מערכות הפעלה - תרגיל בית 3

אוניברסיטת חיפה

סמסטר אביב תשפ"ד

לתרגיל זה שני חלקים - תיאורטי ומעשי.

1. חלק תיאורטי (16 נקודות), בו יהיה עליכם לענות על מספר שאלות תיאורטיות.
2. חלק מעשי (84 נקודות), בו יהיה עליכם לממש רשימה מקושרת מקבילית.

חלק תיאורטי

1. שני תהליכים מריצים במקביל את התוכנית באיור 1. האם יש mutual exclusion?
2. ציינו את כל הפלטים האפשריים של התוכנית באיור 2. הניחו שהתכנית מקבלת בדיוק שני ארגומנטים (למשל ./a.out 1 2).

```
int pid = getpid();
while (1) {
        x = pid;
        if (y && y != pid) continue;
        y = pid;
        if (x != pid) continue;
        /* critical section */
}
```

איור 1: תכנית קצרה. המשתנים x, y משותפים לשני התהליכים ומאותחלים ל-0 בתחילת הריצה.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int last = -1, sum = 0, mode;
pthread_mutex_t mutex;

void* f(void* p) {
        if (mode) pthread_mutex_lock(&mutex);
        int v = *(int *) p;
        sum += v;
        last = v;
        if (mode) pthread_mutex_unlock(&mutex);
}

int main(int argc, char** argv) {
        int n = atoi(argv[1]);
        int *x;
        mode = atoi(argv[2]);
        pthread_t tid;
        pthread_mutex_init(&mutex, NULL);
        while (n--) {
                x = (int*) malloc(sizeof(int));
                *x = n;
                pthread_create(&tid, NULL, f, (void*) x);
        }
        printf("%d %d\n", sum, last);
}
```

איור 2: תכנית קצרה.

חלק מעשי

- בתרגיל זה עליכם לממש מבנה נתונים מסוג רשימה מקושרת (Linked List), אשר יתמוך במקביליות - כלומר, יאפשר למספר חוטים לעדכן את מבנה הנתונים במקביל.
- איברי הרשימה יהיו ממויינים בסדר עולה, לפי ערכו של שדה integer שכל רשומה תכיל.
- עליכם להשתמש במנגנוני סנכרון של הספרייה pthreads, כדי לתאם את הגישה של החוטים השונים לנתוני הרשימה המקושרת.
- הממשק התוכנתי שעליכם לממש מתואר באיור 3.

```
typedef struct node node;
typedef struct list list;
list* create_list();
void delete_list(list* list);
void print_list(list* list);
void insert_value(list* list, int value);
void remove_value(list* list, int value);
void count_list(list* list, int (*predicate)(int));
```

איור 3: ממשק הרשימה המקושרת.

פירוט הממשק

- המבנה **struct node**: במבנה זה עליכם להגדיר את השדות אשר כל צומת ברשימה המקושרת יכיל.
- המבנה **struct list**: במבנה זה עליכם להגדיר את השדות של רשימה מקושרת מקבילית.
- הפונקציה **create_list**: פונקציה זו יוצרת רשימה מקושרת חדשה (ריקה).

- ⊗ ערך חזרה: מצביע לרשימה החדשה.

- ⊕ הפונקציה **delete_list**: פונקציה זו משחררת רשימה קיימת.
  - ⊗ אם הרשימה לא ריקה, על הפונקציה למחוק את כל איברי הרשימה.
  - ⊗ בנוסף, על הפונקציה לשחרר איזורי זיכרון אשר הוקצו עבור הרשימה ואיבריה.
  - ⊗ פרמטר קלט **list**: מצביע לרשימה המיועדת למחיקה.

- ⊕ הפונקציה **print_list**: פונקציה זו מדפיסה את האיברים של רשימה מקושרת.
  - ⊗ פרמטר קלט **list**: מצביע לרשימה שאיבריה יודפסו.

- ⊕ הפונקציה **insert_value**: פונקציה זו מכניסה איבר חדש לרשימה (תוך כדי שמירה על סדר ממוין עולה), אשר ערכו יהיה שווה ל-value.
  - ⊗ פרמטר קלט **list**: מצביע לרשימה אליה יוכנס האיבר החדש.
  - ⊗ פרמטר קלט **value**: ערכו של האיבר החדש שיוכנס לרשימה.

- ⊕ הפונקציה **remove_value**: פונקציה זו מסירה את האיבר הראשון ברשימה אשר ערכו שווה ל-value.
  - ⊗ פרמטר קלט **list**: מצביע לרשימה הרלוונטית.
  - ⊗ פרמטר קלט **value**: ערכו של האיבר שיוסר.

- ⊕ הפונקציה **count_list**: פונקציה זו מונה ומדפיסה את מספר האיברים ברשימה אשר מקיימים את ה-predicate שהמשתמש מספק.
  - ⊗ פרמטר קלט **list**: מצביע לרשימה הרלוונטית.
  - ⊗ פרמטר קלט **predicate**: מצביע לפונקציה אשר תופעל על כל איברי הרשימה. אם איבר מקיים את תנאי ה-predicate אז הפונקציה תחזיר 1, ואחרת 0.

דרישות מימוש

בבדיקת התרגיל ייבדקו הדגשים הבאים (**באופן אוטומטי**):

נכונות מימוש

על הרשימה המקושרת לעבוד כמצופה (ללא קשר לתמיכה במקביליות).

תמיכה במקביליות

- ⊕ **אסור** לנעול את **כל** הרשימה כאשר חוט מסוים ניגש לעדכן את איברי הרשימה, ובכך למנוע מחוטים אחרים לעדכן נתונים בלתי תלויים אחרים של הרשימה.
- ⊕ למשל, נניח שקיימת רשימה עם 100 איברים, שמכילה את האיברים $\{1, \ldots, 100\}$ (בסדר ממוין עולה). כעת, נניח שחוט A מבקש להסיר את האיבר שערכו 80.

  1. בעת ביצוע פעולת ההסרה, חוט A יבצע שינויים רק בסביבה המקומית של האיבר המוסר (80). לכן, לא צריכה להיות מניעה מחוט B להסיר במקביל את האיבר שערכו 10.
  2. אם חוט B ינסה להסיר בו-זמנית את 80 (או לחילופין, את 79) בזמן שחוט A עדיין מבצע את פעולת ההסרה, אז חוט B צריך להיחסם עד ש-A יסיים את פעולתו.

- ⊕ דרך מימוש אפשרית לתמיכה בדרישות אלו היא hand over hand locking.

דליפות זיכרון

כל איזור זיכרון שהוקצה באופן דינמי (באמצעות malloc) חייב להיות משוחרר עד סוף ריצת התוכנית.

קומפילציה, הרצה ובדיקה

קבצי התרגיל

בקובץ ה-zip של התרגיל מסופקים לכם הקבצים הבאים:

1. concurrent_list.c

⊕ קוד שלד שמהווה נקודת התחלה לכתיבת התרגיל - השתמשו בו.

2. concurrent_list.h (**אין לערוך קובץ זה**)

⊕ מכיל את הגדרות הפונקציות ומבני הנתונים שעליכם לממש, כפי שמופיע בתיאור התרגיל לעיל.

⊕ אין לבצע שינויים לבצע בקובץ ואין להגישו - קובץ זה יתווסף אוטומטית בבדיקת ההגשה.

3. Makefile (**אין לערוך קובץ זה**)

⊕ קובץ זה מגדיר כיצד להדר את התרגיל באמצעות GCC.

⊕ כדי להדר את התרגיל, הריצו את הפקודה `make` ב-shell - כך GCC יהדר את הקובץ test.c יחד עם המימוש של concurrenct_list.c.

⊕ אם הקומפילציה מצליחה יווצר קובץ ELF בשם test, אותו תוכלו להריץ כדי לבדוק את המימוש שלכם ע"י הזנת פקודות כמפורט בהמשך.

4. test.c (**אין לערוך קובץ זה**)

⊕ קובץ זה מגדיר תוכנית בדיקה, באמצעותה תבדקו את המימוש של concurrent_list.cc. תוכנית הבדיקה מקבלת את הפקודות הבאות:

(א) **create_list** - פקודה זו תיצור רשימה מקושרת מקבילית חדשה ע"י קריאה לפונקציה create_list.

(ב) **delete_list** - פקודה זו תיצור חוט חדש אשר ימחק את הרשימה שנוצרה ע"י create_list. החוט יקרא לפונקציה delete_list.

(ג) **print_list** - פקודה זו תיצור חוט חדש אשר ידפיס את הרשימה שנוצרה ע"י create_list. החוט יקרא לפונקציה print_list.

(ד) **insert_value #$value$** - פקודה זו תיצור חוט חדש אשר יכניס איבר חדש עם ערך #$value$ לרשימה שנוצרה ע"י create_list. החוט יקרא לפונקציה insert_value.

(ה) **remove_value #$value$** - פקודה זו תיצור חוט חדש אשר ינסה להסיר איבר קיים בעל ערך #$value$ מהרשימה שנוצרה ע"י create_list. החוט יקרא לפונקציה remove_value.

(ו) **count_greater #$value$** - פקודה זו תיצור חוט חדש אשר ידפיס את מספר האיברים ברשימה שנוצרה ע"י create_list שערכם גדול מ-#$value$. החוט יקרא לפונקציה count_list, עם פונקציית predicate מוגדרת מראש (אתם מוזמנים לבחון את הקוד).

(ז) **join** - פקודה זו תגרום לחוט הראשי (זה שמריץ את ה-main של test.c) להמתין לסיום כל החוטים שנוצרו עד כה.

(ח) **exit** - פקודה זו תסיים את ריצת התוכנית. ניתן להניח שבבדיקת התרגיל תמיד תתבצע הפקודה delete_list לפני הפקודה exit.

⊕ מומלץ לעבור על הפונקציה execute_command בקובץ, על מנת להבין כיצד הפקודות מפורשות.

⊕ אתם מוזמנים להרחיב את הקובץ לבדיקת דגשים נוספים שלא מטופלים על ידי המימוש הנתון של test.c (למשל, בדיקה מקיפה של תמיכה במקביליות, כמתואר בדרישות המימוש).

⊕ ניתן לראות דוגמת הרצה של test בלינק.

5. minimal.supp (**אין לערוך קובץ זה**)

⊕ קובץ זה משמש בכדי לבדוק באופן אוטומטי האם קיימות דליפות זיכרון במימוש של concurrent_list.c, כמפורט בהמשך.

6. valgrind-3.4.1.tar.bz2 (**אין לערוך קובץ זה**)

⊕ קובץ zip המכיל כלי באמצעותו תתבצע בדיקה לדליפות זיכרון בתוכנית.

⊕ עקבו אחר ההוראות בלינק כדי להתקין את הכלי במכונה הוירטואלית.

בזמן העבודה על התרגיל, ודאו כי כל הקבצים נמצאים באותה הספרייה.

בדיקת דליפות זיכרון

1. התקינו את valgrind, מוזמנים להיעזר בקישור לעיל.
2. כדי לבדוק דליפות זיכרון בתוכנית, הריצו את ה-test ELF באמצעות הפקודה:
```
valgrind --leak-check=full --suppressions=minimal.supp ./test < stress.in
```
3. אם תרצו להריץ את התוכנית עם קובץ בדיקה שמכיל פקודות מוכנות מראש, הריצו את הפקודה:
```
valgrind --leak-check=full --suppressions=minimal.supp ./test < input_file.in
```
4. במידה וקיימת דליפת זיכרון בתוכנית, הפלט של valgrind יפרט איזו שורה בקובץ גרמה להקצאת איזור הזיכרון שלא שוחרר, בדומה לאיור 4.

```
==29704== 28 bytes in 1 blocks are definitely lost in loss record 1 of 2
==29704==    at 0x4017BFD: malloc (m_replacemalloc/vg_replace_malloc.c:207)
==29704==    by 0x8049330: create_list (concurrent_list.c:159)
==29704==    by 0x8048C29: execute_command (test.c:134)
==29704==    by 0x8048FAF: main (test.c:228)
```

איור 4: דוגמא לפלט של valgrind.

הגשה

הגשה במודל, לפי הפורמט הבא:

1. עליכם ליצור קובץ zip (השתמשו ב-zip או gzip בלבד) בשם hw3_id1_id2.zip, כאשר id1, id2 הם מספרי תעודות הזהות של המגישים.
2. קובץ ה-zip מכיל **אך ורק** את הקבצים ההבאים, **ללא תתי-ספריות**.

- ⊕ myshell.c
- ⊕ dry.pdf, שמכיל את התשובות לחלק התיאורטי - **קובץ pdf בלבד**.
- ⊕ submitters.txt, שמכיל את מספרי הזהות והשמות של מגישי התרגיל, מופרדים ע"י פסיק. למשל:

```
Bill Gates,bill@microsoft.com,123456789
Linus Torvalds,linus@gmail.com,234567890
```

3. צרו את קובץ ה-zip באמצעות הפקודה

```
zip hw3_id1_id2.zip concurrent_list.c dry.pdf submitters.txt
```

4. הגישו את ה-zip דרך המודל.