ב"ה

תרגיל מס' 4 – תכנות מרובה חוטים

הוראות הגשה

- שאלות בנוגע לתרגיל יש לשלוח בפורום הקורס במודל.
- **מועד אחרון להגשה: 14/6/20 23:59.**

מועד אחרון להגשה עם בדיקה מאוחרת (בצירוף LATE-SUBMISSION, עפ"י ההנחיות שנכתבו בלוח ההודעות): 30/6/20 23:59 .

- 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 היא אישית. "עבודה משותפת" דינה כהעתקה.**
- אין להדפיס שום דבר מעבר למה שנתבקש בתרגיל.
- יש לוודא שהתרגיל מתקמפל ורץ על ה U2 ללא שגיאות/אזהרות.
- חובה לציין שם מלא ות.ז בראש התרגיל, לפי הפורמט הבא:

#Israel Israeli 123456789

עבור הגשה באיחור יש להוסיף LATE-SUBMISSION

#Israel Israeli 123456789 LATE-SUBMISSION

- שימו לב להערות בסוף התרגיל

תכנות מרובה חוטים

הנחיות עבור ex4

- שם התרגיל: ex4
- שמות קבצי המקור (source files) 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 יווצר עם N threads – על מנת שיוכל לטפל בלכל היותר N משימות בו זמנית. הפונקציה tpInsertTask() אחראית על הכנסת משימה חדשה לתוך תור משימות בתוך ה thread pool. משימה שהוכנסה (enqueue) לתוך התור תישאר שם עד שאחד מה threads יסיים את משימתו הקודמת, יוציא (dequeue) את המשימה הבאה מהתור ויבצע אותה. אם thread מתוך ה thread pool מסיים לבצע את משימתו ואין משימות שהוכנסו לתור – הוא ימתין (ללא busy waiting!) על התור עד שמשימה חדשה תוכנס.

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 זו 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 אחרי זה לחזור. אם המספר שווה לאפס, יש לחכות רק לסיום המשימות שכבר רצות (ולא ניתן יהיה 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- אם נכשלת במידה והפונקציה tpDestroy בדיוק נקראת עבור ה thread pool.

<u>התוכנית</u>

עליכם לממש את הממשק שתואר למעלה 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 בתוך ה thread pool – כל הפעולות יצליחו. חוץ כמובן, ממקרה בו ה thread pool נהרס, מקרה בו ההתנהגות לא מוגדרת ואנחנו 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 ש:
   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.
   בבדיקה שלנו נבדוק רק מקרים בטוחים שבהם בטוח ש tpDestroy מחכה למשימות שיסתיימו (לא ננסה להכניס משימות חדשות בזמן הביצוע של פונקציה זו וגם לא ננסה לקרוא לה שוב ע"י thread אחר).
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!
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. יש לטפל בכל הקצאת זיכרון ולדאוג לשחרורו – בתרגיל זה נבדוק זליגות. ניתן להיעזר בכלי שבודק זליגות כמו Valgrind.
11. **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.
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.c, osqueue.h**.

   יש להריץ:

   ```
   make
   ./run
   ```

13. אין צורך להגיש קבצי בדיקה שלכם.
14. לסיכום – הקבצים שצירפנו הם:

    Makefile: אין להגיש אותו

    threadPool.h: יש להגיש אותו, לאחר שינויים שהוגדרו בתרגיל (ודאו שעקבתם!)

    osqueue.c, osqueue.h: קבצי עזר לשימושכם. אין להגיש או לשנות אותם (לצורך בדיקות שלכם). בדיקת התרגיל תתבצע עם הקבצים שהעברנו לכם.

    הקבצים אותם יש להגיש:

    threadPool.c – מימוש מלא שלכם

    threadPool.h –יש לעבוד על הקובץ שצורף

הערות:

1. אין להשתמש בפונקציונאליות קיימת של thread pool או במימושים קיימים. העתקות ייבדקו.
2. אתם יכולים להשתמש בכל קריאות המערכת שנלמדו בתרגולים עד היום. **אין להשתמש בפונקציות ספריה אלטרנטיביות לקריאות המערכת.**
3. במצב שקריאת מערכת (SYSCALL) נכשלה יש להדפיס את הודעת השגיאה "Error in system call" בעזרת הפונקציה perror.

   במצב שפונקציה אחרת (כלומר פונקציית ספרייה שאינה system call).במקרה של שגיאה יש לצאת מהתוכנית בצורה מסודרת ולדאוג לשחרור משאבים.
4. **אין להשתמש ב sleep ככלי סנכרון או בכלל.**

   בנוסף אין להשתמש בסיגנלים (כלומר פונקציות כמו signal לסוגיה השונים, kill, alarm, pause וכו' – אסורות).
5. אין להשתמש במשתנים גלובליים.

6. התרגיל צריך לעבוד בצורה תקינה על ה-U2. עם זאת, שימו לב, יתכן שתקבלו את השגיאה Resource temporarily unavailable בניסיון הרצה של מספר גדול של חוטים על ה-U2. אנחנו מודעים לכך וזה בסדר. בכל מקרה, השגיאה צריכה להיות מודפסת (ע"י perror).

בהצלחה!