ב"ה

תרגיל מס' 4 – תכנות מרובה חוטים

הוראות הגשה

- שאלות בנוגע לתרגיל יש לשלוח בפורום הקורס במודל.
- **מועד אחרון להגשה: 13/05/19 23:59.**
  מועד אחרון להגשה עם הורדת 10 נקודות 14/05/19 23:59 .
  לא תינתן הארכה נוספת משום סיבה, אנא תכננו בהתאם את הזמן!
- 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
- **חובה לבדוק כל פונקציה האם היא הצליחה או לא**, אם היא לא הצליחה יש לתת הודעה מתאימה ל STDERR ולסיים את התכנית (בצורה נקייה כמובן).
- **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.
- שימו לב להערות בסוף התרגיל

תכנות מרובה חוטים

הנחיות עבור ex4

- שם התרגיל: ex4
- שמות קבצי המקור (source file) שיש לשלוח: threadPool.h, threadPool.c
- יש לכתוב שם מלא ות.ז. בראש הקובץ.

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 pol. 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 מימוש של תור (queue) – osqueue.h, osqueue.c ואתם יכולים להיעזר בזה לצורך מימוש התרגיל.

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.

התוכנית

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.

דגשים חשובים

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.
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 thead 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
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
   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.
11. אין צורך להגיש קבצי בדיקה שלכם.

הערות:

1. אין להשתמש בפונקציונאליות קיימת של thread pool
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 write ל file descriptor מספר 2 (stderr).
4. **אין להשתמש ב SLEEP ככלי סנכרון או בכלל.**

בהצלחה !