תרגיל בית 3

<u>מועד הגשת התרגיל</u>: עד יום 01.01.2017 בשעה 17:59:59. לא יהיו דחיות.

<u>מטרות התרגיל</u>:

- העמקת ההבנה במושגי החוט (Thread) במערכת הפעלה בכלל וב-Windows בפרט.
- עבודה עם מספר רב של חוטים במקביל.
- שימוש ב-Mutex ו-Semaphore לסנכרון גישה לזיכרון משותף בין חוטים.
- הימנעות מ-deadlock-ים.

<u>מבוא</u>:

בתרגיל זה, תכתבו תכנית שמבצעת פעולות בנקאיות: הפקדה ומשיכה, על מספר חשבונות בנק. בתרגיל זה, נממש סנכרון של הגישה לזיכרון המשותף בין החוטים מכיוון שפעולות ההפקדה והמשיכה לאותו החשבון לא יכולות להתבצע במקביל. את הסנכרון נממש תוך הימנעות מ-deadlock-ים.

<u>מימוש</u>:

עליכם לכתוב תכנית לניהול חשבונות בנק התומכת בפועלות הבאות:

- פעולות ניהול חשבון:
  1. פתיחת חשבון
  2. סגירת חשבון
  3. הדפסת יתרות
- פעולות בנקאיות:
  1. הפקדה
  2. משיכה

<u>הערה</u>: הפעולות הבנקאיות יכולות להתבצע רק על חשבונות בבנק הקיימים במסד הנתונים.

התכנית תקבל את סט הפקודות להרצה כקלט. התכנית תריץ כל פעולה כחוט נפרד.
התכנית תסיים לאחר שכל הפעולות יסיימו את ביצוען ויבוצעו הדפסות הסיום הנדרשות כפי שיופרט בהמשך. **אין צורך שכל חשבונות הבנק יסגרו על מנת שהתכנית תסיים.**
חשוב לציין, כי אין להניח שידוע חסם כלשהו על גודל הזיכרון הדרוש לשמירת המידע על חשבונות הבנק. הסכומים צריכים לתמוך בסדרי גודל של טריליונים ועד רמת האגורה (אין חלקי אגורה).

<u>שורת הרצה</u>:

התוכנית תרוץ ע"י שורת פקודה עם הארגומנטים הבאים:

<Command File> <Balance Report File> <Runtime Log File>

- Command File – זהו קובץ שמכיל את כל הפעולות הבנקאיות שעל התכנית לבצע. זהו ארגומנט שחובה לספקו לתכנית על מנת שתרוץ.
- Balance Report File - קובץ הפלט של התוכנית, תוכנו יוסבר בחלק הפלט.
- Runtime Log File – קובץ הלוג (תיעוד של התכנית), תוכנו יופרט בהמשך.

ניתן להניח שכל הקבצים יהיו בתיקייה הנוכחית ואין צורך לתמוך ב-Full path.
דוגמא לשורת הרצה:

Hw3.exe command_file.cmd Account_Balance.rpt runtime_log.log

שימו לב, שהשם של התכנית בדוגמא, הוא להמחשה בלבד ואתה יכולים להשתמש בשם אחר.

<u>קלט</u>:

<u>תוכן Command File</u>:

התכנית תקבל קובץ המכיל את הפעולות הבנקאיות שעל התכנית לבצע. על התכנית לתמוך בפעולות הבאות:

- CreateAccount <account number> <current balance>

הפקודה יוצרת חשבון בנק חדש. השדה account number מכיל את מספר החשבון החדש. מספר החשבון יורכב מרצף של ספרות 0-9. השדה current balance מגדיר את היתרה בנוכחית בחשבון. שימו לב, current balance יכול להיות מספר שלילי. לפני ביצוע פקודה זו, **יש לוודא שכל הפקודות שהיו כתובות לפניה בקובץ הקלט סיימו את ריצתן**. כמו-כן, רק לאחר שפקודה זו מסיימת את ריצתה, אפשר להמשיך לקרוא פקודות מקובץ הקלט.
בסיום יצירת החשבון, התכנית תדפיס לקובץ הלוג את ההודעה הבאה:

Successfully created bank account number <account number> with current balance of <current balance>.

אם כבר קיים חשבון בנק מספר account number, על התוכנית להדפיס את הודעת השגיאה הבאה לקובץ הלוג:

!!! Account number <account number> already exists. Can't create account. Skipping command. !!!

- CloseAccount <account number>

הפקודה מוחקת חשבון בנק ממסד הנתונים. השדה account number מכיל את מספר החשבון שיש למחוק. לפני ביצוע פקודה זו, **יש לוודא שכל הפקודות שהיו כתובות לפניה בקובץ הקלט סיימו את ריצתן**. כמו-כן, רק לאחר שפקודה זו מסיימת את ריצתה, אפשר להמשיך לקרוא פקודות מקובץ הקלט.
בסיום סגירה מוצלחת של החשבון, התכנית תדפיס לקובץ הלוג את ההודעה הבאה:

Successfully closed bank account number <account number>.

אם חשבון בנק מספר account number, לא קיים במסד הנתונים על התוכנית להדפיס את הודעת השגיאה הבאה לקובץ הלוג:

!!! Account number <account number> doesn't exist. Can't close account. Skipping command. !!!

- PrintBalances

הפקודה מדפיסה את היתרות בכל החשבונות הפעילים במסד הנתונים. לפני ביצוע פקודה זו, **יש לוודא שכל הפקודות שהיו כתובות לפניה בקובץ הקלט סיימו את ריצתן**. כמו-כן, רק לאחר שפקודה זו מסיימת את ריצתה, אפשר להמשיך לקרוא פקודות מקובץ הקלט.
יש להדפיס את הנתונים בפורמט הבא:

Current balances in bank accounts are:
Bank Account #,Current Balance
XXXXXX,YYYYYY

<u>הערות לגבי PrintBalances</u>:

1. *יש למיין את המידע בסדר עולה לפי מספר חשבון הבנק.*
2. *במידה ואין חשבונות בנק פעילים במסד הנתונים, יש להדפיס את שתי השורות הראשונות בלבד.*

<u>הערות לגבי פעולות ניהול חשבון:</u>

1. *שימו לב, שיש צורך במנגנון סנכרון.*
2. *על מנגנון הסנכרון לבצע את הפעולות הבאות:*
   a. *הפסקת קריאת הפקודות מהקובץ עד לסיום הפעולה.*
   b. *המתנה לסיום החוטים שכבר רצים ו/או ממתינים לריצה לפי ביצוע הפעולה. יש להשתמש ב-WaitForMultipleObjects על כל החוטים, פרט לחוט הפקודה הנוכחית (פתיחה, סגירה, או הדפסה).*

- Deposit <account number> <amount>

הפקודה מפקידה את הסכום (amount) לחשבון מספר account number.

ניתן להניח, כי סכומי ההפקדות יהיו חיוביים בלבד.

בסיום ההפקדה מוצלחת, התכנית תדפיס את ההודעה הבאה:

Successfully deposited <amount> to account number <account number>.

במידה וחשבון הבנק לא קיים יש להדפיס את ההודעה הבאה:

!!! Unable to deposited <amount> to account number <account number>. Account doesn't exist. Skipping command. !!!

- Withdrawal <account number> <amount>

הפקודה מושכת את הסכום (amount) מחשבון מספר account number.

ניתן להניח, כי סכומי המשיכות יהיו חיוביים בלבד.

בסיום משיכה מוצלחת, התכנית תדפיס את ההודעה הבאה:

Successfully withdrew <amount> from account number <account number>.

במידה וחשבון הבנק לא קיים יש להדפיס את ההודעה הבאה:

!!! Unable to withdrew <amount> from account number <account number>. Account doesn't exist. Skipping command. !!!

<u>הערות לגבי פעולות בנקאיות:</u>

1. *שימו לב, שיש צורך במנגנון סנכרון.*
2. *במהלך פקודות אלו, יש להגן ב-mutex על אותו חשבון הבנק אותו רוצים לעדכן, כך שפעולות משיכה ו/או הפקדה נוספות לאותו החשבון לא יוכלו להתרחש במקביל.*

דוגמא לקובץ Commands File:

```
CreateAccount 12345 10
CreateAccount 11111 55
CreateAccount 11111 100
Deposit 12345 1000
Withdraw 12345 50
PrintBalances
CloseAccount 12345
CloseAccount 12345
CloseAccount 11111
```

פלט:

- Runtime Log File:

קובץ זה ירכז את תיעוד הריצה של התוכנית. אליו יש לכתוב את כל הודעות של הפעולות ניהול חשבונות הבנק והודעות נוספות שאתם מוציאים במהלך התוכנית (במיוחד הודעות שגיאה).

בסיום התכנית יש לכתוב את ההודעה הבאה:

Program successfully finished running. Exiting.

דוגמא לקובץ Runtime Log File:

```
Successfully created bank account number 12345 with current balance of 10.
Successfully created bank account number 11111 with current balance of 55.
!!! Account number 11111 already exists. Can't create account. Skipping command. !!!
Successfully deposited 1000 to account number 12345.
Successfully withdrew 50 from account number 12345.
Current balances in bank accounts are:
Bank Account #, Current Balance
11111,55
12345,960
Successfully closed bank account number 12345.
!!! Account number 12345 doesn't exist. Can't close account. Skipping command. !!!
Successfully closed bank account number 1111.
Program successfully finished running. Exiting.
```

- Balance Report File:

זהו קובץ הסיכום של התכנית. יש לכתוב אליו את התוכן בסיום התוכנית לפני הפורמט הבא:

Summary of balances in bank accounts:
Bank Account #,Current Balance,Initial Balance,Total Deposited,Total Withdrawal,# of Deposits,# of Withdrawals
XX,YY,ZZ,AA,BB,CC,DD

הערות:

1. *יש למיין את המידע בסדר עולה לפי מספר חשבון הבנק.*
2. *במידה אין חשבונות בנק פעילים במסד הנתונים, יש להדפיס את שתי השורות הראשונות בלבד.*

דוגמא לקובץ Balance Report File:

```
Summary of balances in bank accounts:
Bank Account #,Current Balance,Initial Balance,Total Deposited,Total Withdrawal,# of Deposits,# of Withdrawals
11111,55,55,0,0,0,0
12345,960,10,1000,50,1,1
```

הערה: הדוגמא הזו לא ממשיכה במדויק את התרחיש מהעמודים הקודמים (כי בתיאור שם החשבונות נסגרו). כאן מודגם פורמט הפלט כאשר שני החשבונות נותרו פתוחים.

סנכרון:

בתרגיל זה יידרשו מנגנוני סנכרון בין החוטים השונים. משאב דורש סנכרון אם מתקיים עבורו שבאותה נקודת זמן שבה אחד החוטים/תהליכים כותב לתוכו, קיים לפחות גורם נוסף שמבצע קריאה או כתיבה אל אותו המשאב.

בשלב הראשון חשבו איזה מבני נתונים דרושים על מנת ולממש את דרישות התרגיל.

להלן הנחיות לעבודה עם סנכרון. חשבו על הנחיות האלה כשאתם מתכננים באילו מבני נתונים להשתמש על מנת ולפתור את התרגיל.

בשלב השני, תצטרכו לתרגם את ההנחיות הנ"ל, לתרחישים השונים שבהם צריך בקוד שלכם סנכרון, אלה יתרגמו לאזורים קריטיים בקוד שלכם.

חשבו אלו בקרות אתם צריכים לממש על מנת ולמנוע גישה מקבילית למשאב משותף.

לאחר מכן, צריך לחשוב אלו תרחישים עלולים לגרום ל-deadlockים או חמור מכך לגישה בו זמנית לאזורים קריטיים בקוד.

מומלץ להשקיע זמן בתכנון הסנכרון של התכנית. זה יחסוך לכם זמן בפתרון התרגיל.

<u>עליכם לדאוג שהתכנות המקבילי יתנהג בצורה הבאה:</u>

1. יש לוודא שכל החוטים עובדים במקביל.
2. אל תשמשו ב-mutex יחיד כדי להגן על מבנה נתונים גדול מדי.
3. יש להימנע מ-deadlock-ים.
4. אין לסיים חוטים באזורים הקריטיים, כלומר יש לצאת מהקטע הקריטי לפני סיום ריצת החוט במידה וצריך.

<u>הסנכרון צריך להתבצע כך ש:</u>

1. כל חשבון בנק שאותו אתם מנהלים צריך להיות מוגן ב-mutex, כדי לאפשר רק לחוט אחד לשנות אותו בכל רגע נתון.
2. עבור פקודות ניהול חשבון: פתיחת חשבון, סגירת חשבון וההדפסה (PrintBalances), יש לבצע שתי פעולות:
   i. הפסקת קריאת הפקודות מהקובץ עד לסיום הפעולה.
   ii. המתנה לסיום החוטים שכבר רצים ו/או ממתינים לריצה לפי ביצוע הפעולה. יש להשתמש ב-WaitForMultipleObjects על כל החוטים, פרט לחוט הפקודה הנוכחית (פתיחה, סגירה, או הדפסה).
3. פעולות בנקאיות, כלומר משיכה או הפקדה, צריכות להתבצע כל הניתן במקביל. שימו לב, שיש במהלך פקודות אלו, יש להגן ב-mutex על אותו חשבון הבנק אותו רוצים לעדכן, כך שפעולות משיכה ו/או הפקדה נוספות לאותו החשבון לא יוכלו להתרחש במקביל.
4. כתיבה לקובץ הלוג, Runtime Log File, צריכה להיות מוגנת במנעול, כדי לאפשר רק לחוט אחד לכתוב לקובץ בכל רגע נתון. שאר ההדפסות צריכות להמתין עד שיתאפשר להן גישה לקובץ.

התמודדות עם שגיאות:

יש לבדוק את הצלחה של כל פונקציה שעשויה להיכשל (הקצאת, זיכרון, פתיחת קבצים, יצירת חוט וכו׳). במידה שמתרחשת שגיאה, יש לסיים את התוכנית באופן מסודר:

1. יש לשחרר/לסגור בצורה מסודרת את כל המשאבים שהוקצו: זיכרון, חוטים, קבצים וכו׳.
2. יש להדפיס הודעת שגיאה למסך ולקובץ הלוג.
3. יש להשאיר את קבצי הפלט כפי שהם ולהוסיף בהם הודעת שגיאה.
4. התמודדות עם שגיאה שמתרחשת באחד החוטים:

- יש לסיים בהקדם האפשרי את התכנית באופן נקי ומסודר. כלומר, על כל החוטים לשחרר את כל המשאבים שברשותם ולהסתיים. על מנת ולממש זאת, יש לממש מנגנון שיאפשר לחוט לדעת שיש בעיה בחוט אחר ושעליו להסתיים.
- יש ליידע את החוט הראשי באמצעות שימוש בערך היציאה (exit code) של החוט שבו התרחשה השגיאה.
- יש לסיים את התכנית בצורה מסודרת ולהוציא הודעת שגיאה.

דוגמא למנגנון שמודיע לחוטים שעליהם להסתיים הוא:
Manual-Reset Event גלובאלי, בשם AllThreadsMustEnd שמאותחל למצב non-signaled.
אם חוט מזהה שנוצר מצב שמחייב את סגירתו, הוא משנה את הערך של ה-Event למצב signaled. בקוד של כל אחד מהחוטים, ערכו של ה-event נבדק לעיתים קרובות, באופן הבא:
WaitForSingleObject(AllThreadsMustEnd,0)
יש לשים לב, שה-timeout הוא אפס ולכן הפונקציה WaitForSingleObject מסתיימת מיד ואין המתנה. לפי הערך המוחזר ניתן לדעת אם ה-event במצב signaled או לא. אם מתברר שכן, החוט מסתיים מיד בצורה מסודרת.
יש לדאוג להימנע ממצב של deadlock כתוצאה מהסיום הפתאומי. יש לתכנן את מנגנוני הסנכרון להימנע ממצב זה.

<u>הנחיות נוספות:</u>
אין דרך אחת נכונה לפתור את התרגיל והתרגיל לא כוון לפתרון ספציפי.
הקפידו על ביצוע design לתוכנית לפני שאתם ניגשים לכתוב אותה. חישבו אילו פונקציות עזר אתם צריכים לפני שאתם ניגשים לממש אותן. זכרו כי כל קטע קוד שאתם משתמשים בו יותר מפעם אחת, צריך להיכתב כפונקציה נפרדת. כאשר פונקציה נעשית גדולה ומסובכת, פצלו אותה למספר פונקציות.
כתבו קוד קריא ותעדו אותו.
השתמשו בזיכרון דינמי לאחסן מידע שגודלו אינו ידוע בזמן הקומפילציה. אינכם רשאים להניח חסם עליון שרירותי לגודל המידע.
השתמשו בקבועים ושימו לב לשחרור זיכרון דינמי.
אתחלו את כל הפוינטרים ל-NULL.
אל תחכו עד שכל התכנית תהיה כתובה כדי לקמפל ולהריץ בדיקות. קמפלו ובדקו כל יחידה בנפרד (יחידה יכולה להיות פונקציה משמעותית, מודול, קבוצת מודולים וכו׳).
לפני שאתם משתמשים בפקודת API בפעם הראשונה, רצוי לבדוק את דף הMSDN שלה ע"מ להבין מה היא עושה ומה הפרמטרים שהיא מקבלת ומוציאה. אתם **חייבים** לבדוק בתוכנית את הערך שהיא מחזירה, ברגע שהיא מסתיימת ולפעול בהתאם. באופן כללי, רצוי גם לקרוא את הפונקציות שמופיעות תחת Related Functions ב-MSDN.

<u>טיפים:</u>
- חשבו על מנגנוני הסנכרון הדרושים לכם ואיך לבצע אותם כך שמצד אחד לא היווצרו צווארי בקבוק ומצד שני התכנית לא תהיה מבוזרת מידי. באופן כללי, יש להשתמש במנגנוני סנכרון, כאשר יש משאב משותף לשני חוטים (למעט כאשר המשאב המשותף משמש לקריאות בלבד).
- ודאו שאתם משחררים מנעולים כדי למנוע מצבי deadlock.

- מומלץ להשתמש בבלוקים של finally-try בעת הגישה למנעול, כך שבבלוק ה-try נבצע את הבקשה לנעילה ואת הקטע הקריטי ובבלוק ה-finally, נבצע את שחרור הנעילה. כך נוכל לבדוק שאנחנו משחררים את הנעילה כמעט בכל מקרה.
- זכרו להצהיר משתנים משותפים מסוג volatile.
- מטרת התרגיל הינה לתרגל סנכרון וזה הקושי בו. מומלץ להתחיל בריצות הניסיון הראשונות לרוץ בלי חוטים בכלל (כלומר לרוץ עם קלטים שלא מצריכים גישה למשאבים משותפים). לאחר מכן לרוץ עם מספר מינימלי של חוטים על מנת ולבדוק שהסנכרון עובד ורק לאחר מכן להריץ עם מספר רב של חוטים. כמו-כן, מומלץ להריץ את התכנית בשלב הדיבוג עם קלטים מנוונים, כאלה שיפשטו את הריצה, לדוגמא, כמו גודל הקלט, גודל סדרת חשבונות הבנק. המטרה היא לפשט את סביבת העבודה על מנת ולהקל על תהליכי הדיבוג.

בהצלחה!