תרגיל בית 3

<u>מועד הגשת התרגיל:</u> עד יום ראשון 20/12/20 בשעה 23:55. לא תהינה דחיות

בודק אחראי: אורי ברכה uribracha@mail.tau.ac.il

מטרת התרגיל

- העמקת ההבנה והשימוש ב-thread-ים.
- עבודה במקביל עם מספר רב של thread-ים.
- שימוש ב-Mutex ו-Semaphore לסנכרון גישה לזיכרון משותף בין חוטים.
- הימנעות מ-deadlock-ים והתמודדות עם מצבים שעשויים ליצור deadlock-ים.

הגשה

צורת ההגשה מפורטת במסמך "הנחיות להגשת תרגילי בית – תשפ"א" שבאתר המודל. אנה הקפידו למלא אחר ההוראות.

הגישו פרוייקט מלא, כולל קבצי פרויקט (*.sln, *.vcxproj, *.vcxproj.filters) של Visual Studio 2019, באופן שיאפשר לבודק התרגילים לפתוח את הפרויקט על ידי לחיצה כפולה על קובץ ה-solution ולקמפל את הפרויקט ללא **אזהרות** או **שגיאות**.

הגישו בנוסף את תיקיית ה-Debug עם ה-Exe-ים.

אין צורך להגיש את תיקיית ה-vs.

דגשים

הקפידו על קוד קריא ומתועד.

עבדו באיטרציות. בדקו את הקוד שכתבתם לפחות בסוף כל סעיף.

זכרו להשתמש בכלי הדיבוג שה-IDE מספק.

הפורום עומד לשירותכם. אנו מעודדים אתכם לנסות תחילה לחפש תשובות באינטרנט, כאשר מדובר בשאלות תכנות כלליות.

בהצלחה!

הנחות והנחיות

- פורום הקורס הוא מחייב, תשובות שמתקבלות שם תקפות לכלל הסטודנטים.

- הימנעו משימוש במספרי קסם ומחרוזות מפורשות בקוד עצמו – רכזו את כל זה בקובץ ה-HardCodedData.h שיכיל את כל ה-#define-ים שלכם.

- זכרו לשחרר משאבים כגון: זכרון דינאמי (malloc, HeapAlloc וכו'), Kernel Objects (HANDLE-ים של קבצים, Event-ים וכו') וכל משאב אחר שהתוכנית שלכם תופסת לפני שהיא מסיימת את ריצתה ובפרט במקרה של שגיאה – גם במקרה של שגיאה יש לשחרר את כלל המשאבים בצורה best effort-ית, כלומר אם CloseHandle נכשלת פשוט נמשיך הלאה לנסות לשחרר את שאר המשאבים ולא נסיים בפתאומיות.

- בכל מקרה שלא מוגדרת ההתנהגות הנדרשת מכם אתם רשאים לבחור בכל התנהגות **סבירה**. התנהגות סבירה כוללת בתוכה סיום אלגנטי של התוכנית (לא לקרוס) תוך הדפסת הודעת שגיאה מתאימה ושחרור משאבים. לעניין זה סיום אלגנטי – התוכנית תסיים את ריצתה על ידי ה-return שב-main.

- יש לחלק את הקוד שלכם למודולים ואת המודולים לחלק לפונקציות בצורה הגיונית.

- **חובה** לקרוא על כל אחת מהפונקציות שמתוארות בהמשך באמצעות הקישורים/חיפוש בגוגל לצורך הבנה מיטבית שלהן לתרגיל זה ולטובת התרגילים בהמשך הקורס. שימוש בפונקציה בצורה לא נכונה/לא מוגדרת תגרור הורדת ניקוד גם אם התוכנית כן מתקמפלת ועובדת כראוי.

- ניתן להניח שמספר ה-thread-ים שתתבקשו ליצור לא יעלה על המספר המקסימלי של אובייקטים שניתן להמתין עליהם עם WaitForMultipleObjects.
- אינכם צריכים לבדוק אם מספר התהליכים או ה-thread-ים שנתבקשתם ליצור עובר את המספר המותר על ידי מערכת ההפעלה וניתן להניח שלא תקבלו קלט שעובר גבול זה.

- החלוקה למדרגות היא המלצת עבודה בלבד שנועד לחלק את התרגיל למשמות קטנות יותר. המדרגות נבנות זו על גבי זו ובסוף המדרגה האחרונה הקוד שלכם צריך לממש את כלל דרישות התרגיל. בפרט, ניתן לעבוד בסדר שונה מזה שמוצג במדרגות.

- על הקוד להתקמפל out-of-the-box, כלומר ללא שום התערבות מצד הבודק למעט פתיחת הפרויקט וביצוע Build.

- על הפרויקט להתקמפל ללא שגיאות או אזהרות למיניהן ב-Debug x86.

- יש לתעד ב-Header-ים מעל כל פונקציה מה היא מקבלת, מה היא מחזירה ומה היא עושה וכן לחלק את ה-Header לאזורים (עבור include-ים, עבור #define-ים, עבור פונקציות וכו'). בקבצי ה-Source יש לתעד בגוף הפונקציות על מנת שיהיה קל להבין מה הקוד עושה.

- בפונקציות מיוצאות **יש** לבדוק את תקינות הקלט (לוודא שמצביעים תקינים, גדלים חיוביים וכו') בתחילת הפונקציה. **בפונקציות פנימיות** אין צורך לבצע בדיקות אלו אך מומלץ להשתמש ב-assert-ים (מוגדרים ב-assert.h) על מנת להקל את תהליך הפיתוח והבדיקה.

טיפ של אלופים: מומלץ **מאוד** לעבוד עם שירות Version Control כלשהו למשל כמו GitHub על מנת לשמור את ההתקדמות שלכם וזאת משום שהתרגיל הוא איטרטיבי. ביצוע commit לאחר השלמת מדרגה בתרגיל יעזור לכם לעקוב אחר ההתקדמות שלכם, לזהות בעיות ויאפשר לכם לחזור לנקודה האחרונה שבה הקוד שלכם עבד במקרה שהסתבכתם ואתם לא מצליחים למצוא את הבעיה. למי מכם שמעולם לא השתמש ב-Git ו-GitHub עדיף מאוחר מאשר אף פעם.

רקע

בתרגיל זה תממשו מספר מודולים שמטרתם לאפשר עיבוד של משימות במקביל.

המודולים העיקריים בתרגיל זה הם:

1. מודול Queue לניהול תור משימות.
2. מודול מנעול (Lock) שמאפשר מקבול של קריאה ממשאב אבל מסנכרן כתיבה.

אתם רשאים לכתוב מודולים נוספים אך מודולים אלו **חייבים** להופיע בפרויקט שלכם ולהיות בעלי הפונקציונליות שאנחנו נגדיר.

התוכנית תקבל קובץ משימות, קובץ תיעדוף, מספר המשימות (ששווה למספר השורות בכל קובץ) ומספר ה-thread-ים שיש להשתמש בהם.

כל שורה בקובץ המשימות תכיל משימה אחת. משימה היא מספר כלשהו שיש לפרקו לגורמים ראשוניים. למשל אם בשורה i כתוב 30 אז המשימה היא לפרק את 30 לגורמים ראשוניים (2, 3 ו-5) ולכתוב אותם (לפי פורמט שיוגדר בהמשך) בסוף הקובץ המקורי. לשם כך יהיה עליכם לעטוף את הגישות לקובץ המשימות במנעול שתממשו. כל שורה נגמרת בתו '\n' כולל השורה האחרונה.

קובץ התיעדוף של המשימות יהיה בעל אותו מספר שורות כמו קובץ המשימות.
בשורה ה-i בקובץ התיעדוף יהיה רשום המיקום שבו נמצאת משימה כלשהי בקובץ המשימות.
למשל, אם שורה i בקובץ המשימות מתחילה בהיסט x מתחילת הקובץ והמשימה בשורה ה-i היא המשימה בעלת החשיבות הגבוהה ביותר אז בשורה הראשונה בקובץ התיעדוף יהיה רשום x.

עם תחילת ריצתה של התוכנית ה-thread הראשי יאתחל Queue, יקרא את קובץ התיעדופים ועבור כל שורה בקובץ התיעדוף יכניס איבר ל-Queue שערכו יהיה הערך שבאותה שורה בקובץ התיעדוף. כלומר אם בשורה הראשונה של קובץ התיעדוף רשום 5 אז נכניס את 5 לתור. זה אומר שהמשימה שבמיקום 5 בקובץ המשימות היא המשימה בעלת התיעדוף הגבוה ביותר מבין כל המשימות.

אחר שה-thread הראשי סיים למלא את ה-Queue הוא ייצור את ה-thread-ים. כל thread ייגש בצורה בטוחה ומסונכרנת ל-Queue, יוציא את המשימה שבראש התור, יקרא את השורה שתחילתה במיקום המתאים בקובץ המשימות, יבצע את המשימה (יפרק את המספר שהוא קרא מקובץ המשימות לגורמים ראשוניים) ויכתוב את התוצאה בסוף הקובץ המקורי בפורמט הבא:

The prime factors of {number} are: {prime-1}, {prime-2}, ..., {prime-n}

דוגמה נוספת עבור 30:

The prime factors of 30 are: 2, 3, 5\r\n

שימו לב שהפירוק הוא עם כפילויות, כלומר אם מספר ראשוני כלשהו הוא מופיע כמה פעמים כגורם ראשוני של מספר אז יש להדפיס אותו מספר פעמים, לדוגמה עבור 24:

The prime factors of 24 are: 2, 2, 2, 3\r\n

הקריאה והכתיבה לקובץ צריכה להיות מסונכרנת בעזרת המנעול שכתבתם כדי לאפשר למספר thread-ים לקרוא במקביל משימות שונות ולאפשר רק ל-thread יחיד לכתוב את התוצאה שלו לסוף הקובץ.

על ה-Thread-ים להוציא משימות מהתור ולהשלים אותן עד שהתור מתרוקן. לאחר שהתור יתרוקן על כל ה-thread-ים לשחרר את כלל המשאבים שהם מחזיקים וטרם שוחררו ולסיים את ריצתם. ה-thread הראשי ימתין (פרק זמן מכובד אבל סופי) לסיום העבודה של כל ה-thread-ים ורק אז יסיים את ריצתו שלו ובכך את ריצת התוכנית כולה.

פורמטים

הנחת היסוד היא שהקבצים נכתבו תחת Windows.

קובץ המשימות

משימות

הפורמט של המשימות הוא:

```
{number}\r\n
{number}\r\n
{number}\r\n
```

להלן קובץ עם 3 משימות:

```
25\r\n
13\r\n
1337\r\n
```

לא ניתן להניח דבר על אורך השורות. ניתן להניח שאחרי כל מספר תהיה ירידת שורה **כולל** אחרי המספר האחרון, כלומר השורה האחרונה בקובץ לא תכיל מספר.

פלט

את הפלט יש לכתוב בפורמט הבא לסוף קובץ המשימות:

```
The prime factors of {number} are: {prime-1}, {prime-2}, ..., {prime-n}
The prime factors of {number} are: {prime-1}, {prime-2}, ..., {prime-n}
The prime factors of {number} are: {prime-1}, {prime-2}, ..., {prime-n}
```

כאשר הגורמים הראשוניים מסודרים בסדר עולה, כלומר הגורם הראשון הוא הקטן ביותר והגורם האחרון ברשימה הוא הגדול ביותר. כמו בקובץ המקורי, השורה האחרונה בקובץ תהיה שורה ריקה.

ניקח את הדוגמה של קובץ עם 3 משימות למעלה, לאחר ריצת התוכנית הקובץ ייראה כך:

```
25\r\n
13\r\n
1337\r\n
The prime factors of 25 are: 5, 5\r\n
The prime factors of 13 are: 13\r\n
The prime factors of 1337 are: 7, 191\r\n
```

אין חשיבות לסדר שבו הפלט מודפס, כלומר זה גם היה יכול להיות:

```
25\r\n
13\r\n
1337\r\n
The prime factors of 25 are: 5, 5\r\n
The prime factors of 1337 are: 7, 191\r\n
The prime factors of 13 are: 13\r\n
```

דגשים והערות על הפורמט:

- : שימו לב **שיש** רווח אחרי ה-':' ואחרי כל ','.
- ירידת השורה ב-Windows מורכבת משני תווים – '\r' ו-'\n' ולכן ירידת שורה תופסת שני בתים.
- השורה האחרונה בקובץ תהיה שורה ריקה.

יש להקפיד על הפורמט הנ"ל , סטייה ממנו תגרור הורדת נקודות.

קובץ התיעדופים

קובץ התיעדופים יכיל בכל שורה את המיקום בקובץ המשימות של משימה כלשהי. החשיבות של אותה משימה נקבעת לפי המיקום שלה בקובץ תיעדופים וניתן לומר שהתיעדוף שלה הוא מספר השורה שלה בקובץ התיעדופים, ככל שמספר השורה קטן יותר כך המשימה יותר חשובה ויש לה תיעדוף גבוה יותר.

לאותו קובץ משימות יכולים להיות הרבה תיעדופים שונים, ניקח את קובץ המשימות:

```
25\r\n
13\r\n
1337\r\n
```

וקובץ תיעדופים לדוגמה יהיה:

```
8\r\n
0\r\n
4\r\n
```

גם כאן, כל שורה תכיל מיקום יחיד והשורה האחרונה בקובץ תהיה ריקה. לפי הדוגמה הזאת, המשימה שמתחילה ב-byte ה-8 (לא התו השמיני, זכרו שיש ירידת שורה אחרי כל מספר) היא בעלת התיעדוף הגבוה ביותר, כלומר המשימה 1337, לאחר מכן המשימה 25 ולבסוף המשימה 13.

מודולים

אתם רשאים לחלק את הפרויקט למודולים לבחירתכם. עם זאת עליכם לממש את שני המודולים הבאים:

Queue

מטרת המודול היא לייצג מבנה נתונים FIFO של תור. על מודול לייצא את הפונקציות הבאות:

- InitializeQueue - מחזירה מצביע ל-struct שמייצג את התור.
  כל שאר הפונקציות במודול זה מקבלות מצביע שכזה ומבצעות עליו פעולות.

- Top - מקבלת מצביע ל-struct של תור ומחזירה את האלמנט שבראש התור מבלי להוציא אותו.

- Pop - מקבלת מצביע ל-struct של תור ומוציאה ממנו את האלמנט הראשון, מעדכנת את התור ככה שעכשיו האלמנט הבא בתור הוא האלמנט הראשון.

- Push - מקבלת מצביע ל-struct של תור ואלמנט ומוסיפה אותו לסוף התור.

- Empty - מקבלת מצביע ל-struct של תור ומחזירה אמת אם התור ריק ושקר אם הוא לא ריק.

- DestroyQueue - מקבלת מצביע ל-struct של תור, משחרר את כל המשאבים.
  בסיום הפונקציה צריך לבצע השמה של NULL לערך של המצביע.

המימוש הפנימי של התור עצמו נתון לבחירתכם.

שימו לב: האיברים בתור הם **ההיסטים** של המשימות לפי קובץ התיעדופים **ולא** המשימות עצמן.

Lock

מטרת המודול היא לאפשר למספר thread-ים לקרוא מאותו משאב (קריאה, לענייננו היא כל פעולה שכוללת גישה למשאב בצורה כזו שלא משנה את התוכן או המצב של המשאב) אך מאפשר רק ל-thread אחד לקרוא מהמשאב. על המודול לייצא את הפונקציות הבאות:

- InitializeLock - מחזירה מצביע ל-struct שמייצג את המנעול. כל שאר הפונקציות המיוצאות במודול הזה מקבלות מצביע שכזה ומבצעות עליו פעולות.

- read_lock - מקבלת מצביע ל-struct של מנעול ומבצעת נעילה לקריאה.
  תפיסת המנעול לקריאה חוזרת אם המנעול לא תפוס לכתיבה. אם המנעול תפוס לכתיבה אז היא תמתין עד ל-timeout ותחזור עם כישלון.
  בפרט לתפוס את המנעול לקריאה בזמן שהמנעול נעול לקריאה תמיד חוזרת (זה המקרה של מספר קוראים במקביל).

- read_release - מקבלת מצביע ל-struct של מנעול ומשחררת המנעול מתפיסה לקריאה שנעשתה על ידי אותו ה-thread.
  שימו לב שכל קורא שמסיים לקרוא צריך לשחרר את המנעול מתפיסה לצורך קריאה ורק אחרי שכל מי שתפס את המנעול לצורך קריאה שחרר המנעול באמת ישתחרר.
  לדוגמה: אם thread A ו-thread B שניהם תפסו את המנעול לצורך לקריאה ואז thread B סיים לקרוא ומשחרר את התפיסה שלו עדיין A תופס את המנעול לצורך קריאה ולכן אם thread C ינסה לתפוס את המנעול לצורך כתיבה הוא ייאלץ להמתין עד שגם A ישחרר את התפיסה שלו.

- write_lock - מקבלת מצביע ל-struct של מנעול ונועלת את המנעול לכתיבה.
  עילה לכתיבה חוזרת אם ורק אם המנעול לא תפוס לכתיבה ואין אף אחד שתופס אותו לקריאה.
  שימו לב שבאופן טבעי נעילה לכתיבה מכילה בתוכה גם נעילה לקריאה (thread שנעל לכתיבה יכול גם לקרוא וגם לכתוב וזה תקין וחוקי).
- write_release - מקבלת מצביע ל-struct של מנעול ומשחררת את המנעול מהתפיסה לכתיבה. שימו לב שרק ה-thread שתפס את המנעול לכתיבה יכול לשחרר אותו.

- DestroyLock - מקבלת מצביע ל-struct של מנעול ומשחררת את כלל המשאבים שלו.
  בסיום הפונקציה צריך לבצע השמה של NULL לערך של המצביע.

הערות:

- המימוש הפנימי נתון לבחירתכם. מומלץ להשתמש בשילוב כלשהו של Event, Mutex, Semaphore.
- אפשר לממש את הפונקציות הנ"ל בצורה כזו שמקבלת מבחוץ את ה-timeout-ים עבור כל סוג של נעילה. בכל מקרה ה-timeout של פונקציות ה-lock יהיה סופי וגודלו ייבחר בהתאם לזמן שלוקח לפרק מספרים בתחום שייבדקו (ראה הערות בחלק הבא).

פירוק לגורמים ראשוניים

לנוחיותכם פסודוקוד למציאת הגורמים הראשוניים של מספר, הוא לא בהכרח האלגוריתם הכי יעיל למשימה (באופן כללי זו היא בעיה קשה שאין לה פתרון יעיל למקרה הכללי). אתם רשאים לחפש פתרונות יעילים יותר.

קלט: מספר n

פלט: רשימת הגורמים הראשוניים שלו עם חזרות $\{p_1, p_2, \dots, p_k\}$.

האלגוריתם:

1. INITIALIZE $P \leftarrow \phi$
2. WHILE n (mod 2) = 0 DO:
   2.1. $n \leftarrow \frac{n}{2}$
   2.2. $P \leftarrow P \uplus \{2\}$
3. INITIALIZE $i \leftarrow 3$
4. WHILE $i \leq \sqrt{n}$ DO:
   4.1. WHILE $i\ (mod\ n) = 0$ DO:
      4.1.1. $n \leftarrow \frac{n}{i}$
      4.1.2. $P \leftarrow P \uplus \{i\}$
   4.2. $i \leftarrow i + 2$
5. IF $n > 2$ DO:
   5.1. $P \leftarrow P \uplus \{n\}$

הערה: האופרטור $\uplus$ הוא איחוד זר שנועד לאפשר ל-P להכיל כפילויות.

הסבר: 2 הוא ראשוני, לכן כל עוד n הוא זוגי נחלק ב-2 ונוסיף 2 לרשימת הגורמים הראשוניים עד שנגיע למספר א"ז. לאחר מכן, נתחיל מ-$i = 3$ ונתקדם עד ל-$\sqrt{n}$ (לא יכול להיות גורם של n שגדול מ-$\sqrt{n}$), עבור כל i נבדוק אם הוא גורם של n הנוכחי ואם כן נוסיף אותו לרשימת הגורמים. אם אי אפשר לחלק את n ב-i אז i לא גורם של n. נקדם את i ב-2 (מספר ראשוני שגדול מ-2 הוא בהכרח א"ז) ונמשיך. אם הגענו ל-$n = 1$ סיימנו ומצאנו את כל הגורמים הראשוניים של n. אם לא הגענו ל-$n = 1$ אז סימן ש-n והוא גדול מ-2 אז זה סימן לכך ש-n הנוכחי עצמו ראשוני ולכן נוסיף את n לרשימת הגורמים הראשוניים.

השלב הראשון נפטר למעשה בכל ה-2-ים שבמספר המקורי.

בשלב השני מספיק לבדוק את כל המספרים עד השורש של n העדכני לפי תכונה של מספרים פריקים כל מספר פריק הוא בעל לפחות גורם ראשוני אחד שקטן מהשורש שלו. נניח בשלילה שזה לא כך, יהיו a, b שני גורמים של n כך ש-$a \cdot b = n$. אזי אם שניהם גדולים מ-$\sqrt{n}$ נקבל ש-$a \cdot b >$ $\sqrt{n} \cdot \sqrt{n} = n$ בסתירה לכך ש-a ו-b הם גורמים של n.

לכאורה האלגוריתם עובר על כל המספרים האי-זוגיים אבל בפועל זה לא יקרה. נדגים זאת, נניח שהמספר שלנו n מתחלק ב-15, 2 לא גורם ראשוני ולכן ישר נתחיל לבדוק את 3 ונחלק ונקבל $\frac{n}{3}$ ונמשיך כך שהוא כבר לא התחלק ב-3. מכיוון ש-n המקורי התחלק ב-15 ה-n החדש בהכרח לא מתחלק ב-15 כי החלוקה ב-3 מסירה את כל הכפילויות של 3 ובתוך כל 15 יש גורם ראשוני של 3. לכן כאשר האלגוריתם יגיע ל-$i = 15$ ה-n העדכני כבר לא יתחלק ב-15.

הערות:

- ניתן להניח שהמספרים לפירוק בתרגיל יהיו בעלי 9 ספרות לכל היותר.כלומר בתחום 1- 999,999,999. יש לבחור את זמני ה-*timeout*-ים בהתאם לכך.

מדרגות

החלוקה למדרגות איננה חובה ונועדה לעזור לכם לחלק את העבודה שלכם.

מדרגה 1

פתחו פרוייקט חדש ב-Visual Studio בשם Factori. הפרויקט יהיה executable כלומר, ביצוע build ייצור קובץ exe הניתן להרצה.

כתבו פונקציה שיודעת לפרק מספר לגורמים ראשוניים, פונקציה שיודעת למיין את הגורמים האלה ופונקציה שיודעת לפרמט את התוצאה למחרוזת לפי הפורמט שהוגדר.

כתבו main שיקבל את הארגומנטים שפורטו קודם לכן ובשלב זה הסתפקו בליצור thread יחיד. ה-thread הראשי יקרא משימה אחת בכל פעם מקובץ המשימות לפי הסדר שמגדיר קובץ התיעדופים ועבור כל משימה ייצור thread שיבצע אותה, ימתין לסיום ורק אז יעבור למשימה הבאה.

מדרגה 2

כתבו את מודול התור. שימו לב, האיברים בתור הם **ההיסטים** של המשימות לפי קובץ התיעדופים **ולא** המשימות עצמן.

כעת ה-thread הראשי יקרא את קובץ התיעדופים וימלא בהתאם את התור. כל אלמנט בתור יהיה ההיסט של משימה בקובץ המשימות.

מדרגה 3

כתבו את מודול המנעול.

כעת, עדיין עם thread יחיד בנוסף ל-thread הראשי, בדקו שהמודול שלכם עובד.
לאחר שהצלחתם, נסו עם שני thread-ים.

אל תשכחו לעטוף את הגישות לקבצים ולתור בעזרת המנעול שלכם עם הנעילה הנכונה (לקריאה/כתיבה) בהתאם לסוג הפעולות שאתם עושים.

אל תשכחו גם לשחרר את הנעילה אחרי שסיימתם וגם במקרה של כישלון אל תשכחו לשחרר את הנעילה כדי שלא תגיעו למצב של deadlock.

מדרגה 4

בדקו את עצמכם על הקובץ לדוגמה וצרו קבצים נוספים כדי לבדוק את עצמכם.
לשם כך אפשר ואפילו מומלץ לכתוב קוד על מנת לייצר צמדי קבצים של משימות ותיעדוף.

בונוס (10 נקודות)

כעת, במקום שרק ה-thread הראשי יעבוד קשה בהתחלה וימלא את התור לבד כל ה-thread-ים צריכים ביחד למלא את התור על ידי קריאה במקביל של קובץ התיעדופים. שימו לב, יש צורך לחשוב על אסטרטגיית קריאה ככה שה-thread-ים השונים לא יקראו את אותן השורות. ניקוד מלא יינתן על מימוש יעיל שעושה שימוש מועט ככל הניתן במנגנוני סנכרון.

הערות

- אין צורך לדאוג יותר מדי מיעילות, לא ניתן מספרים גדולים מדי שקשה לפרק. עם זאת על רשימה של כמה עשרות מספרים של עד אלפים ועשרות אלפים הקוד שלכם צריך לסיים בזמן סביר (**לכל היותר** 2-1 דקות). על כן זמני ה-timeout צריכים להיות בהתאמה אך עליהם להיות **סופיים** ולא INFINITE.
- מומלץ מאוד לעבוד עם קבצים כמו שעבדתם איתם בתרגיל בית 2, בעזרת הפונקציות ReadFile, WriteFile, CreateFile, SetFilePointer, etc.
- אין להשאיר deadlock-ים בקוד גם אם הם קורים בהסתברות נמוכה. הקוד תמיד צריך לסיים וה-race-ים היחידים צריכים להיות אל נעילה של המנעול ולא לגישה למשאבים (בהגדרה ניסיון לנעול מעול מכמה thread-ים הוא race).