הקבצים והמחלקות שאני משתמש בהם בקוד ואסביר בקצרה למה הם משמשים.

### קבצים הקשורים לbencode

- Bencode.cpp – מחלקה אשר מפרסרת סטרינג אשר מקודד בbencode. היא תומכת בencode וdecode.
- benCodeObject.cpp – המחלקה המוחזרת לאחר הפרסור של bencode. היא יכולה להכיל 4 טיפוסים: מחרוזת, מספר, רשימה של benCodeObject ומילון של benCodeObject.string.

### קבצים הקשורים לשירותים שרצה לעשות (utilities)

- FilePathUtils.cpp – יש למחלקה זו 4 פונקציות שימושיות שנוכל להשתמש בהם לעזרה של הורדת הקבצים במיקום מסוים, למצוא את מיקום הבסיסי של קובץ ועוד. היא משתמשת בכל מיני פעולות על מחרוזות (לדוגמא find\_last\_of) בכדי לפרסר נתיב מסוים.
- HexUtils.cpp – כאשר אנחנו רוצים לשלוח בקשה עם פרוטוקול BitTorrent נצטרך לשלוח הודעות הכתובת בצורה שאי אפשר להשתמש במחרוזת, נשתמש בbuffer. פונקציה זו תעזור לנו להכניס משנים לbuffer מסוים וגם תעזור לנו להעביר בין טיפוסים, לדוגמא: להעביר בין buffer לuint16\_t.
- UrlUtils.cpp – כתובת של tracker כתובה בפורמט מסוים, ונרצה להפריד אותה ל: פורט, סוג פרוטוקול וקישור, נעזר במחלקה הזאת כדי לעשות זאת.
- CryptoGen.cpp – הוא מכין מספר בתים רנדומליים.
- Defs.h – הגדרת טיפוסים בסיסיים (כגון Buffer) והבאת ספריות בסיסיות (כגון Windows.h).
- Directory.cpp – מקבל נתיב ובונה תיקייה חדשה במערכת הקבצים. ניתן להוסיף לה קבצים בעזרת הפונקציה addFile של מחלקה זו.
- Exception.cpp – מחלקה שתעזור לנו לזרוק שגיאות בצורה נכונה.
- Sha1.cpp – ביצוע sha1.

### קבצים הקשורים לWinApi

- Handle.cpp – פונקציית בסיס לשימוש של HADNLE בוינאוס. היא בעצם תהיה פונקציית מעטפת לHANDLE. הפונקציה תקבל HADNE והיא תשמור אותו רק אצלה במשנה private. ברגע שמחלקה זו תיצא מן scope, המחלקה תקרא destructor שלה לCloseHandle. חשוב מאוד להכין פונקציה כזאת משום שבמקרה ושכחנו לקרוא לCloseHandle יהיה לנו דליפת משאבים חמורה מהתוכנה שלנו ואולי יהיה אפשר לנצל אותה לרעה.

- File.cpp – הפונקציה אחראית על קובץ אחד. אפשר בconstructor ליצור קובץ חדש או להשתמש באחד קיים. הפונקציה תחזיק מופע של Handle של File מסוים. היא תתמוך ב: קריאה, כתיבה, קריאה של שורות מסוים. בנוסף לכך למחלקה זו יש גם כמה פקודות סטטיות חשובות כגון: שינוי שם קובץ, בדיקה אם קובץ קיים, מחיקת קובץ, להשוות קבצים.
- WinShok.cpp – מאתחלת את הפקודה WSStartup אשר מאפשרת לנו להשתמש בsockets.
- destructor היא תקרא WSACleanup, זו היא בעצם פונקציית מעטפת לWSA.
- Socket.cpp – יצירת חיבור של סוקט ושליחת הודעות. היא תשתמש בWinshok.cpp.

### קבצים אשר קשורים לפרוטוקול BitTorrent

- Messages.cpp – כפי שראינו יש אפשרות לשלוח מספר רב של הודעות בפרוטוקול, לכן החלטתי להכין חלקה אשר מכינה אותם. היא תומכת בהכנת 11 סוגים של הודעות שונות. לכל הודעה היא תקבל את הפרמטרים הרלוונטיים.
- torrentParser – המחלקה מכילה מספר פקודות סטטיות שכל אחת מהם מקבלת את קובץ הטורנט (כבר מפורסר) ומחלצת ממנו משהו. לדוגמא: אם נרצה לדעת כמה בתים כל חתיכה, נקרא לפקודה pieceLength וניתן לה את הטורנט, היא תחזיר לי מספר בגודל size\_t.
- Tracker.cpp – יהיה אחראי על התקשורת עם tracker, ייצור איתו חיבור ראשוני ויבקש את רשימת הפירים.
- Peer.cpp – יהיה אחראי על התקשורת עם peer, ייצור איתו חיבור ראשוני, יתחיל לבקש חתיכות ממחלקת pieces ויוריד אותם מהפיר אליו הוא מחובר.
- Pieces.cpp – יש צורך במחלקה אשר אומרת לכל פיר איזה חתיכה להוריד. מחלקה זו תצטרך לעבוד עם מנעולים מסוג mutex משום שלא יוצר race conditions עם הthreadים השונים המנסים לגשת לאותה פונקציה. כאשר מסתיים ביקוש החתיכות, המחלקה הזאת גם אחראית להוריד את הקובץ אל מערכת הקבצים של המחשב בשימוש באחד מהספריות של utils.
- Downloader.cpp – משלב את כל המחלקות אשר תיארו. יוצר חיבור עם tracker בעזרת המחלקה, מבקש פירים, עובר עליהם ויוצר עם כל אחד מהם חיבור בעזרת מחלקת peer, כל פיר שהוא ייצור איתו חיבור הוא ישלח אותו לthread משלו והקובץ יתחיל להיות מורד.

### קובץ ראשי

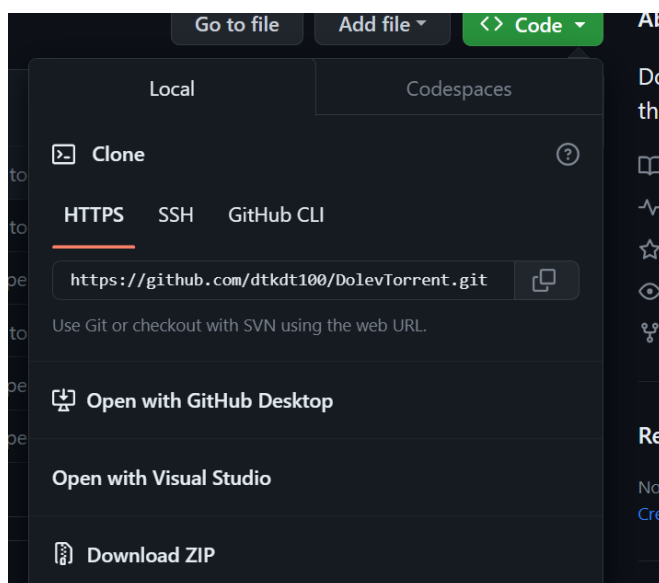
- Main.cpp – נגדיר שם את הנתבי של קובץ הטורנט ונקרא למחלקת downloader.

## קבצי הבדיקות

- BenCodeTests.cpp – קובץ בדיקות על פרסור קובץ torrent. הבדיקות יהיו בעזרת סיפריית gtest.
- UrlUtils.cpp – קובץ בדיקות על פרסור מחרוזת URL. הבדיקות יהיו בעזרת סיפריית gtest.

## איך להפעיל את המערכת שלי מ0

1. יש להוריד את הקוד של הפרויקט מקוד פתוח הנמצא בgithub שלי :  
<https://github.com/dtkdt100/DolevTorrent>. נוריד את התוכנה בעזרת השימוש בפקודה git clone או הורדת קובץ zip מגיטהאב וחילוץ.



2. כפי שניתן לראות יש לי כמה קבצי טורנט לדוגמא, ניתן להשתמש בהם. אם תרצו להוריד קובץ טורנט בנפרד ותרצו להשתמש בו תוכלו להוריד מהאתר : <https://www.pirateproxy-bay.com>
3. לאחר שהורדנו את הקבצים הנדרשים, נצטרך להריץ את הקוד cpp. יש עדיפות להשתמש ב visual studio 2022, זו היא התוכנה שאני השתמשתי אך כל תוכנה אשר מריצה קוד cpp תעבוד.
4. גררו את קובץ הטורנט לתוכנה, כעת יש לנו את הנתבי שלה. ניתן להריץ את התוכנה ולראות את logins אשר התוכנה מדפיסה. במידה ונמצא פירים שיתנו לנו שירות, התוכנה תתחיל להוריד את הקובץ וכאשר שהורדת הקובץ תסתיים, ניתן למצוא אותה בקובץ הורדה שנחבר.

## שלב ראשון – פרסור קובץ torrent

בשלב הראשון של הפרויקט יש לבצע פרסור ראשוני לקובץ הטורנט שאנחנו מקבלים מהשתמש. האלגוריתם עובד בצורה רקורסיבית והוא מחזיר מחלקה הנקראת `BenCodeObject`. משום של `bencode` יש רק 4 טיפוסים (מחרוזת, מספר, רשימה או מילון), גם ל `BenCodeObject` יהיו ארבעה constructors בהתאם לכל אחד מהטיפוסים. שניים מהטיפוסים, רשימה ומילון, הינם טיפוסים מורכבים והם בעצם יהיו `templates` של `BenCodeObject` אחר. זאת אומרת שיכול להיות לנו `BenCodeObject` שהוא רשימה, הוא יהיה רשימה של `BenCodeObject` אחרים, וכל אחד מהם הוא מספר.

```
using BenCodeInteger = int64_t;
using BenCodeString = std::string;
template <typename T> using BenCodeList = std::vector<T>;
template <typename T> using BenCodeDictionary = std::map<std::string, T>;
```

```
BenCodeObject::BenCodeObject(BenCodeInteger v) {
    value.benCodeInteger = v;
    bType = BenCodeType::BenCode_Integer;
}

BenCodeObject::BenCodeObject(BenCodeString&& v) {
    value.benCodeString = new BenCodeString(std::move(v));
    bType = BenCodeType::BenCode_String;
}

BenCodeObject::BenCodeObject(BenCodeList<BenCodeObject>&& v) {
    value.benCodeList = new BenCodeList<BenCodeObject>(std::move(v));
    bType = BenCodeType::BenCode_List;
}

BenCodeObject::BenCodeObject(BenCodeDictionary<BenCodeObject>&& v) {
    value.benCodeDictionary = new BenCodeDictionary<BenCodeObject>(std::move(v));
    bType = BenCodeType::BenCode_Dictionary;
}
```

יצרתי מחלקת `bencode` והיא תומכת גם ב `decode` וגם ב `encode`.

```
class bencode {
public:
    static BenCodeObject decode(std::string& decodedStr);
    static std::string encode(BenCodeObject obj);

private:
    static BenCodeObject decode(std::string& decodedStr, int* i);
    static BenCodeInteger decodeInteger(std::string& decodedStr, int* i);
    static BenCodeString decodeString(std::string& decodedStr, int* i);
    static BenCodeList<BenCodeObject> decodeList(std::string& decodedStr, int* i);
    static BenCodeDictionary<BenCodeObject> decodeDictionary(std::string& decodedStr, int* i);
    static BenCodeInteger getInBenCodeInteger(std::string& decodedStr, int* i, char ending = 'e');

    static std::string encodeInteger(BenCodeInteger obj);
    static std::string encodeString(BenCodeString obj);
    static std::string encodeList(BenCodeList<BenCodeObject> obj);
    static std::string encodeDictionary(BenCodeDictionary<BenCodeObject> obj);
};
```

נסביר כעת על decode, בוא אנחנו נשתמש כדי לפרסר את קובץ torrent. ראשית אנחנו נבדוק באיזה תו מתחיל הסטרינג שקיבלו, ובאיזה index של המחרוזת אנחנו מסכלים. כפי הסברתי מקודם, יש ארבעה טיפוסים וכל אחד מתחיל עם אות שונה.

```
BenCodeObject bencode::decode(std::string& decodedStr, int* i) {
    if (decodedStr[*i] == 'i') {
        return BenCodeObject(decodeInteger(decodedStr, i));
    } else if (decodedStr[*i] == 'l') {
        return BenCodeObject(decodeList(decodedStr, i));
    } else if (decodedStr[*i] == 'd') {
        return BenCodeObject(decodeDictionary(decodedStr, i));
    } else {
        return BenCodeObject(decodeString(decodedStr, i));
    }
}
```

נראה פונקציה לדוגמא של פרסור אינטגר:

```
BenCodeInteger bencode::decodeInteger(std::string& decodedStr, int* i) {
    *i += 1;
    BenCodeInteger retValue = getInBenCodeInteger(decodedStr, i);
    return BenCodeInteger(retValue);
}

BenCodeInteger bencode::getInBenCodeInteger(std::string& decodedStr, int* i, char ending) {
    int retValue = 0;
    int sign = 1;

    if (decodedStr[*i] == '-') {
        sign = -1;
        *i += 1;
    }

    while (decodedStr[*i] != ending) {
        retValue = retValue * 10 + (decodedStr[*i] - '0');
        *i += 1;
        if (*i >= decodedStr.size()) {
            throw Exception(INVALID_BENCODE_STRING);
        }
    }
    *i += 1;
    return retValue * sign;
    return BenCodeInteger(retValue);
}
```

לאחר הפרסור ושמירת המידע הנחרש נעבוד לשלב השני – התחלת תהליך התקשורת עם Tracker.

## שלב שני – התחברות לTracker וקבלת רשימת הפירים

בכדי לתקשר עם tracker, אנחנו נצטרך ליצור חיבור מבוסס socket.

נעשה זאת על ידי יצירת מחלקת socket בסיס אשר מאפשרת את הפונקציות הבאות: יצירת חיבור של tcp/udp, שליחת הודעה, קבלת הודעה, קבלת הודעה מלאה (לפעמים ההודעות שהשרת מחזיר הן ארוכות מידי בשביל פקטה אחת, אז נשלח בתחילת הפקטה הראשונה אז גודל החתיכה ונעשה while עד שהגיע כל החבילה) וסגירת החיבור.

```
class Socket {
public:
    Socket(int sType, int sProtocol);
    ~Socket();
    void connectSocket(std::string& hostname, int port);
    void sendMessage(std::string& message);
    std::string receiveMessage(int bytesToReceive);
    std::string onWholeMessage(bool handshake);
    void closeSocket();
private:
    WinShok winShok;
    SOCKET sock;

    int sType;
    int sProtocol;
};
```

מחלקת tracker תיצור מופע של ספריית socket ותתחבר אליו. ברגע שנוצר חיבור ואין שגיאה של timeout, נכל להתחיל בבקשת ההתחברות לtracker וקבלת הפירים.

נבנה בהתחלה את בקשת ההתחברות על פי ההוראות בפרוטוקול BitTorrent ונשלח אותה. נפרסר את התשובה ואם הכל בסדר נבקש את רשימת הפירים.

```
std::vector<PeerInfo> Tracker::getPeers(BenCodeObject& torrent) {
    std::string connReq = buildConnReq();
    sock.sendMessage(& message: connReq);
    std::string response = sock.receiveMessage(bytesToReceive: 1024);
    analyzeConnRespose(& response);
    std::string annoReq = buildAnnounceReq(& torrent);
    sock.sendMessage(& message: annoReq);
    response = sock.receiveMessage(bytesToReceive: 1024);
    return analyzeAnnounceRespose(& response);
}
```

להלן דוגמא של בניית בקשת announce:

```

std::string Tracker::buildAnnounceReq(BencodeObject& torrent) {
    Buffer buf;
    // connection id
    HexUtils::writeUInt64BE(&buf, connectionId);
    // action
    HexUtils::writeUInt32BE(&buf, 1);
    // transaction id
    transactionId = CryptoGen::randomBytes(4);
    HexUtils::writeUInt32BE(&buf, transactionId);
    // info hash
    std::string info = TorrentParser::infoHash(torrent);
    infoHash = HexUtils::hexStringToBufferHex(info);
    HexUtils::writeBytes(&buf, infoHash);
    // peer id
    if (peerId.size() == 0) {
        generatePeerId();
    }
    HexUtils::writeBytes(&buf, peerId);
    // downloaded
    HexUtils::writeUInt64BE(&buf, 0);
    // left
    HexUtils::writeUInt64BE(&buf, TorrentParser::fileSize(torrent));
    // uploaded
    HexUtils::writeUInt64BE(&buf, 0);
    // event
    HexUtils::writeUInt32BE(&buf, 0);
    // ip address
    HexUtils::writeUInt32BE(&buf, 0);
    // key
    HexUtils::writeUInt32BE(&buf, CryptoGen::randomBytes(4));
    // num want
    HexUtils::writeUInt32BE(&buf, -1);
    // port
    HexUtils::writeUInt16BE(&buf, 6881);

    return std::string(buf.begin(), buf.end());
}

```

המחלקה שאני משתמש בה, הנקראת HexUtils, בניתי לבד והיא עוזרת לדחוף מספר בתים מסוים לbuffer. הbuffer הינו template של וקטור של אות.

```
using Buffer = std::vector<unsigned char>;
```

לאחר קבלת הפירים, נעבור לשלב הבא.

## שלב שלישי – הורדת הקובץ מהpeer

כעת יש לנו רשימה של פירים, כל פיר מורכב מ:

- IP – IP של הפיר
- Port – postn שצריך לתקשר עם הפיר (נתקשר איתו תמיד בTCP לא משנה מהport)
- Id – מזהה ייחודי של אותו פיר

כעת ניגש לפונקציה אשר מורידה את הקובץ מרשימה של פירים, היא שולחת כל peer נפרד לthread. זו היא הפונקציה הכי חשובה בקוד לכן אסביר עליה ביסודיות.

### פונקציית update

לשיטת עבודה כזאת (עבודה עם thread) יש בעיה אחת מרכזית: איך אנחנו מתקשרים בין הthreadים, הרי יש לנו מספר peer שונים שיצרנו איתם חיבור, אנחנו לא רוצים שכל אחד מהם יורד אותה חתיכה כמו האחר, כי מה עשינו בכך?

מחלקת הpieces באה לפתור את בעיה זאת. הרעיון הוא שנאתחל מופע של מחלקה זו במחלקת בסיס (downlaoder) ושכל thread יקבל את הכתובת בזיכרון (מופע) של המחלקה ויעבוד עמה. המחלקה תגיד לו איזה חתיכה צריך לקבל, הוא יוריד אותה, ואז ישלח חזרה לpieces והיא תשמור את החתיכה אצלה.

### מחלקת pieces

מחלקת הpieces תאוחל עם קובץ הtorrent המפורסר בכדי שנדע כמה חתיכות יש להוריד וכמה בקשות נצטרך לבקש מהפירים.

נכין שני וקטורים:

1. וקטור של וקטור של buffer – שומר את כל החתיכות שהתקבלו על ידי הפירים.
2. וקטור של וקטור של bool – שומר איזה חתיכות בוקשו כבר ואיזה לא בכדי שנדע לא לבקש את אותה חתיכה פעמיים. ברגע שפיר מסוים יבקש חתיכה, אותה חתיכה תשתנה לtrue ואם תהיה שגיאה בבקשת החתיכה אז היא תשתנה לfalse ונבקש אותה מפיר אחר.

### פונקציית pieceNedded

הפונקציה מחזירה PieceInfo המכיל:

- Index – מספר החתיכה שכרגע מבוקשת



- **Begin** – חתיכה לפעמים יכולה להיות גדולה יותר מהגודל שנהוג לבקש לכל בקשה ( $16384 = 2^{14}$ ), לכן כל חתיכה מפוצלת לblocks ואם גודל החתיכה גדולה מהגודל של הבלוק, מבקשים כל בלוק בנפרד. begin הוא ההתחלה של החתיכה המבוקשת.
- **Length** – גודל של הבלוק המבוקש, הוא תמיד הולך להיות 16384

הפונקציה עטופה מנעול מסוג mutex. משתמשים באובייקט mutex כדי להגן על משאב משותף מפני גישה בו-זמנית על ידי שרשרים או תהליכים מרובים. כל שרשר חייב להמתין לבעלות על ה-mutex לפני שיוכל להפעיל את הקוד הניגש למשאב המשותף. לדוגמה, אם מספר שרשרים חולקים גישה למסד נתונים, השרשרים יכולים להשתמש באובייקט mutex כדי לאפשר רק לשרשר אחד בכל פעם לכתוב למסד הנתונים.

אנחנו עוטפים את הפונקציה במנעול משום שהרבה threads במקביל ירצו לבקש חתיכות ואנחנו לא רוצים ששני threads יבקשו את אותה חתיכה. ברגע שהפונקציה מוצאת חתיכה שצריך לשלוח, או לא מוצאת חתיכה והקובץ כבר סיים לרדת, היא תשחרר את המנעול והוא יהיה פתוח לthreads אחרים לקרוא לפונקציה הזו.

```
PieceInfo Pieces::pieceNeeded() {
    DWORD waitResult = WaitForSingleObject(
        mutexHandle.getRawHandle(),
        INFINITE);
    if (waitResult == WAIT_OBJECT_0) {
        for (int i = 0; i < pieces.size(); i++) {
            for (int j = 0; j < pieces[i].size(); j++) {
                if (pieces[i][j].size() == 0 && !requested[i][j]) {
                    requested[i][j] = true;
                    ReleaseMutex(mutexHandle.getRawHandle());
                    return PieceInfo(i, j * BLOCKSIZE, TorrentParser::blockLen(torrent, i, j));
                }
            }
        }
    }

    ReleaseMutex(mutexHandle.getRawHandle());
    return PieceInfo(0, 0, 0);
}
```

### פונקציית downloadFilesToDisk

לאחר שההורדה של הקובץ מהפירים השונים הסתיימה, הפונקציה downloadFilesToDisk נקראת והקובץ בעצם עובר מהRAM לdisk הקשיח.

ראשית, נעביר את כל החתיכות לbuffer אחד על ידי לולאה פשוטה:

```
Buffer allPieces;

for (int i = 0; i < pieces.size(); i++) {
    for (int j = 0; j < pieces[i].size(); j++) {
        for (int k = 0; k < pieces[i][j].size(); k++) {
            allPieces.push_back(pieces[i][j][k]);
        }
    }
}
```

לאחר מכן נבדוק האם הקובץ המורד הוא רשימה של קבצים או קובץ יחיד ונוריד ונפתח תיקייה בהתאם.

כעת לאחר שהסברנו על מחלקת pieces ויצרנו מופע שלה במחלקת Downloader, נצטרך לשלוח את הכתובת שלה בזיכרון לכל אחד מהthreads.

נעבור על הרשימה של הפירים עם לולאה, ועל כל אחד מהפירים ננסה ליצור חיבור. אם הצלחנו ליצור חיבור נשלח אותו לthread משלו, במקרה ולא, נעבור הלאה להמשך הפירים.

## מחלקת peer

כאשר אנחנו ננסה ליצור מופע של מחלקה זו, נצטרך להעביר peerInfo, שזה בעצם המידע שנצטרך על כל פיר כדי ליצור איתו חיבור (הוסבר בעמ' 34).

ניצור socket עם חיבור tcp, ואם peer מתחבר אלינו, הכל בסדר ואנחנו שולחים את הpeer הזה לthread בנפרד.

```
Peer::Peer(PeerInfo info): sock(SOCK_STREAM, IPPROTO_TCP, hostname(info.ip), id(info.id) {  
    sock.connectSocket(info.ip, info.port);  
}
```

## פונקציית download

הפונקציה מקבלת פוינטר למחלקת pieces (אותו פוינטר שהגדרנו במחלקת downloader), נשמור אותו במשתנים הלוקלים של המחלקה כי נשתמש בו הרבה בהמשך.

נבנה את הודעת "לחיצת היד" (הוסבר עליה בחלק התיאורטי) ונשלח לpeer. נקבל את התשובה ונקרא לפונקציה הנקראת messageHandler

## פונקציית messageHandler

נבדוק אם התשובה שקיבלנו מהבקשה הקודמת היא גם לחיצת יד, אם אין תשובה או שהתשובה אינה לחיצת היד הpeer אינו טוב וננתק את הקשר.

המשיך לעבוד לפי ההוראות ונשלח בקשת interested, ואם התשובה המוחזרת לנו היא unchoke, נתחיל לבקש חתיכות. במקרה והיא התשובה היא choke, ננתק את החיבור מפני שהפיר לא רוצה לתקשר איתנו.

## פונקציית requestPiece

הפונקציה ראשית מבקשת ממחלקת החתיכות איזה חתיכה היא צריכה עכשיו לשלוח. במקרה ואורך החתיכה היא לא 0 (אם היא 0 סיימנו להוריד את הקובץ).

נשלח בקשת חתיכה לפיר שאנחנו מחבורים אליו ונחכה לתשובה. נבדוק שהתשובה היא חתיכה (המזהה של ההודעה צריך להיות המספר 7). במקרה זה כן חתיכה והכל עבר בסדר נשלח למחלקת החתיכות את החתיכה שהתקבלה ונבקש חתיכה חדשה.

## המשך פונקציית update

בתוך הלולאה של הפירים נעשה :

1. נקרא לconstructor של מחלקת peer ונוודא שנוכל ליצור קישור פעיל לפיר. במקרה ולא (נתפוס את שגיאת ההתחברות עם try וcatch) אז נעבור לפיר הבא ברשימה. נגדיר את מופע הפיר כunique\_ptr, אסביר בהמשך למה.
2. ניצור פוינטר בעזרת unique\_ptr של כל הפרמטרים שאנחנו צריכים להעביר לthread :

- פוינטר אל מופע של מחלקת peer
- פוינטר אל מופע של מחלקת pieces
- Infohash – לשליחת בקשת "לחיצת היד" נצטרך את sha1 של info של קובץ torrent.

3. נתחיל thread בעזרת winapi ונשמור אותו בוקטור של HANDLEים של כל הthreads שיצרנו עד עכשיו.

נשמור את כולם בוקטור משום

שכך נוכל להשתמש בפקודה :

WaitForMultipleObjects

ולחכות לכל הthreads שהסתיימו.

ניצור פונקציית start אשר היא

```
DWORD __stdcall Downloader::start(DownloadPeerParams* paramsRaw) {
    try {
        std::unique_ptr<DownloadPeerParams> mergeParams(paramsRaw);
        paramsRaw->peer->download(paramsRaw->pieces, paramsRaw->infoHash);
    }
    catch (...) {
        return 0;
    }
}
```

נקראת מיד בהתחלת הthread. היא מקבלת את הפרמטרים שהנ"ל והיא קוראת לpeer->download.

4. נשתמש בפקודה WaitForMultipleObjects ונחכה על סיום כל הthreads והורדת הקובץ.

```
bool Downloader::update(std::vector<PeerInfo>& peers) {
    std::vector<HANDLE> threads;
    for (auto& peerInfo : peers) {
        try {
            auto peer = std::make_unique<Peer>(peerInfo);
            auto paramsPtr = std::make_unique<DownloadPeerParams>(peer.release(), &pieces, infoHash);
            std::cout << "connected " << peerInfo.ip << " " << std::endl;
            HANDLE thread = CreateThread(NULL, 0,
                reinterpret_cast<LPTHREAD_START_ROUTINE>(start),
                paramsPtr.release(), 0, NULL);
            threads.push_back(thread);
            peersNum += 1;
        }
        catch (Exception e) {
            std::cout << "failed to connect " << peerInfo.ip << " " << std::endl;
        }
    }

    if (threads.size() == 0) {
        return false;
    }

    DWORD waitStatus = WaitForMultipleObjects(threads.size(), threads.data(), true, INFINITE);
    if (waitStatus < WAIT_OBJECT_0 || waitStatus >= WAIT_OBJECT_0 + threads.size()) {
        throw Exception("Error while waiting for threads to finish");
    }

    return true;
}
```

## ספריות אשר השתמשתי בהם בהכנת הלקוח

- Windows.h – הינה ספרייה ספציפית לווינדוס עבור שפות התכנות C, C++, היא הספרייה הראשית לשימוש ב-Win32 API. המחלקה הזאת מייבאת את כל הפונקציות הנדרשות בכדי שנוכל לעבוד עם מערכת הקבצים של המחשב, נוכל לנהל thread ולשים מנעולים בניהם.
- Iostream – נותן לנו את האופציה לעבוד עם קלט ופלט הסטנדרטים, אחת הספריות הבסיסיות יותר ב-C++.
- String – מייבא לנו את מבנה הנתונים של מחרוזת. std::string
- vector – מייבא לנו את מבנה הנתונים של מחרוזת. std::vector
- Random - ספרייה זו מציגה מתקנים ליצירת מספרים אקראיים. std::random\_device הוא מחולל מספרים אקראיים שלמים בחלוקה אחידה המייצר מספרים אקראיים לא דטרמיניסטיים.
- Thread, chrono – עוזר לנו לעשות sleep (שהפונקציה תחכה כמה שניות שנגיד לה).
- Cstdint - כולל את הכותרת של ספריית Cstdint.h <Standard C ומוסיפה את השמות המשויכים למרחב השמות STD. הכללת כותרת זו מבטיחה שהשמות המוצהרים באמצעות קישור חיצוני בכותרת הספרייה Standard C יוכרזו במרחב השמות std.
- Fstream - מגדיר מספר מחלקות התומכות בפעולות iostreams על רצפים המאוחסנים בקבצים חיצוניים.
- Iomanip - כלול את הכותרת הסטנדרטית של iomanip <iostreams כדי להגדיר מספר מניפולטורים שכל אחד מהם לוקח ארגומנט בודד.
- Sstream - מגדיר מספר תבניות מחלקות התומכות בפעולות iostreams ברצפים המאוחסנים באובייקט מערך שהוקצה. רצפים כאלה מומרים בקלות לאובייקטים של תבנית מחלקה basic\_string וממנו.

### שימוש בסוקטים

- Ws2tcpip.h - header זה משמש את Windows Sockets 2 ומאשרת לנו ליצור client של סוקט עם חיבור TCP או UDP או HTTP.
- Ws2\_32.lib - הספרייה ws2\_32.lib היא ספריית ייבוא. היא מורכבת מקטעים קטנים שיפנו ליישום בפועל ב-ws2\_32.dll. ה-DLL ייטען בזמן טעינת התוכנית. זה נקרא קישור דינמי בזמן טעינה.

### טסטים (בדיקות)

- Gtest - GoogleTest הוא מסגרת הבדיקה והmocking של C++ של Google. יעזור לנו בעיקר לעשות בדיקות ל-bencoding.

1. Odom Wendell, and Tom Knott (2006). Networking basics: CCNA 1 companion guide. Indianapolis, IN: Cisco Press, March 22, print.
2. *BitTorrentSpecification - TheoryOrg*. (n.d.).  
<https://wiki.theory.org/BitTorrentSpecification>
3. Olaf van der Spek . (n.d.). *bep\_0015.rst\_post*.  
[http://www.bittorrent.org/beps/bep\\_0015.html](http://www.bittorrent.org/beps/bep_0015.html)
4. Allen. (2016, May 4). *How to make your own bittorrent client*.  
<https://allenkim67.github.io/programming/2016/05/04/how-to-make-your-own-bittorrent-client.html>
5. Schmidt, Douglas C., and Stephen D. Huston. C++ network programming. Boston: Addison-Wesley, 2002. Print.
6. עומר רוזנבוים, שלומי הוד וברק גונן (2020). "רשתות ומחשבים".
7. ברק גונן (2021). "תכנות בשפת פייתון".

## Main.cpp

```
#include "BitTorrent/Downloader.h";

int main() {
    try {
        // The torrent file path
        std::string path = "C:\\temp\\small_file.torrent";
        Downloader d(path);
    }
    catch (Exception e) {
        e.printStackTrace();
    }

    return 0;
}
```

## Bencode.h

```
#pragma once
#include "BenCodeObject.h"

class bencode {
    // Bencode (pronounced like Bee-encode) is the encoding used by the peer-to-peer
    // file sharing system BitTorrent for storing and transmitting loosely structured
    data.
    // This class can decode a string that is decoded with bencode or encode it.
public:
    static BenCodeObject decode(std::string& decodedStr);
    static std::string encode(BenCodeObject obj);

private:
    static BenCodeObject decode(std::string& decodedStr, int* i);
    static BenCodeInteger decodeInteger(std::string& decodedStr, int* i);
    static BenCodeString decodeString(std::string& decodedStr, int* i);
    static BenCodeList<BenCodeObject> decodeList(std::string& decodedStr, int* i);
    static BenCodeDictionary<BenCodeObject> decodeDictionary(std::string& decodedStr,
int* i);
    static BenCodeInteger getInBenCodeInteger(std::string& decodedStr, int* i, char
ending = 'e');

    static std::string encodeInteger(BenCodeInteger obj);
    static std::string encodeString(BenCodeString obj);
    static std::string encodeList(BenCodeList<BenCodeObject> obj);
    static std::string encodeDictionary(BenCodeDictionary<BenCodeObject> obj);

};
```

## BenCodeObject.h

```
#pragma once
#include <map>
#include "Exception.h"

using BenCodeInteger = int64_t;
using BenCodeString = std::string;
template <typename T> using BenCodeList = std::vector<T>;
template <typename T> using BenCodeDictionary = std::map<std::string, T>;

enum BenCodeType {
    BenCode_Integer,
    BenCode_String,
    BenCode_List,
    BenCode_Dictionary
};

class BenCodeObject {
public:
    BenCodeType bType;

    BenCodeObject();
    BenCodeObject(BenCodeInteger v);
    BenCodeObject(BenCodeString&& v);
    BenCodeObject(BenCodeList<BenCodeObject>&& v);
    BenCodeObject(BenCodeDictionary<BenCodeObject>&& v);

    BenCodeInteger getInteger();
    BenCodeString getString();
    BenCodeList<BenCodeObject>* getList();
    BenCodeDictionary<BenCodeObject>* getDictionary();

private:
    union Value {
        BenCodeInteger benCodeInteger;
        BenCodeString* benCodeString;
        BenCodeList<BenCodeObject>* benCodeList;
        BenCodeDictionary<BenCodeObject>* benCodeDictionary;
    } value;
};
```



## Messages.h

```
#pragma once
#include "Exception.h"
#include "HexUtils.h"
#include "BenCodeObject.h"
#include "Pieces.h"

enum MessagesTypes {
    choke = 0,
    unchoke = 1,
    interested = 2,
    not_interested = 3,
    have = 4,
    bitfield = 5,
    request = 6,
    piece = 7,
    cancel = 8,
    port = 9,
    extended = 20
};

class Messages {
public:
    static std::string buildHandShake(Buffer& infoHash, Buffer& peerId);
    static std::string buildKeepAlive();
    static std::string buildChoke();
    static std::string buildUnchoke();
    static std::string buildInterested();
    static std::string buildUnInterested();
    static std::string buildHave(int pieceIndex);
    static std::string buildBitfield(std::vector<bool> bitfield);
    static std::string buildRequest(PieceInfo info);
    static std::string buildPiece(int pieceIndex, int blockIndex, std::string&
block);
    static std::string buildCancel(int pieceIndex, int blockIndex);
    static std::string buildPort(int port);

    static BenCodeDictionary<BenCodeObject> parseResponse(std::string &msg);
};
```

## Peer.h

```
#pragma once
#include "Tracker.h"
#include "Messages.h"
#include "Pieces.h"

class Peer {
public:
    Peer(PeerInfo info);
    void download(Pieces* pieces, Buffer infoHash);
    void sendKeepAlive();
    std::string hostname;
private:

    void messageHandler(std::string& msg);
    bool isHandShake(std::string& msg);
    void chokeHandler();
    void unchokeHandler();
    void haveHandler(BenCodeObject payload);
    void bitfieldHandler(BenCodeObject payload);
    void pieceHandler(BenCodeObject payload);
    void requestPiece();

    Socket sock;
    Pieces* pieces;
    Buffer id;
};
```

## Pieces.h

```
#pragma once
#include "Exception.h"
#include "BenCodeObject.h"
#include "TorrentParser.h"
#include "Directory.h"

const std::string DOWNLOADPATH = "C:\\temp";

struct PieceInfo {
    int index;
    int begin;
    int len;

    PieceInfo(int index, int begin, int len) {
        this->index = index;
        this->begin = begin;
        this->len = len;
    }

    PieceInfo(){}
};

class Pieces {
public:
    Pieces(BenCodeObject& torrent);

    void addPiece(PieceInfo piece, Buffer& block);
    void checkPiece(int index);
    PieceInfo pieceNeeded();
    void failedPiece(PieceInfo info);
    bool onDone();
    void printPerstage();
    void downloadFilesToDisk();

    BenCodeObject& torrent;
private:
    size_t downloaded();

    std::vector<std::vector<Buffer>> pieces;
    std::vector<std::vector<bool>> requested;
```

```
bool startDownloadedToDisk = false;  
Handle createLock() const;  
Handle mutexHandle;  
};
```

## TorrentParser.h

```
#pragma once
#include "Defs.h"
#include "BenCodeObject.h"
#include "bencode.h"
#include "Sha1.h"

const size_t BLOCKSIZE = 16384; // 2^14

class TorrentParser {
public:
    static std::string infoHash(BenCodeObject& torrent);
    static size_t numberPieces(BenCodeObject& torrent);
    static size_t fileSize(BenCodeObject& torrent);
    static size_t pieceLength(BenCodeObject& torrent, int pieceIndex);
    static size_t blocksPerPiece(BenCodeObject& torrent, int pieceIndex);
    static size_t blockLen(BenCodeObject& torrent, int pieceIndex, int blockIndex);
    static BenCodeObject getInfo(BenCodeObject& torrent);
    static std::string infoHash(std::string& torrent);
};
```

## Tracker.h

```
#pragma once
#include "Socket.h"
#include "Exception.h"
#include "Defs.h"
#include "HexUtils.h"
#include "CryptoGen.h"
#include "TorrentParser.h"

struct PeerInfo {
    std::string ip;
    uint16_t port;
    Buffer id;
};

class Tracker {

public:
    Tracker(std::string host, int port);
    std::vector<PeerInfo> getPeers(BenCodeObject& torrent);

    Buffer infoHash;
    Buffer peerId;
private:
    void analyzeConnResponse(std::string& response);
    std::vector<PeerInfo> analyzeAnnounceResponse(std::string& response);
    std::string buildConnReq();
    std::string buildAnnounceReq(BenCodeObject& torrent);
    void generatePeerId();

    Socket sock;
    uint32_t transactionId;
    uint64_t connectionId;
};
```

## File.h

```
#pragma once
#include "Handle.h"
#include "Exception.h"

class File final {
public:
    File(const std::string& path, bool createNew = false);
    Buffer read(uint32_t numberOfBytes, LONG startOffset);
    Buffer readAll();
    LinesBuffer readLines(uint32_t numberOfBytes, LONG startOffset);
    void write(const Buffer& buffer);
    void writeLines(const LinesBuffer& linesBuffer);

    int getSize();
    bool compareFiles(File& other);
    void close();

    static void rename(const std::string& oldPath, const std::string& newPath);
    static bool exists(const std::string& filePath);
    static void deleteFile(const std::string& filePath);
    static bool compareFiles(const std::string& path1, const std::string& path2);

private:
    Handle openFileInternal(const std::string& filePath, bool createNew) const;

    Handle m_file;
};
```

## Handle.h

```
#pragma once
#include <exception>
#include <utility>
#include <Windows.h>

class Handle final {
public:

    Handle(Handle&& other);

    Handle(HANDLE handle);

    Handle() = delete;
    Handle(const Handle&) = delete;
    Handle& operator=(const Handle&) = delete;
    Handle& operator=(Handle&&) = delete;

    void close();

    ~Handle();

    HANDLE getRawHandle();

    bool isValid() const;

private:
    HANDLE m_handle;
};
```



## Socket.h

```
#pragma once
#pragma comment(lib, "Ws2_32.lib")
#include <ws2tcpip.h>
#include <Windows.h>
#include <iostream>
#include "Exception.h"
#include "WinShok.h"

class Socket {
public:
    Socket(int sType, int sProtocol);
    ~Socket();
    void connectSocket(std::string& hostname, int port);
    void sendMessage(std::string& message);
    std::string receiveMessage(int bytesToReceive);
    std::string onWholeMessage(bool handshake);
    void closeSocket();
private:
    WinShok winShok;
    SOCKET sock;

    int sType;
    int sProtocol;
};
```

## WinShok.h

```
#pragma once
#include "Exception.h"

class WinShok {
public:
    WinShok();
    ~WinShok();
};
```

## CryptoGen.h

```
#pragma once
#include <random>
#include "Defs.h"

class CryptoGen {
public:
    static uint32_t randomBytes(size_t bSize);
    static Buffer randomBytesBuffer(size_t bSize);
private:
    static unsigned randomGen();
};
```

## Defs.h

```
#pragma once
#include <Windows.h>
#include <iostream>
#include <string>
#include <vector>

using Buffer = std::vector<unsigned char>;
using LinesBuffer = std::vector<std::string>;
```

## Directory.h

```
#pragma once
#include "File.h"
#include "FilePathUtils.h"

class Directory {
public:
    Directory(std::string& path);
    Directory(std::string&& path);
    File addFile(std::string& fileName);
private:
    void createDir();

    std::string path;
};
```

## Exception.h

```
#pragma once
#include "Defs.h"

using ExceptionType = char const* const;

ExceptionType NO_EXCEPTION = "";

ExceptionType FILE_EXSITS_EXCEPTION = "File already exists";
ExceptionType FILE_NOT_EXSITS_EXCEPTION = "File does not exist";

ExceptionType WRITE_TO_FILE_EXCEPTION = "Error writing to file";
ExceptionType RENAME_FILE_Exception = "Error renaming file";
ExceptionType DELETE_FILE_EXCEPTION = "Error deleting file";
ExceptionType SET_START_FILE_POINTER_EXCEPTION = "Error setting start file pointer";
ExceptionType READ_FROM_FILE_EXCEPTION = "Error reading from file";

ExceptionType INAVILD_HANDLE_EXCEPTION = "Invalid handle";
ExceptionType INVALID_FILE_NAME_EXCEPTION = "Invalid file name";

ExceptionType INVALID_BENCODE_STRING = "Invalid bencode string";

class Exception {
public:
    Exception(ExceptionType eType);
    void printException();
private:
    ExceptionType eType;
};
```

## FilePathUtils.h

```
#pragma once
#include "Exception.h"

class FilePathUtils {
public:
    FilePathUtils() = delete;
    static std::string getFilePath(const std::string& filePath);
    static std::string appendPath(const std::string& path1, const std::string&
path2);
    static std::string getFileName(const std::string& filePath);
    static std::string generateFilePath(const std::string& path, const std::string&
filename, const std::string& ending);
private:
    static bool isEndOfFilePath(const std::string::size_type& pos);
};
```

## HexUtils.h

```
#pragma once
#include "Defs.h"

class HexUtils {
public:
    static void writeUInt8(Buffer* b, uint8_t val);
    static void writeUInt16BE(Buffer* b, uint16_t val);
    static void writeUInt32BE(Buffer* b, uint32_t val);
    static void writeUInt64BE(Buffer* b, uint64_t val);
    static void writeBytes(Buffer* b, Buffer bVal);
    static void writeString(Buffer* b, std::string&& s);

    static uint16_t bufferToUInt16(Buffer& b);
    static uint32_t bufferToUInt32(Buffer& b);
    static uint64_t bufferToUInt64(Buffer& b);
    static std::string toExNNFormat(unsigned char& c);
    static Buffer hexStringToBufferHex(std::string& hexString);
    static uint32_t hexToInt(std::string&& hexStr);
    static uint32_t hexToInt16(std::string&& hexStr);
    static uint32_t hexToInt(char& hexChar);
    static uint64_t hexToInt64(std::string&& hexStr);
};
```



## Sha1.h

```
/*
    sha1.hpp - source code of
    =====
    SHA-1 in C++
    =====
    100% Public Domain.
    Original C Code
        -- Steve Reid <steve@edmweb.com>
    Small changes to fit into bglibs
        -- Bruce Guenter <bruce@untroubled.org>
    Translation to simpler C++ Code
        -- Volker Diels-Grabsch <v@njh.eu>
    Safety fixes
        -- Eugene Hopkinson <slowriot at voxelstorm dot com>
    Header-only library
        -- Zlatko Michailov <zlatko@michailov.org>
*/

#pragma once
#include <cstdint>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <string>
#include "Defs.h"

static const size_t BLOCK_INTS = 16; /* number of 32bit integers per SHA1 block */
static const size_t BLOCK_BYTES = BLOCK_INTS * 4;

class Sha1 {
public:
    Sha1();
    void update(const std::string& s);
    void update(std::istream& is);
    std::string final();
    static std::string from_file(const std::string& filename);

private:
    uint32_t digest[5];
    std::string buffer;
```

```
uint64_t transforms;  
};
```

## UrlUtils.h

```
#pragma once
#include <iostream>

enum Protocol {
    WSS,
    UDP,
    HTTPS,
    HTTP,
    UNKNOWN
};

class UrlUtils {
public:
    static std::string getDomain(std::string& url);
    static int getPort(std::string& url);
    static Protocol getProtocol(std::string& url);
};
```

## Downloader.h

```
#pragma once
#include "Peer.h"
#include "UrlUtils.h"

struct DownloadPeerParams {
    Peer* peer;
    Pieces* pieces;
    Buffer infoHash;

    DownloadPeerParams(Peer* peer, Pieces* pieces, Buffer infoHash) {
        this->peer = peer;
        this->pieces = pieces;
        this->infoHash = infoHash;
    }
};

class Downloader {
public:
    Downloader(std::string& filePath);
private:
    BenCodeObject& analyzeTorrent(std::string& filePath);
    void backupTrackers();
    void updateFromTracker(std::string& url);
    bool update(std::vector<PeerInfo>& peers);

    static DWORD WINAPI start(DownloadPeerParams* paramsRaw);
    BenCodeObject torrent;
    Pieces pieces;
    Buffer infoHash;

    int peersNum = 0;
};
```

## Bencode.cpp

```
#include <iostream>
#include "bencode.h"
#include "HexUtils.h"
#include "Exception.h"

BenCodeObject bencode::decode(std::string& decodedStr) {
    int index = 0;
    return decode(decodedStr, &index);
}

std::string bencode::encode(BenCodeObject obj) {
    if (obj.bType == BenCodeType::BenCode_Integer) {
        return encodeInteger(obj.getInteger());
    }
    else if (obj.bType == BenCodeType::BenCode_List) {
        return encodeList(*obj.getList());
    }
    else if (obj.bType == BenCodeType::BenCode_Dictionary) {
        return encodeDictionary(*obj.getDictionary());
    }
    else {
        return encodeString(obj.getString());
    }
}

BenCodeObject bencode::decode(std::string& decodedStr, int* i) {
    if (decodedStr[*i] == 'i') {
        return BenCodeObject(decodeInteger(decodedStr, i));
    }
    else if (decodedStr[*i] == 'l') {
        return BenCodeObject(decodeList(decodedStr, i));
    }
    else if (decodedStr[*i] == 'd') {
        return BenCodeObject(decodeDictionary(decodedStr, i));
    }
    else {
        return BenCodeObject(decodeString(decodedStr, i));
    }
}

BenCodeInteger bencode::decodeInteger(std::string& decodedStr, int* i) {
    *i += 1;
    BenCodeInteger retValue = getInBenCodeInteger(decodedStr, i);
```

```

        return BenCodeInteger(retValue);
    }

BenCodeString bencode::decodeString(std::string& decodedStr, int* i) {
    // <length>:<contents>: 4:spam -> spam
    BenCodeInteger numberOfBytes = getInBenCodeInteger(decodedStr, i, ':');

    BenCodeString retValue;

    for (int j = 0; j < numberOfBytes; j++) {
        retValue.push_back(decodedStr[*i]);
        *i += 1;
    }

    return retValue;
}

BenCodeList<BenCodeObject> bencode::decodeList(std::string& decodedStr, int* i) {
    // l<contents>e: l4:spami42ee -> [spam, 42]
    BenCodeList<BenCodeObject> retValue;
    *i += 1;
    while (decodedStr[*i] != 'e') {
        retValue.push_back(decode(decodedStr, i));
    }
    *i += 1;
    return retValue;
}

BenCodeDictionary<BenCodeObject> bencode::decodeDictionary(std::string& decodedStr,
int* i) {
    // starts with d and ends with 3: d3:bar4:spam3:fooi42ee -> {"bar": "spam",
"foo": 42}
    BenCodeDictionary<BenCodeObject> retValue;
    *i += 1;
    while (decodedStr[*i] != 'e') {
        BenCodeString key = decodeString(decodedStr, i);
        retValue[key] = decode(decodedStr, i);
    }
    *i += 1;
    return retValue;
}

```

```

BenCodeInteger bencode::getInBenCodeInteger(std::string& decodedStr, int* i, char
ending) {
    // The interager starts with an i and ends with e. for exanple: i42e -> 42
    int retValue = 0;
    int sign = 1;

    if (decodedStr[*i] == '-') {
        sign = -1;
        *i += 1;
    }

    while (decodedStr[*i] != ending) {
        retValue = retValue * 10 + (decodedStr[*i] - '0');
        *i += 1;
        if (*i >= decodedStr.size()) {
            throw Exception(INVALID_BENCODE_STRING);
        }
    }
    *i += 1;
    return retValue * sign;
    return BenCodeInteger(retValue);
}

std::string bencode::encodeInteger(BenCodeInteger obj) {
    std::string retValue = "i" + std::to_string(obj) + "e";
    return retValue;
}

std::string bencode::encodeString(BenCodeString obj) {
    std::string retValue = std::to_string(obj.size()) + ":";
    for (int i = 0; i < obj.size(); i++) {
        retValue += obj[i];
    }
    return retValue;
}

std::string bencode::encodeList(BenCodeList<BenCodeObject> obj) {
    std::string retValue = "[";
    for (BenCodeObject& o : obj) {
        retValue += encode(o);
    }
}

```

```

    retValue += "e";
    return retValue;
}

std::string bencode::encodeDictionary(BenCodeDictionary<BenCodeObject> obj) {
    std::string retValue = "d";
    for (auto& pair : obj) {
        retValue += encodeString(pair.first);
        retValue += encode(pair.second);
    }
    retValue += "e";
    return retValue;
}

```



## BenCodeObject.cpp

```
#include "BenCodeObject.h"

BenCodeObject::BenCodeObject()
{
}

BenCodeObject::BenCodeObject(BenCodeInteger v) {
    value.benCodeInteger = v;
    bType = BenCodeType::BenCode_Integer;
}

BenCodeObject::BenCodeObject(BenCodeString&& v) {
    value.benCodeString = new BenCodeString(std::move(v));
    bType = BenCodeType::BenCode_String;
}

BenCodeObject::BenCodeObject(BenCodeList<BenCodeObject>&& v) {
    value.benCodeList = new BenCodeList<BenCodeObject>(std::move(v));
    bType = BenCodeType::BenCode_List;
}

BenCodeObject::BenCodeObject(BenCodeDictionary<BenCodeObject>&& v) {
    value.benCodeDictionary = new BenCodeDictionary<BenCodeObject>(std::move(v));
    bType = BenCodeType::BenCode_Dictionary;
}

BenCodeInteger BenCodeObject::getInteger() {
    if (bType == BenCodeType::BenCode_Integer) {
        return value.benCodeInteger;
    }
    else {
        throw Exception("BenCodeObject is not an integer");
    }
}

BenCodeString BenCodeObject::getString() {
    if (bType == BenCodeType::BenCode_String) {
        return *value.benCodeString;
    }
    else {
```

```

        throw Exception("BenCodeObject is not a string");
    }
}

BenCodeList<BenCodeObject>* BenCodeObject::getList() {
    if (bType == BenCodeType::BenCode_List) {
        return value.benCodeList;
    }
    else {
        throw Exception("BenCodeObject is not a list");
    }
}

BenCodeDictionary<BenCodeObject>* BenCodeObject::getDictionary() {
    if (bType == BenCodeType::BenCode_Dictionary) {
        return value.benCodeDictionary;
    }
    else {
        throw Exception("BenCodeObject is not a dictionary");
    }
}

```

## Messages.cpp

```
#include "Messages.h"
```

```
std::string Messages::buildHandShake(Buffer& infoHash, Buffer& peerId) {  
    Buffer buf;  
    // pstrlen  
    HexUtils::writeUInt8(&buf, 19);  
    // pstr  
    HexUtils::writeString(&buf, "BitTorrent protocol");  
    // reserved  
    HexUtils::writeUInt64BE(&buf, 0);  
    // info hash  
    HexUtils::writeBytes(&buf, infoHash);  
    // peer id  
    HexUtils::writeBytes(&buf, peerId);  
  
    return std::string(buf.begin(), buf.end());  
}
```

```
std::string Messages::buildKeepAlive() {  
    Buffer buf;  
    HexUtils::writeUInt32BE(&buf, 0);  
    return std::string(buf.begin(), buf.end());  
}
```

```
std::string Messages::buildChoke() {  
    Buffer buf;  
    // length  
    HexUtils::writeUInt32BE(&buf, 1);  
    // id  
    HexUtils::writeUInt8(&buf, 0);  
    return std::string(buf.begin(), buf.end());  
}
```

```
std::string Messages::buildUnchoke() {  
    Buffer buf;  
    // length  
    HexUtils::writeUInt32BE(&buf, 1);  
    // id  
    HexUtils::writeUInt8(&buf, 1);  
    return std::string(buf.begin(), buf.end());  
}
```

```

}

std::string Messages::buildInterested() {
    Buffer buf;
    // length
    HexUtils::writeUInt32BE(&buf, 1);
    // id
    HexUtils::writeUInt8(&buf, 2);
    return std::string(buf.begin(), buf.end());
}

std::string Messages::buildUnInterested() {
    Buffer buf;
    // length
    HexUtils::writeUInt32BE(&buf, 1);
    // id
    HexUtils::writeUInt8(&buf, 3);
    return std::string(buf.begin(), buf.end());
}

std::string Messages::buildHave(int pieceIndex)
{
    return std::string();
}

std::string Messages::buildBitfield(std::vector<bool> bitfield)
{
    return std::string();
}

std::string Messages::buildRequest(PieceInfo info) {
    Buffer buf;
    // length
    HexUtils::writeUInt32BE(&buf, 13);
    // id
    HexUtils::writeUInt8(&buf, MessagesTypes::request);
    // index
    HexUtils::writeUInt32BE(&buf, info.index);
    // begin
    HexUtils::writeUInt32BE(&buf, info.begin);
    // len
    HexUtils::writeUInt32BE(&buf, info.len);
}

```

```

        return std::string(buf.begin(), buf.end());
    }

    std::string Messages::buildPiece(int pieceIndex, int blockIndex, std::string& block)
    {
        return std::string();
    }

    std::string Messages::buildCancel(int pieceIndex, int blockIndex)
    {
        return std::string();
    }

    std::string Messages::buildPort(int port)
    {
        return std::string();
    }

    BenCodeDictionary<BenCodeObject> Messages::parseResponse(std::string& msg) {
        if (msg.size() < 4) {
            throw Exception("Invalid response");
        }
        int id = msg[4];
        BenCodeDictionary<BenCodeObject> retVal;

        if (id < 6) {
            retVal["payload"] = BenCodeObject(msg.substr(5, msg.size() - 5));
        }
        else if (id == 6 || id == 7 || id == 8) {
            BenCodeDictionary<BenCodeObject> payload;

            payload["index"] = BenCodeObject(HexUtils::hexToInt(msg.substr(5, 4)));
            payload["begin"] = BenCodeObject(HexUtils::hexToInt(msg.substr(9, 4)));
            payload[id == 7 ? "block" : "length"] = BenCodeObject(msg.substr(13,
msg.size() - 13));

            retVal["payload"] =
BenCodeObject(BenCodeDictionary<BenCodeObject>(payload));
        }

        retVal["id"] = BenCodeObject(id);
        retVal["size"] = HexUtils::hexToInt(msg.substr(0, 4));
    }

```

```
        return retVal;  
    }
```

## Peer.cpp

```
#include <chrono>
#include <thread>
#include "Peer.h"
#include "File.h"

Peer::Peer(PeerInfo info): sock(SOCK_STREAM, IPPROTO_TCP), hostname(info.ip),
id(info.id) {
    sock.connectSocket(info.ip, info.port);
}

void Peer::download(Pieces* pieces, Buffer infoHash) {
    this->pieces = pieces;
    std::string handShakeMsg = Messages::buildHandShake(infoHash, id);
    sock.sendMessage(handShakeMsg);
    std::string response = sock.receiveMessage(68);

    messageHandler(response);
}

void Peer::sendKeepAlive() {
    std::string keepAliveMsg = Messages::buildKeepAlive();
    sock.sendMessage(keepAliveMsg);
}

void Peer::messageHandler(std::string& msg) {
    if (isHandShake(msg)) {
        std::string interestedMsg = Messages::buildInterested();
        sock.sendMessage(interestedMsg);
        std::string response = sock.onWholeMessage(false);
        response = sock.receiveMessage(5);
        BenCodeDictionary<BenCodeObject> res = Messages::parseResponse(response);
        int id = res["id"].getInteger();
        switch (id) {
            case MessagesTypes::choke:
                chokeHandler();
            case MessagesTypes::unchoke:
                unchokeHandler();
                break;
            case MessagesTypes::have:
                haveHandler(res["payload"]);
        }
    }
}
```

```

        break;
    case MessagesTypes::bitfield:
        bitfieldHandler(res["payload"]);
        break;
    case MessagesTypes::piece:
        pieceHandler(res["payload"]);
        break;
    }
}

bool Peer::isHandShake(std::string& msg) {
    if (msg.size() < 68) {
        return false;
    }
    if (msg.substr(1, 19) != "BitTorrent protocol") {
        return false;
    }
    return true;
}

void Peer::chokeHandler() {
    sock.closeSocket();
}

void Peer::unchokeHandler() {
    requestPiece();
}

void Peer::haveHandler(BenCodeObject payload) {
}

void Peer::bitfieldHandler(BenCodeObject payload) {
}

void Peer::pieceHandler(BenCodeObject payload) {
    BenCodeDictionary<BenCodeObject> p = *payload.getDictionary();
    std::string pie = p["block"].getString();
    PieceInfo info;
    info.begin = p["begin"].getInteger();
    info.index = p["index"].getInteger();
}

```



```

    Buffer buf;
    for (int i = 0; i < pie.size(); i++) {
        buf.push_back(pie[i]);
    }

    pieces->addPiece(info, buf);
    pieces->printPerstage();

    if (pieces->onDone()) {
        pieces->downloadFilesToDisk();
    }
}

void Peer::requestPiece() {
    PieceInfo next = pieces->pieceNeeded();

    while (next.len != 0) {
        try {
            std::string reqPiece = Messages::buildRequest(next);

            sock.sendMessage(reqPiece);
            std::string response = sock.onWholeMessage(false);

            while (response.size() < 5) { // keep alive message
                response = sock.onWholeMessage(false);
            }

            BenCodeDictionary<BenCodeObject> res =
Messages::parseResponse(response);
            int id = res["id"].getInteger();

            if (id == MessagesTypes::piece) {
                pieceHandler(res["payload"]);
            }
        }
        catch (const Exception& e) {
            pieces->failedPiece(next);
            requestPiece();
        }
        next = pieces->pieceNeeded();
    }
}

```

```
std::this_thread::sleep_for(std::chrono::seconds(5));

if (pieces->onDone()) {
    pieces->downloadFilesToDisk();
}

sock.closeSocket();
}
```

## Pieces.cpp

```
#include "Pieces.h"
#include "HexUtils.h"

Pieces::Pieces(BenCodeObject& torrent): torrent(torrent), mutexHandle(createLock()) {
    int pieceLength = TorrentParser::numberPieces(torrent);
    pieces.resize(pieceLength);
    requested.resize(pieceLength);

    for (int i = 0; i < pieceLength; i++) {
        size_t blocksPerPiece = TorrentParser::blocksPerPiece(torrent, i);
        pieces[i].resize(blocksPerPiece);
        requested[i].resize(blocksPerPiece);

        for (size_t j = 0; j < blocksPerPiece; j++) {
            requested[i][j] = false;
        }
    }
}

void Pieces::addPiece(PieceInfo piece, Buffer& block) {

    while (block.size() > BLOCKSIZE) {
        block.pop_back();
    }

    pieces[piece.index][piece.begin / BLOCKSIZE] = block;
}

void Pieces::checkPiece(int index) {
    std::string piece = "";
    for (int i = 0; i < pieces[index].size(); i++) {
        Buffer buf = pieces[index][i];
        for (int j = 0; j < buf.size(); j++) {
            piece += buf[j];
        }
    }

    BenCodeObject info = TorrentParser::getInfo(torrent);
    BenCodeDictionary<BenCodeObject> infoDict = *info.getDictionary();
    std::string piecesHash = infoDict["pieces"].getString();
}
```

```

std::string sha1Piece = TorrentParser::infoHash(piece);

for (int i = index*20; i < index*20 + 20; i++) {
    unsigned char c = piecesHash[i];
    std::string piecesHashNN = HexUtils::toEexNNFormat(c);
    for (int j = 0; j < 2; j++) {
        if (piecesHashNN[j] != sha1Piece[(i-index*20)*2+j]) {
            throw Exception("Error in piece, not the same sha1");
        }
    }
}

}

PieceInfo Pieces::pieceNeeded() {
    DWORD waitResult = WaitForSingleObject(
        mutexHandle.getRawHandle(),
        INFINITE);
    if (waitResult == WAIT_OBJECT_0) {
        for (int i = 0; i < pieces.size(); i++) {
            for (int j = 0; j < pieces[i].size(); j++) {
                if (pieces[i][j].size() == 0 && !requested[i][j]) {
                    requested[i][j] = true;
                    ReleaseMutex(mutexHandle.getRawHandle());
                    return PieceInfo(i, j * BLOCKSIZE,
TorrentParser::blockLen(torrent, i, j));
                }
            }
        }
    }

    ReleaseMutex(mutexHandle.getRawHandle());
    return PieceInfo(0, 0, 0);
}

void Pieces::failedPiece(PieceInfo info) {
    requested[info.index][info.begin/BLOCKSIZE] = false;
    pieces[info.index][info.begin / BLOCKSIZE].clear();
}

bool Pieces::onDone() {
    size_t total = TorrentParser::fileSize(torrent);
    size_t downloadedB = downloaded();

```

```

        return downloadedB == total;
    }

void Pieces::printPerstage() {
    size_t downloadedB = downloaded();
    size_t total = TorrentParser::fileSize(torrent);

    double_t perstage = (double)downloadedB / (double)total * 100;
    perstage = std::round(perstage * 100) / 100;

    std::cout << "Downloaded: " << perstage << "%" << std::endl;
    std::cout << "Downloaded: " << downloadedB << " / " << total << std::endl;
}

void Pieces::downloadFilesToDisk() {
    if (startDownloadedToDisk) {
        return;
    }
    std::cout << "Downloading to disk..." << std::endl;
    startDownloadedToDisk = true;
    BenCodeObject info = TorrentParser::getInfo(torrent);
    BenCodeDictionary<BenCodeObject> infoDict = *info.getDictionary();

    Buffer allPieces;

    for (int i = 0; i < pieces.size(); i++) {
        for (int j = 0; j < pieces[i].size(); j++) {
            for (int k = 0; k < pieces[i][j].size(); k++) {
                allPieces.push_back(pieces[i][j][k]);
            }
        }
    }

    if (infoDict.find("files") != infoDict.end()) {
        // multifile
        std::string downloadDir = FilePathUtils::appendPath(DOWNLOADPATH,
        (infoDict["name"]).getString());
        Directory dir(downloadDir);

        BenCodeObject files = infoDict["files"];
        BenCodeList<BenCodeObject> filesList = *files.getList();
    }
}

```

```

    int index = 0;
    for (auto file: filesList) {
        BenCodeDictionary<BenCodeObject> fileDict = *file.getDictionary();
        BenCodeObject path = fileDict["path"];
        BenCodeList<BenCodeObject> pathList = *path.getList();
        std::string pathString = "";
        for (auto p: pathList) {
            pathString += p.getString();
        }
        File f = dir.addFile(pathString);
        int fileLen = fileDict["length"].getInteger();
        Buffer fileBuf;
        int endOfFile = index + fileLen;
        for (index; index < endOfFile; index++) {
            fileBuf.push_back(allPieces[index]);
        }
        f.write(fileBuf);
        f.close();
    }
    std::cout << "Done: " << downloadDir << std::endl;

}

else if (infoDict.find("length") != infoDict.end()) {
    // single file
    std::string downloadDir = FilePathUtils::appendPath(DOWNLOADPATH,
(infoDict["name"]).getString());
    File f(downloadDir, true);
    f.write(allPieces);
    f.close();
    std::cout << "Done: " << downloadDir << std::endl;
}

else {
    throw Exception("Error in torrent file, no length or files");
}

}

size_t Pieces::downloaded() {
    size_t downloaded = 0;
    for (int i = 0; i < pieces.size(); i++) {
        for (int j = 0; j < pieces[i].size(); j++) {
            downloaded += pieces[i][j].size();
        }
    }
}

```

```
    }  
    return downloaded;  
}  
  
Handle Pieces::createLock() const {  
    const auto fileHandle = CreateMutexA(  
        NULL,  
        FALSE,  
        "Hello");  
    return Handle(fileHandle);  
}
```

## TorrentParser.cpp

```
#include "TorrentParser.h"
#include "HexUtils.h"

std::string TorrentParser::infoHash(BenCodeObject& torrent) {
    BenCodeObject info = getInfo(torrent);
    std::string encoded = bencode::encode(info);
    return infoHash(encoded);
}

size_t TorrentParser::numberPieces(BenCodeObject& torrent) {
    BenCodeObject info = getInfo(torrent);
    BenCodeDictionary<BenCodeObject> infoDict = *info.getDictionary();
    BenCodeObject pieces = infoDict["pieces"];
    return pieces.getString().size() / 20;
}

size_t TorrentParser::fileSize(BenCodeObject& torrent) {
    BenCodeObject info = getInfo(torrent);

    BenCodeDictionary<BenCodeObject> infoDict = *info.getDictionary();

    if (infoDict.find("length") != infoDict.end()) {
        return infoDict["length"].getInteger();
    }
    else if (infoDict.find("files") != infoDict.end()) {
        BenCodeList<BenCodeObject> files = *infoDict["files"].getList();
        size_t fSize = 0;
        for (auto it = files.begin(); it != files.end(); it++) {
            BenCodeDictionary<BenCodeObject> file = *it->getDictionary();
            if (file.find("length") == file.end()) {
                throw Exception("torrent file contains a file with no
length");
            }
            fSize += file["length"].getInteger();
        }
        return fSize;
    }

    throw Exception("torrent file has no length");
}
```



```

}

size_t TorrentParser::pieceLength(BenCodeObject& torrent, int pieceIndex) {
    size_t totalSize = fileSize(torrent);

    BenCodeObject info = getInfo(torrent);

    BenCodeDictionary<BenCodeObject> infoDict = *info.getDictionary();

    size_t pieceLength = infoDict["piece length"].getInteger();

    size_t lastPieceLength = totalSize % pieceLength;
    int lastPieceIndex = std::floor(totalSize / pieceLength);

    if (pieceIndex == lastPieceIndex) {
        return lastPieceLength;
    }
    else {
        return pieceLength;
    }
}

size_t TorrentParser::blocksPerPiece(BenCodeObject& torrent, int pieceIndex) {
    size_t pieceLen = pieceLength(torrent, pieceIndex);

    double blocksPerPiece = (double)pieceLen / (double)BLOCKSIZE;
    return std::ceil(blocksPerPiece);
}

size_t TorrentParser::blockLen(BenCodeObject& torrent, int pieceIndex, int blockIndex)
{
    size_t pieceLen = pieceLength(torrent, pieceIndex);

    size_t lastPieceLength = pieceLen % BLOCKSIZE;
    int lastPieceIndex = std::floor(pieceLen / BLOCKSIZE);

    if (blockIndex == lastPieceIndex) {
        return lastPieceLength;
    }
    else {
        return BLOCKSIZE;
    }
}

```

```

    }
}

BenCodeObject TorrentParser::getInfo(BenCodeObject& torrent) {
    if (torrent.bType != BenCodeType::BenCode_Dictionary) {
        throw Exception("torrent file is not a dictionary");
    }
    BenCodeDictionary<BenCodeObject> dict = *torrent.getDictionary();
    if (dict.find("info") == dict.end()) {
        throw Exception("torrent file has no info dictionary");
    }
    BenCodeObject& info = dict["info"];
    if (info.bType != BenCodeType::BenCode_Dictionary) {
        throw Exception("info is not a dictionary");
    }
    return info;
}

std::string TorrentParser::infoHash(std::string& torrent) {
    Sha1 hashFun;
    hashFun.update(torrent);
    return hashFun.final();
}

```

## Tracker.cpp

```
#include "Tracker.h"

Tracker::Tracker(std::string host, int port): sock(SOCK_DGRAM, IPPROTO_UDP) {
    sock.connectSocket(host, port);
}

std::vector<PeerInfo> Tracker::getPeers(BenCodeObject& torrent) {
    std::string connReq = buildConnReq();
    sock.sendMessage(connReq);
    std::string response = sock.receiveMessage(1024);
    analyzeConnResponse(response);
    std::string annoReq = buildAnnounceReq(torrent);
    sock.sendMessage(annoReq);
    response = sock.receiveMessage(1024);
    return analyzeAnnounceResponse(response);
}

void Tracker::analyzeConnResponse(std::string& response) {
    if (response.size() != 16) {
        throw Exception("Not a valid connection response");
    }
    for (int i = 0; i < 4; i++) {
        if (response[i] != '\0') {
            throw Exception("Not a valid connection response");
        }
    }
    if (transactionId != HexUtils::hexToInt(response.substr(4, 8))) {
        throw Exception("Not a valid connection response");
    }
    connectionId = HexUtils::hexToInt64(response.substr(8, 8));
}

std::vector<PeerInfo> Tracker::analyzeAnnounceResponse(std::string& response) {
    if (response.size() < 20) {
        throw Exception("Not a valid announce response");
    }

    if (0x0001 != HexUtils::hexToInt(response.substr(0, 8))) {
        throw Exception("Not a valid announce response");
    }
}
```

```

    }

    if (transactionId != HexUtils::hexToInt(response.substr(4, 8))) {
        throw Exception("Not a valid announce response");
    }

    int interval = HexUtils::hexToInt(response.substr(8, 8));
    int leechers = HexUtils::hexToInt(response.substr(12, 8));
    int seeders = HexUtils::hexToInt(response.substr(16, 8));
    std::vector<PeerInfo> peers;
    for (int i = 20; i < response.size(); i += 6) {
        PeerInfo p;
        std::string ip = "";

        for (int j = 0; j < 4; j++) {
            unsigned char c = response[i + j];
            std::string s = std::to_string(c);

            for (int k = 0; k < s.size(); k++) {
                ip.push_back(s[k]);
            }

            if (j != 3) ip.push_back('.');
        }
        p.ip = ip;
        p.port = HexUtils::hexToInt16(response.substr(i + 4, 2));
        p.id = peerId;
        peers.push_back(p);
    }
    return peers;
}

std::string Tracker::buildConnReq() {
    Buffer buf;
    // connection id
    HexUtils::writeUInt32BE(&buf, 0x417);
    HexUtils::writeUInt32BE(&buf, 0x27101980);
    // action
    HexUtils::writeUInt32BE(&buf, 0);
    // transaction id
    transactionId = CryptoGen::randomBytes(4);
    HexUtils::writeUInt32BE(&buf, transactionId);
}

```

```

        return std::string(buf.begin(), buf.end());
    }

std::string Tracker::buildAnnounceReq(BenCodeObject& torrent) {
    Buffer buf;
    // connection id
    HexUtils::writeUInt64BE(&buf, connectionId);
    // action
    HexUtils::writeUInt32BE(&buf, 1);
    // transaction id
    transactionId = CryptoGen::randomBytes(4);
    HexUtils::writeUInt32BE(&buf, transactionId);
    // info hash
    std::string info = TorrentParser::infoHash(torrent);
    infoHash = HexUtils::hexStringToBufferHex(info);
    HexUtils::writeBytes(&buf, infoHash);
    // peer id
    if (peerId.size() == 0) {
        generatePeerId();
    }
    HexUtils::writeBytes(&buf, peerId);
    // downloaded
    HexUtils::writeUInt64BE(&buf, 0);
    // left
    HexUtils::writeUInt64BE(&buf, TorrentParser::fileSize(torrent));
    // uploaded
    HexUtils::writeUInt64BE(&buf, 0);
    // event
    HexUtils::writeUInt32BE(&buf, 0);
    // ip address
    HexUtils::writeUInt32BE(&buf, 0);
    // key
    HexUtils::writeUInt32BE(&buf, CryptoGen::randomBytes(4));
    // num want
    HexUtils::writeUInt32BE(&buf, -1);
    // port
    HexUtils::writeUInt16BE(&buf, 6881);

    return std::string(buf.begin(), buf.end());
}

```

```
void Tracker::generatePeerId() {  
    peerId.push_back('F'); // Franco  
    peerId.push_back('T'); // Torrent  
    peerId.push_back('2'); // 2.0  
    Buffer randomBytes = CryptoGen::randomBytesBuffer(17);  
    for (int i = 0; i < randomBytes.size(); i++) {  
        peerId.push_back(randomBytes[i]);  
    }  
}
```

## File.cpp

```
#include <sstream>
#include "File.h"

File::File(const std::string& path, bool createNew) :
    m_file(openFileInternal(path, createNew)) {

}

Buffer File::read(uint32_t numberOfBytes, LONG startOffset) {
    Buffer data(numberOfBytes);
    DWORD outBytes = 0;

    if (SetFilePointer(m_file.getRawHandle(), startOffset, NULL, FILE_BEGIN) ==
INVALID_SET_FILE_POINTER) {
        throw Exception(SET_START_FILE_POINTER_EXCEPTION);
    }

    if (!ReadFile(m_file.getRawHandle(), data.data(), numberOfBytes, &outBytes,
nullptr)) {
        throw Exception(READ_FROM_FILE_EXCEPTION);
    }

    if (outBytes != numberOfBytes) {
        throw Exception(READ_FROM_FILE_EXCEPTION);
    }

    return data;
}

Buffer File::readAll() {
    return read(getSize(), 0);
}

LinesBuffer File::readLines(uint32_t numberOfBytes, LONG startOffset) {
    LinesBuffer lines;
    Buffer data1 = read(numberOfBytes, startOffset);
    std::string line;
    std::stringstream stringstream;
    std::copy(data1.begin(), data1.end(), std::ostream_iterator<unsigned
char>(stringstream));
```

```

        while (std::getline(stringstream, line)) {
            line.erase(line.end() - 1);
            lines.push_back(line);
        }

        return lines;
    }

void File::write(const Buffer& buffer) {
    DWORD outBytes = 0;
    if (!WriteFile(m_file.getRawHandle(), buffer.data(), buffer.size(), &outBytes,
        nullptr)) {
        throw Exception(WRITE_TO_FILE_EXCEPTION);
    }

    if (outBytes != buffer.size()) {
        throw Exception(WRITE_TO_FILE_EXCEPTION);
    }
}

void File::writeLines(const LinesBuffer& buffer) {
    Buffer b;
    for (const auto& line : buffer) {
        for (const auto& c : line) {
            b.push_back(c);
        }
        b.push_back('\r');
        b.push_back('\n');
    }
    write(b);
}

int File::getSize() {
    DWORD fileSize = GetFileSize(m_file.getRawHandle(), nullptr);
    return fileSize;
}

bool File::compareFiles(File& other) {
    DWORD fileSize = getSize();
    DWORD otherFileSize = other.getSize();

```



```

        if (fileSize != otherFileSize) {
            return false;
        }
        Buffer data1 = read(fileSize, 0);
        Buffer data2 = other.read(otherFileSize, 0);
        return std::equal(data1.begin(), data1.end(), data2.begin());
    }

    void File::close() {
        m_file.close();
    }

    void File::rename(const std::string& oldPath, const std::string& newPath) {
        // File needs to be closed for rename to work
        if (!MoveFileA(oldPath.c_str(), newPath.c_str())) {
            throw std::exception(RENAME_FILE_Exception);
        }
    }

    bool File::exsits(const std::string& filePath) {
        DWORD dwAttrib = GetFileAttributesA(filePath.c_str());

        return (dwAttrib != INVALID_FILE_ATTRIBUTES &&
            !(dwAttrib & FILE_ATTRIBUTE_DIRECTORY));
    }

    void File::deleteFile(const std::string& filePath) {
        if (!File::exsits(filePath)) return;
        if (!DeleteFileA(filePath.c_str())) {
            throw Exception(DELETE_FILE_EXCEPTION);
        }
    }

    bool File::compareFiles(const std::string& path1, const std::string& path2) {
        File file1(path1);
        File file2(path2);
        return file1.compareFiles(file2);
    }

    Handle File::openFileInternal(const std::string& filePath, bool createNew) const {
        bool fileExsits = File::exsits(filePath);
        if (createNew && fileExsits) {

```

```

        throw Exception(FILE_EXSITS_EXCEPTION);
    }
    else if (!createNew && !fileExsits) {
        throw Exception(FILE_NOT_EXSITS_EXCEPTION);
    }
    const auto fileHandle =
        CreateFileA(filePath.c_str(),
            GENERIC_READ | FILE_APPEND_DATA,
            FILE_SHARE_READ | FILE_SHARE_WRITE,
            nullptr,
            createNew ? CREATE_NEW : OPEN_EXISTING,
            0,
            nullptr);
    return Handle(fileHandle);
}

```

## Handle.cpp

```
#include "Handle.h"
#include "Exception.h"

Handle::Handle(Handle&& other) : m_handle(std::exchange(other.m_handle,
INVALID_HANDLE_VALUE)) {

}

Handle::Handle(HANDLE handle) : m_handle(handle) {
    if (!isValid()) {
        throw Exception(INAVILD_HANDLE_EXCEPTION);
    }
}

void Handle::close() {
    if (isValid()) {
        CloseHandle(m_handle);
        m_handle = INVALID_HANDLE_VALUE;
    }
}

bool Handle::isValid() const {
    return m_handle != nullptr && m_handle != INVALID_HANDLE_VALUE;
}

HANDLE Handle::getRawHandle() {
    return m_handle;
}

Handle::~Handle() {
    try {
        close();
    }
    catch (...) {}
}
```

## Socket.cpp

```
#include "Socket.h"
```

```
Socket::Socket(int sType, int sProtocol): sType(sType), sProtocol(sProtocol) {  
    if ((sock = socket(AF_INET, sType, sProtocol)) == INVALID_SOCKET) {  
        throw Exception("socket failed");  
        closeSocket();  
    }  
}
```

```
Socket::~Socket() {  
    try {  
        closeSocket();  
    }  
    catch (...) {  
    }  
}
```

```
void Socket::connectSocket(std::string& hostname, int port) {  
    struct addrinfo hints = {}, * addrs;  
  
    hints.ai_family = AF_INET;  
  
    hints.ai_socktype = sType;  
    hints.ai_protocol = sProtocol;  
  
    if (getaddrinfo(hostname.c_str(), std::to_string(port).c_str(), &hints, &addrs)  
    != 0) {  
        throw Exception("getaddrinfo failed");  
    }  
  
    if (connect(sock, addrs->ai_addr, addrs->ai_addrlen) != 0) {  
        throw Exception("connected failed");  
    }  
    freeaddrinfo(addrs);  
}
```

```

void Socket::sendMessage(std::string& message) {
    if (send(sock, message.c_str(), message.length(), 0) == SOCKET_ERROR) {
        throw Exception("Error in send");
    }
}

std::string Socket::receiveMessage(int bytesToReceive) {
    char* buffer = new char[bytesToReceive];
    int bytesReceived = recv(sock, buffer, bytesToReceive, 0);
    if (bytesReceived == SOCKET_ERROR) {
        throw Exception("Error in recv");
    }
    std::string retVal = "";
    for (int i = 0; i < bytesReceived; i++) {
        retVal += (buffer)[i];
    }
    delete[] buffer;
    return retVal;
}

static int msgLen(bool handshake, Buffer* buf) {
    if (handshake) {
        return 19 + 49;
    }
    else {
        // read first 4 bytes in buf
        int len = (int)(*buf)[0] << 24 | (int)(*buf)[1] << 16 | (int)(*buf)[2] <<
8 | (int)(*buf)[3];

        return len + 4;
    }
}

std::string Socket::onWholeMessage(bool handshake) {
    Buffer buf;

    int bytesToReceive = 4;
    while (true) {
        std::string message = receiveMessage(bytesToReceive);
        for (auto c : message) {
            buf.push_back(c);
        }
    }
}

```

```

        bytesToReceive = msgLen(handshake, &buf) - 4;
        if (buf.size() >= msgLen(handshake, &buf)) {
            return std::string(buf.begin(), buf.end());
        }
    }
    return "";
}

void Socket::closeSocket() {
    if (closesocket(sock) != 0) {
        throw Exception("closesocket failed");
    }
}

```

## WinShok.cpp

```
#include "WinShok.h"

WinShok::WinShok() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        throw Exception("WSAStartup failed");
    }
}

WinShok::~WinShok() {
    try {
        if (WSACleanup() != 0) {
            throw Exception("WSACleanup failed");
        }
    }
    catch (...) {
    }
}
```

## CryptoGen.cpp

```
#include <iostream>
#include "CryptoGen.h"
#include "HexUtils.h"

uint32_t CryptoGen::randomBytes(size_t bSize) {
    Buffer ran = randomBytesBuffer(bSize);
    return HexUtils::bufferToUInt32(ran);
}

Buffer CryptoGen::randomBytesBuffer(size_t bSize) {
    unsigned x = randomGen();
    Buffer b(bSize);
    for (size_t i = 0; i < bSize; i++) {
        b[i] = x & 0xFF;
        x >>= 8;
        if (x == 0) {
            x = randomGen();
        }
    }
    return b;
}

unsigned CryptoGen::randomGen() {
    std::random_device engine;
    return engine();
}
```



## Directory.cpp

```
#include "Directory.h"

Directory::Directory(std::string& path): path(path) {
    createDir();
}

Directory::Directory(std::string&& path): path(path) {
    createDir();
}

File Directory::addFile(std::string& fileName) {
    std::string completePath = FilePathUtils::appendPath(path, fileName);
    return File(completePath, true);
}

void Directory::createDir() {
    bool res = CreateDirectoryA(path.c_str(), nullptr);

    if (res == 0) {
        throw Exception("CreateDirectoryA failed");
    }
}
```

## Exception.cpp

```
#include "Exception.h"

Exception::Exception(ExceptionType eType): eType(eType) {

}

void Exception::printException() {
    std::cout << "Exception: " << eType << std::endl;
}
```

## FilePathUtils.cpp

```
#include "FilePathUtils.h"
```

```
std::string FilePathUtils::getFileBasePath(const std::string& filename) {  
    std::string::size_type pos = filename.find_last_of("\\");  
  
    if (isEndOfFilePath(pos)) {  
        throw std::exception(INVALID_FILE_NAME_EXCEPTION);  
    }  
  
    return filename.substr(0, pos);  
}
```

```
std::string FilePathUtils::getFileName(const std::string& filePath) {  
    std::string::size_type pos = filePath.find_last_of("\\");  
  
    if (isEndOfFilePath(pos)) {  
        return filePath;  
    }  
  
    return filePath.substr(pos + 1);  
}
```

```
std::string FilePathUtils::generateFilePath(const std::string& path, const std::string&  
filename, const std::string& ending) {  
    return appendPath(path, filename) + "." + ending;  
}
```

```
std::string FilePathUtils::appendPath(const std::string& path1, const std::string&  
path2) {  
    return path1 + "\\ " + path2;  
}
```

```
bool FilePathUtils::isEndOfFilePath(const std::string::size_type& pos) {  
    return pos == std::string::npos;  
}
```

## HexUtils.cpp

```
#include "HexUtils.h"

void HexUtils::writeUInt8(Buffer* b, uint8_t val) {
    b->push_back(val);
}

void HexUtils::writeUInt16BE(Buffer* b, uint16_t val) {
    b->push_back((val >> 8) & 0xFF);
    b->push_back(val & 0xFF);
}

void HexUtils::writeUInt32BE(Buffer* b, uint32_t val) {
    b->push_back((val >> 24) & 0xFF);
    b->push_back((val >> 16) & 0xFF);
    b->push_back((val >> 8) & 0xFF);
    b->push_back(val & 0xFF);
}

void HexUtils::writeUInt64BE(Buffer* b, uint64_t val) {
    b->push_back((val >> 56) & 0xFF);
    b->push_back((val >> 48) & 0xFF);
    b->push_back((val >> 40) & 0xFF);
    b->push_back((val >> 32) & 0xFF);
    b->push_back((val >> 24) & 0xFF);
    b->push_back((val >> 16) & 0xFF);
    b->push_back((val >> 8) & 0xFF);
    b->push_back(val & 0xFF);
}

void HexUtils::writeBytes(Buffer* b, Buffer bVal) {
    for (uint32_t i = 0; i < bVal.size(); i++) {
        b->push_back(bVal[i]);
    }
}

void HexUtils::writeString(Buffer* b, std::string&& s) {
    for (uint32_t i = 0; i < s.length(); i++) {
        b->push_back(s[i]);
    }
}
```

```

    }
}

```

```

uint16_t HexUtils::bufferToUInt16(Buffer& b) {
    return (b[0] << 8) | b[1];
}

```

```

uint32_t HexUtils::bufferToUInt32(Buffer& b) {
    uint32_t val = 0;
    for (int i = 0; i < 4; i++) {
        val <<= 8;
        val |= b[i];
    }
    return val;
}

```

```

uint64_t HexUtils::bufferToUInt64(Buffer& b) {
    uint64_t val = 0;
    for (int i = 0; i < 8; i++) {
        val <<= 8;
        val |= b[i];
    }
    return val;
}

```

```

std::string HexUtils::toEexNNFormat(unsigned char& c) {
    std::string hexNNForamt = "";
    char hex[3] = { 0 };
    sprintf_s(hex, "%02x", c);
    hexNNForamt.push_back(hex[0]);
    hexNNForamt.push_back(hex[1]);
    return hexNNForamt;
}

```

```

Buffer HexUtils::hexStringToBufferHex(std::string& hexString) {
    Buffer buffer;
    for (int i = 0; i < hexString.length(); i += 2) {
        std::string hex = hexString.substr(i, 2);
        buffer.push_back(strtol(hex.c_str(), NULL, 16));
    }
}

```

```

        return buffer;
    }

    uint32_t HexUtils::hexToInt(std::string&& hexStr) {
        Buffer buf;
        for (int i = 0; i < hexStr.size(); i++) {
            buf.push_back(hexStr[i]);
        }
        return bufferToUInt32(buf);
    }

    uint32_t HexUtils::hexToInt16(std::string&& hexStr) {
        Buffer buf;
        for (int i = 0; i < hexStr.size(); i++) {
            buf.push_back(hexStr[i]);
        }
        return bufferToUInt16(buf);
    }

    uint32_t HexUtils::hexToInt(char& hexChar) {
        return (int)(hexChar);
    }

    uint64_t HexUtils::hexToInt64(std::string&& hexStr) {
        Buffer buf;
        for (int i = 0; i < 8; i++) {
            buf.push_back(hexStr[i]);
        }
        return bufferToUInt64(buf);
    }

```

## Sha1.cpp

```
#include "Sha1.h"
```

```
static void reset(uint32_t digest[], std::string& buffer, uint64_t& transforms)
{
    /* SHA1 initialization constants */
    digest[0] = 0x67452301;
    digest[1] = 0xefcdab89;
    digest[2] = 0x98badcfe;
    digest[3] = 0x10325476;
    digest[4] = 0xc3d2e1f0;

    /* Reset counters */
    buffer = "";
    transforms = 0;
}
```

```
static uint32_t rol(const uint32_t value, const size_t bits)
{
    return (value << bits) | (value >> (32 - bits));
}
```

```
inline static uint32_t blk(const uint32_t block[BLOCK_INTS], const size_t i)
{
    return rol(block[(i + 13) & 15] ^ block[(i + 8) & 15] ^ block[(i + 2) & 15] ^
block[i], 1);
}
```

```
/*
 * (R0+R1), R2, R3, R4 are the different operations used in SHA1
 */
```

```
static void R0(const uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t& w, const
uint32_t x, const uint32_t y, uint32_t& z, const size_t i)
{
    z += ((w & (x ^ y)) ^ y) + block[i] + 0x5a827999 + rol(v, 5);
    w = rol(w, 30);
}
```

```

static void R1(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t& w, const
uint32_t x, const uint32_t y, uint32_t& z, const size_t i)
{
    block[i] = blk(block, i);
    z += ((w & (x ^ y)) ^ y) + block[i] + 0x5a827999 + rol(v, 5);
    w = rol(w, 30);
}

```

```

static void R2(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t& w, const
uint32_t x, const uint32_t y, uint32_t& z, const size_t i)
{
    block[i] = blk(block, i);
    z += (w ^ x ^ y) + block[i] + 0x6ed9eba1 + rol(v, 5);
    w = rol(w, 30);
}

```

```

static void R3(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t& w, const
uint32_t x, const uint32_t y, uint32_t& z, const size_t i)
{
    block[i] = blk(block, i);
    z += (((w | x) & y) | (w & x)) + block[i] + 0x8f1bbcdc + rol(v, 5);
    w = rol(w, 30);
}

```

```

static void R4(uint32_t block[BLOCK_INTS], const uint32_t v, uint32_t& w, const
uint32_t x, const uint32_t y, uint32_t& z, const size_t i)
{
    block[i] = blk(block, i);
    z += (w ^ x ^ y) + block[i] + 0xca62c1d6 + rol(v, 5);
    w = rol(w, 30);
}

```

```

/*
 * Hash a single 512-bit block. This is the core of the algorithm.
 */

```



```

static void transform(uint32_t digest[], uint32_t block[BLOCK_INTS], uint64_t&
transforms)
{
    /* Copy digest[] to working vars */
    uint32_t a = digest[0];
    uint32_t b = digest[1];
    uint32_t c = digest[2];
    uint32_t d = digest[3];
    uint32_t e = digest[4];

    /* 4 rounds of 20 operations each. Loop unrolled. */
    R0(block, a, b, c, d, e, 0);
    R0(block, e, a, b, c, d, 1);
    R0(block, d, e, a, b, c, 2);
    R0(block, c, d, e, a, b, 3);
    R0(block, b, c, d, e, a, 4);
    R0(block, a, b, c, d, e, 5);
    R0(block, e, a, b, c, d, 6);
    R0(block, d, e, a, b, c, 7);
    R0(block, c, d, e, a, b, 8);
    R0(block, b, c, d, e, a, 9);
    R0(block, a, b, c, d, e, 10);
    R0(block, e, a, b, c, d, 11);
    R0(block, d, e, a, b, c, 12);
    R0(block, c, d, e, a, b, 13);
    R0(block, b, c, d, e, a, 14);
    R0(block, a, b, c, d, e, 15);
    R1(block, e, a, b, c, d, 0);
    R1(block, d, e, a, b, c, 1);
    R1(block, c, d, e, a, b, 2);
    R1(block, b, c, d, e, a, 3);
    R2(block, a, b, c, d, e, 4);
    R2(block, e, a, b, c, d, 5);
    R2(block, d, e, a, b, c, 6);
    R2(block, c, d, e, a, b, 7);
    R2(block, b, c, d, e, a, 8);
    R2(block, a, b, c, d, e, 9);
    R2(block, e, a, b, c, d, 10);
    R2(block, d, e, a, b, c, 11);
    R2(block, c, d, e, a, b, 12);
    R2(block, b, c, d, e, a, 13);
    R2(block, a, b, c, d, e, 14);

```

```

R2(block, e, a, b, c, d, 15);
R2(block, d, e, a, b, c, 0);
R2(block, c, d, e, a, b, 1);
R2(block, b, c, d, e, a, 2);
R2(block, a, b, c, d, e, 3);
R2(block, e, a, b, c, d, 4);
R2(block, d, e, a, b, c, 5);
R2(block, c, d, e, a, b, 6);
R2(block, b, c, d, e, a, 7);
R3(block, a, b, c, d, e, 8);
R3(block, e, a, b, c, d, 9);
R3(block, d, e, a, b, c, 10);
R3(block, c, d, e, a, b, 11);
R3(block, b, c, d, e, a, 12);
R3(block, a, b, c, d, e, 13);
R3(block, e, a, b, c, d, 14);
R3(block, d, e, a, b, c, 15);
R3(block, c, d, e, a, b, 0);
R3(block, b, c, d, e, a, 1);
R3(block, a, b, c, d, e, 2);
R3(block, e, a, b, c, d, 3);
R3(block, d, e, a, b, c, 4);
R3(block, c, d, e, a, b, 5);
R3(block, b, c, d, e, a, 6);
R3(block, a, b, c, d, e, 7);
R3(block, e, a, b, c, d, 8);
R3(block, d, e, a, b, c, 9);
R3(block, c, d, e, a, b, 10);
R3(block, b, c, d, e, a, 11);
R4(block, a, b, c, d, e, 12);
R4(block, e, a, b, c, d, 13);
R4(block, d, e, a, b, c, 14);
R4(block, c, d, e, a, b, 15);
R4(block, b, c, d, e, a, 0);
R4(block, a, b, c, d, e, 1);
R4(block, e, a, b, c, d, 2);
R4(block, d, e, a, b, c, 3);
R4(block, c, d, e, a, b, 4);
R4(block, b, c, d, e, a, 5);
R4(block, a, b, c, d, e, 6);
R4(block, e, a, b, c, d, 7);
R4(block, d, e, a, b, c, 8);

```

```

R4(block, c, d, e, a, b, 9);
R4(block, b, c, d, e, a, 10);
R4(block, a, b, c, d, e, 11);
R4(block, e, a, b, c, d, 12);
R4(block, d, e, a, b, c, 13);
R4(block, c, d, e, a, b, 14);
R4(block, b, c, d, e, a, 15);

/* Add the working vars back into digest[] */
digest[0] += a;
digest[1] += b;
digest[2] += c;
digest[3] += d;
digest[4] += e;

/* Count the number of transformations */
transforms++;
}

static void buffer_to_block(const std::string& buffer, uint32_t block[BLOCK_INTS])
{
    /* Convert the std::string (byte buffer) to a uint32_t array (MSB) */
    for (size_t i = 0; i < BLOCK_INTS; i++)
    {
        block[i] = (buffer[4 * i + 3] & 0xff)
            | (buffer[4 * i + 2] & 0xff) << 8
            | (buffer[4 * i + 1] & 0xff) << 16
            | (buffer[4 * i + 0] & 0xff) << 24;
    }
}

Sha1::Sha1()
{
    reset(digest, buffer, transforms);
}

void Sha1::update(const std::string& s)
{
    std::istringstream is(s);

```

```

        update(is);
    }

void Sha1::update(std::istream& is)
{
    while (true)
    {
        char sbuf[BLOCK_BYTES];
        is.read(sbuf, BLOCK_BYTES - buffer.size());
        buffer.append(sbuf, (std::size_t)is.gcount());
        if (buffer.size() != BLOCK_BYTES)
        {
            return;
        }
        uint32_t block[BLOCK_INTS];
        buffer_to_block(buffer, block);
        transform(digest, block, transforms);
        buffer.clear();
    }
}

/*
 * Add padding and return the message digest.
 */

std::string Sha1::final()
{
    /* Total number of hashed bits */
    uint64_t total_bits = (transforms * BLOCK_BYTES + buffer.size()) * 8;

    /* Padding */
    buffer += (char)0x80;
    size_t orig_size = buffer.size();
    while (buffer.size() < BLOCK_BYTES)
    {
        buffer += (char)0x00;
    }

    uint32_t block[BLOCK_INTS];
    buffer_to_block(buffer, block);

```

```

if (orig_size > BLOCK_BYTES - 8)
{
    transform(digest, block, transforms);
    for (size_t i = 0; i < BLOCK_INTS - 2; i++)
    {
        block[i] = 0;
    }
}

/* Append total_bits, split this uint64_t into two uint32_t */
block[BLOCK_INTS - 1] = (uint32_t)total_bits;
block[BLOCK_INTS - 2] = (uint32_t)(total_bits >> 32);
transform(digest, block, transforms);

/* Hex std::string */
std::ostringstream result;
for (size_t i = 0; i < sizeof(digest) / sizeof(digest[0]); i++)
{
    result << std::hex << std::setfill('0') << std::setw(8);
    result << digest[i];
}

/* Reset for next run */
reset(digest, buffer, transforms);

return result.str();
}

std::string Sha1::from_file(const std::string& filename)
{
    std::ifstream stream(filename.c_str(), std::ios::binary);
    Sha1 checksum;
    checksum.update(stream);
    return checksum.final();
}

```

## Downloader.cpp

```
#include "Downloader.h"
```

```
Downloader::Downloader(std::string& filePath): pieces(Pieces(analyzeTorrent(filePath)))
{
    // The functions receives a torrent file path and download the files
    std::string info = TorrentParser::infoHash(torrent);
    infoHash = HexUtils::hexStringToBufferHex(info);
    BenCodeDictionary<BenCodeObject> dict = *torrent.getDictionary();
    if (dict.find("announce") != dict.end()) {
        std::string url = dict["announce"].getString();
        if (UrlUtils::getProtocol(url) == Protocol::UDP) {
            // Find the correct Tracker, now we need to connect
            updateFromTracker(url);
        }
        else {
            backupTrackers();
        }
        if (peersNum == 0) {
            backupTrackers();
        }
    }
    else {
        throw Exception("No announce found in torrent");
    }
}
```

```
BenCodeObject& Downloader::analyzeTorrent(std::string& filePath) {
    // From torrent file to BenCodeObject that will contain all the info in a need
    form.
    File f(filePath);
    Buffer file = f.readAll();
    std::string fileStr(file.begin(), file.end());
    torrent = bencode::decode(fileStr);
    return torrent;
}
```

```
void Downloader::backupTrackers() {
    // backup trackers will be called when the trackers are failed to connect or they
    are not
}
```

```

// supported with UDP protocol
BenCodeDictionary<BenCodeObject> dict = *torrent.getDictionary();
if (dict.find("announce-list") != dict.end()) {
    BenCodeList<BenCodeObject> list = *dict["announce-list"].getList();
    for (BenCodeList<BenCodeObject>::iterator it = list.begin(); it !=
list.end(); ++it) {
        for (BenCodeList<BenCodeObject>::iterator it2 = (*it).getList()-
>begin(); it2 != (*it).getList()->end(); ++it2) {
            std::string url = (*it2).getString();
            if (UrlUtils::getProtocol(url) == Protocol::UDP) {
                updateFromTracker(url);
            }
            if (peersNum > 0) {
                return;
            }
        }
    }
}
}
}

```

```

void Downloader::updateFromTracker(std::string& url) {
    std::string hostname = UrlUtils::getDomain(url);

    std::cout << "Checking tracker: " << hostname << std::endl;
    if (hostname == "tracker.coppersurfer.tk") {
        return;
    }
    int port = UrlUtils::getPort(url);
    if (port == 0) {
        port = 80;
    }
    try {
        // calling the tracker class. The class will handle the connect
        // If the tracker failed to connect it wil throw exception
        Tracker t(hostname, port);
        // we will get the peers form the tracker
        std::vector<PeerInfo> peers = t.getPeers(torrent);
        update(peers);
    }
    catch (Exception& e) {

```

```

    }
}

bool Downloader::update(std::vector<PeerInfo>& peers) {
    // This function will loop throw the array of peers we will try to connect
    // to each one and if we can connect, we will send the peer to a Thread
    // and we will start downloading the wanted files.
    std::vector<HANDLE> threads;
    for (auto& peerInfo : peers) {
        try {
            auto peer = std::make_unique<Peer>(peerInfo);
            // Create a unique instance of the pieces class in order to prevent
downloading the
            // same pieces from multiple peers
            auto paramsPtr =
std::make_unique<DownloadPeerParams>(peer.release(), &pieces, infoHash);
            std::cout << "connected " << peerInfo.ip << " " << std::endl;
            HANDLE thread = CreateThread(NULL, 0,
                reinterpret_cast<LPTHREAD_START_ROUTINE>(start),
                paramsPtr.release(), 0, NULL);
            threads.push_back(thread);
            peersNum += 1;
        }
        catch (Exception e) {
            std::cout << "failed to connect " << peerInfo.ip << " " << std::endl;
        }
    }

    if (threads.size() == 0) {
        return false;
    }

    // Waiting for all the threads to finish, witch means that file has been
downloaded!
    DWORD waitStatus = WaitForMultipleObjects(threads.size(), threads.data(), true,
INFINITE);
    if (waitStatus < WAIT_OBJECT_0 || waitStatus >= WAIT_OBJECT_0 + threads.size()) {
        throw Exception("Error while waiting for threads to finish");
    }

    return true;
}

```



```

DWORD __stdcall Downloader::start(DonwloadPeerParams* paramsRaw) {
    // This is the thread function, it will call a peer that will start downloading
    the files
    try {
        std::unique_ptr<DonwloadPeerParams> mergeParams(paramsRaw);
        paramsRaw->peer->download(paramsRaw->pieces, paramsRaw->infoHash);
    }
    catch (...) {

    }
    return 0;
}

```

## BenCodeTests.cpp

```
#include "pch.h"
#include "BitTorrent/bencode.h"

// We will test here the class BenCodeObject
TEST(BenCodeTest, TestIntegerType) {
    std::string bencodeStr = "i9e";
    BenCodeObject res = bencode::decode(bencodeStr);
    EXPECT_EQ(res.bType, BenCode_Integer);
}

TEST(BenCodeTest, TestIntegerValue) {
    std::string bencodeStr = "i-9e";
    BenCodeObject res = bencode::decode(bencodeStr);
    EXPECT_EQ(res.getInteger(), -9);
}

TEST(BenCodeTest, TestEncodeInteger) {
    BenCodeObject obj = BenCodeObject(-9);
    std::string res = bencode::encode(obj);
    EXPECT_EQ(res, "i-9e");
}
```

## UrlUtilsTests.cpp

```
#include "pch.h"
#include "BitTorrent/UrlUtils.h"

// // We will test here the class UrlUtils
TEST(UrlUtilsTest, TestDomainName) {
    std::string url = "udp://tracker.openbittorrent.com:80";
    std::string domain = UrlUtils::getDomain(url);
    EXPECT_EQ(domain, "tracker.openbittorrent.com");
}

TEST(UrlUtilsTest, TestPort) {
    std::string url = "udp://tracker.openbittorrent.com:80";
    int port = UrlUtils::getPort(url);
    EXPECT_EQ(port, 80);
}
```