ב"ה

תרגיל מס' 4 – תכנות מרובה חוטים

הוראות הגשה

- שאלות בנוגע לתרגיל יש לשלוח בפורום הייעודי במודל.
- **מועד אחרון להגשה: ~~6/6/21~~ 13/6/21, 23:59.**
- במידה ויהיו עדכונים בנוגע לתאריך ההגשה, הם יפורסמו בלוח ההודעות במודל.
- יש לשלוח את הקבצים באמצעות האתר:
  https://submit.cs.biu.ac.il/cgi-bin/welcome.cgi
  לפני חלוף התאריך הנקוב לעיל.
- שם ההגשה של תרגיל 4: ex4
- **להזכירכם, העבודה הינה אישית. עבודה משותפת דינה כהעתקה.**
- אין להדפיס דבר מעבר למה שנתבקש בתרגיל.
- יש לוודא שהתרגיל מתקמפל ורץ על שרת ה-planet ללא שגיאות/אזהרות.
- **בראשי הקבצים שאותם אתם מגישים, יש לכתוב שם מלא ות.ז. בפורמט הבא:**
  // Israel Israeli 123456789
- יש להגיש קוד מתועד (באנגלית בלבד). אין לכלול תווים בעברית בקובץ הקוד.
- מענה לשאלות בנוגע לתרגיל יינתן עד התאריך המצוין מעלה, בפורום הייעודי במודל.
- שימו לב להערות בסוף התרגיל.

תכנות מרובה חוטים

הנחיות עבור ex4

- שם התרגיל: ex4
- שמות הקבצים שיש לשלוח: **threadPool.h, threadPool.c**

הקדמה

בתרגיל זה תממשו גרסה פשוטה של thread pool.

מתוך ויקיפדיה:

"A thread pool is a design pattern where a number of threads are created to perform a number of tasks, which are usually organized in a queue. Typically, there are many more tasks than threads. As soon as a thread completes its task, it will request the next task from the queue until all tasks have been completed. The thread will then sleep until there are new tasks available."

ה-thread pool שלכם ייווצר עם N threads – על מנת שיוכל לטפל לכל היותר ב-N משימות בו זמנית. הפונקציה tpInsertTask() אחראית על הכנסת משימה חדשה לתוך תור משימות בתוך ה-thread pool.

משימה שהוכנסה (enqueue) לתור תישאר שם עד שאחד מה-threads יוציא (dequeue) אותה על מנת לטפל בה.

במידה ולא קיים thread פנוי שיבצע אותה, המשימה תתבצע רק לאחר שאחד מה-threads יתפנה.

במידה וקיים thread פנוי הוא כמובן יבצע אותה.

אם thread מתוך ה-thread pool מסיים לבצע את משימתו, ואין משימות שהוכנסו לתור, הוא ימתין (ללא busy waiting!) עד שמשימה חדשה תוכנס לתור.

מאחר וזה אינו קורס במבני נתונים, מצ"ב לתרגיל זה הקבצים osqueue.h, osqueue.c ובהם מימוש של תור (queue) –ואתם יכולים להיעזר בהם לצורך מימוש התרגיל. אין להגיש קבצים אלו.

ממשק

ה-thread pool שלכם צריך לתמוך בפונקציות הבאות, המוגדרות בקובץ threadPool.h:

1. **ThreadPool* tpCreate(int numOfThreads);**

   יוצרת thread pool חדש.

   מקבלת כפרמטר את מספר החוטים שיהיו ב-thread pool ומחזירה מצביע למבנה מסוג ThreadPool, שיועבר לכל שאר הפונקציות המטפלות ב-thread pool.

2. **void tpDestroy(ThreadPool* threadPool, int shouldWaitForTasks);**

   הורסת את ה-thread pool ומשחררת זיכרון שהוקצה.

   ברגע שהפונקציה הזו מתבצעת, לא ניתן להקצות יותר משימות ל-thread pool. תוצאת קריאה לפונקציה tpDestroy לאחר שה-thread pool כבר נהרס אינה מוגדרת (ולא תיבדק).

   הפונקציה מקבלת כפרמטר מצביע ל-thread pool ומספר shouldWaitForTasks. אם המספר שונה מאפס, יש לחכות ראשית שכל המשימות יסתיימו לרוץ (גם אלה שכבר רצות וגם אלה שנמצאות בתוך התור בזמן הקריאה לפונקציה. לא ניתן להכניס משימות חדשות לאחר הקריאה

לפונקציה) ורק לאחר מכן לחזור. אם המספר שווה לאפס, יש לחכות רק לסיום המשימות שכבר רצות, מבלי להתחיל משימות שכבר נמצאות בתוך התור.

3. **int tpInsertTask(ThreadPool* threadPool, void (*computeFunc) (void *), void* param);**

מכניסה משימה לתור המשימות של ה-thread pool.
מקבלת כפרמטרים את ה-thread pool, פונקציה computeFunc שתורץ ע"י המשימה, ו-param, פרמטר עבור computeFunc.
הפונקציה תחזיר 0 במקרה של הצלחה, ו-1- במידה והפונקציה tpDestroy בדיוק נקראת עבור ה-thread pool.

<u>התוכנית</u>

עליכם לממש את הממשק שתואר מעלה בתוך קובץ בשם threadPool.c, כך שיהיה לכם thread pool עובד. תצטרכו גם, ככל שתמצאו לנכון, להוסיף שדות (fields) עבור המבנה thread_pool.
כאמור, מדובר בפיתוח של ממשק, כך שהקוד שלכם לא יכיל פונקציית main.

<u>דגשים חשובים</u>

1. על כל הפונקציות שלכם להיות thread safe – כלומר אם מספר threads (מחוץ ל- thread pool) קוראים במקביל לאותה פונקציה מהממשק שלכם, כל הפעולות יצליחו. חוץ ממקרה בו ה-thread pool כבר נהרס, שבו ההתנהגות לא מוגדרת ולכן לא נבדוק מקרה זה.
2. שימו לב ש-tpDestroy יכולה לקחת זמן רב לסיום במקרה והמשימות שצריכות להסתיים קודם הן ארוכות. לכן, בזמן ש-tpDestroy מחכה לסיום המשימות הללו, חשוב:
   א. לא לאפשר למשימות חדשות להיכנס לתור המשימות (אחרת זה יגרום ל-tpDestroy לרוץ לנצח!)
   ב. לא לאפשר לאף thread אחר לקרוא שוב ל-tpDestroy.
   ניתן להניח שבמידה והפונקציה tpInsert תורץ בבדיקה לאחר קריאה ל-tpDestroy (עבור אותו thread pool), זה יהיה רק בזמן ש-tpDestroy מחכה למשימות שיסתיימו ולא בשלב מאוחר יותר של הריסת ה-thread pool לאחר שכל המשימות הסתיימו.
3. תצטרכו להיעזר במצביעים לפונקציות, לדוגמא כדי לשמור אותן בתוך תור המשימות לצורך ביצוע עתידי ע"י thread מתוך ה-thread pool.
4. **הדברים היחידים שאתם יכולים לשנות בקובץ threadPool.h:**
   א. להוסיף #includes במידת הצורך
   ב. להוסיף שדות בתוך struct thread_pool
   ג. להוסיף type definitions משלכם (structs, enums וכו')
5. אין לשנות את חתימת הפונקציות ב-threadPool.h!
6. **אין לשנות את הקבצים osqueue.h, osqueue.c. בנוסף, כאמור, אין להגיש אותם.**
7. יש להשתמש במנגנוני סינכרוניזציה ולהימנע מ-busy waiting.
   קיימות מספר פונקציות העשויות לעזור לכם:
   pthread_cond_init
   pthread_cond_wait
   pthread_cond_destroy

pthread_cond_signal
pthread_cond_broadcast
מומלץ להיעזר ב-man כדי לקבל מידע על אופן השימוש בפונקציות אלו. בנוסף, להלן קישור המכיל הסבר על הפונקציות (אפשר להיעזר גם במקורות אחרים ברשת)
https://docs.oracle.com/cd/E19455-01/806-5257/6je9h032r/index.html#sync-44265

8. הקפידו להגדיר critical sections קטנים ככל האפשר. אתם יכולים לנעול את כל התור במקרה של קריאה ל-enqueuer או dequeuer.
9. בדקו את התוכנית שלכם עם multiple threads.

10. **Testing** – מכיוון שבתרגיל זה תפתחו ממשק, כנראה שתרצו לבדוק את הקוד שלכם בין היתר ע"י הרצת קוד "חיצוני" שישתמש בפונקציונליות שפיתחתם. לנוחיותכם, מצורף קובץ בשם test.c שיסייע לכם בבדיקה של הקוד שלכם (זהו בעצם sanity check). תוכלו לכתוב טסטים מורכבים יותר בצורה דומה. לא יינתנו טסטים נוספים על ידינו, מכיוון שחשוב שתתנסו גם בצד הזה של בדיקת הקוד שלכם, וזהו חלק מהתרגיל.

11. בתרגיל זה נבדוק זליגות זיכרון. יש לטפל בכל הקצאת זיכרון ולדאוג לשחרורו. ניתן להיעזר בכלי שבודק זליגות כמו Valgrind.
לא יופחתו נקודות על זליגות מסוג still reachable, אך עם זאת כאמור מצופה מכם לשחרר כל זיכרון שהוקצה על ידכם.
לנוחיותכם, יינתן פידבק בנוגע לדליפות זיכרון בעת ההגשה במערכת ה-submit, לפי הפורמט הבא:

(1) 0 line errors, memory leak: 0 bytes not freed and 0 bytes lost, 100 points out of 100.

לא יופחתו נקודות על bytes שמסווגים כ- not freed.
שימו לב - בבדיקה הבסיסית ב-submit לא תהיה הפחתת נקודות על דליפות זיכרון, אך זה לא אומר שבבדיקה בפועל לא יופחתו על כך נקודות. כלומר, ניתן לקבל ציון 100 ב-submit גם בהגשה של תרגיל עם דליפות זיכרון, אך זה לא מעיד על הציון הסופי בתרגיל.

גם באופן כללי, חשוב לזכור שהציון בבדיקה הבסיסית ב-submit אינו מעיד על הציון הסופי בתרגיל. מטרת בדיקה זו היא לאפשר לכם לוודא שהקוד שלכם מתקמפל ורץ בצורה תקינה, אך היא אינה בודקת את נכונות הריצה מעבר לבדיקה בסיסית ביותר (שקיבלתם בקובץ test.c).
הציון בבדיקה ב-submit משקף רק הצלחה ב-test הספציפי שהורץ (וכאמור גם לא מתייחס לתוצאת בדיקת הדליפות).

12. **הרצה –** בנוסף לקבצים המצורפים שהוזכרו לאורך התרגיל, מצורף גם Makefile. על מנת להשתמש בקובץ הבדיקה test.c, עליכם לדאוג שהוא ימצא באותה תיקיה יחד עם הקבצים threadPool.h, threadPool.c, Makefile, **osqueue.h, osqueue.c**.
יש להריץ:

```
make
```

./a.out

13. אין להגיש קבצי בדיקה שלכם.

14. לסיכום – <u>הקבצים המצורפים לתרגיל הם:</u>
Makefile: אין להגיש אותו.
threadPool.h: יש להגיש אותו, לאחר שינויים בהתאם לאופן שהוגדר בתרגיל בלבד.
osqueue.c, osqueue.h: קבצי עזר לשימושכם. אין להגיש או לשנות אותם (לצורך בדיקות שלכם). בדיקת התרגיל תתבצע עם הקבצים שהעברנו לכם.
test.c: אין להגיש אותו.
<u>הקבצים אותם יש להגיש:</u>
threadPool.c – מימוש מלא שלכם
threadPool.h –יש לעבוד על הקובץ שצורף

<u>הערות:</u>
1. אין להשתמש בפונקציונאליות קיימת של thread pool או במימושים קיימים. העתקות ייבדקו.
2. במצב שקריאת מערכת (system call - אשר מוזכרת ב-section מספר 2 ב-man) נכשלה יש להדפיס את הודעת השגיאה
"Error in system call" בעזרת הפונקציה perror. זכרו שהפונקציה perror כבר מדפיסה תו ירידת שורה לאחר ההדפסה.
במצב שפונקציה אחרת (כלומר פונקציית ספרייה שאינה system call) נכשלה יש להדפיס הודעה אינפורמטיבית מתאימה לבחירתכם, באמצעות perror.
עבור שני סוגי השגיאות, יש לצאת מהתוכנית בצורה מסודרת (ולהחזיר 1-) ולדאוג לשחרור משאבים.
3. **אין להשתמש ב-sleep (או וריאציה שלה) ככלי סנכרון או בכלל.**
בנוסף אין להשתמש בסיגנלים (כלומר פונקציות כמו signal לסוגיה השונים, kill, alarm, pause וכו' – אסורות).
מעבר לכך, אתם יכולים להשתמש בכל קריאות המערכת שנלמדו בתרגולים עד תרגול 6 כולל.
אין להשתמש בפונקציות ספריה אלטרנטיביות לקריאות המערכת.
4. אין להשתמש במשתנים גלובליים.
5. התרגיל צריך לעבוד בצורה תקינה על שרת ה-planet. עם זאת, שימו לב שיתכן שתקבלו את השגיאה Resource temporarily unavailable בניסיון הרצה של מספר גדול של חוטים על ה-planet. אנחנו מודעים לכך וזה בסדר. בכל מקרה, השגיאה צריכה להיות מודפסת (ע"י perror).

בהצלחה!