תרגיל בית 3

תיאור התרגיל

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 איבר יכיל.

עליכם להשתמש במנגנוני סנכרון של הסיפריה pthreads כדי לתאם את הגישה של החוטים השונים לנתוני הרשימה המקושרת.

הממשק התוכנתי שעליכם לממש, מוגדר באופן הבא:

```
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));
```

פירוט הממשק התוכנית

המבנה struct node

במבנה נתונים זה עליכם להגדיר את השדות אשר כל צומת ברשימה המקושרת יכיל.

המבנה struct list

במנה זה עליכם להגדיר את השדות אשר של רשימה מקושרת מקבילית.

הפונקציה create_list
פונקציה זו יוצרת רשימה מקושרת חדשה וריקה.

- **ערך חזרה**: מצביע לרשימה החדשה.

הפונקציה delete_list
פונקציה זו משחררת רשימה קיימת. אם הרשימה לא ריקה, על הפונקציה למחוק את כל איברי הרשימה. כמו כן, על הפונקציה לשחרר איזורי זיכרון אשר הוקצו עבור הרשימה ואיבריה.

- **פרמטר קלט list**: מצביע לרשימה המיועדת למחיקה.

הפונקציה print_list
פונקציה זו מדפיסה את איבריה של רשימה מקושרת.

- **פרמטר קלט list**: מצביע לרשימה שאיבריה יודפסו.

הפונקציה insert_value
פונקציה זו מכניסה איבר חדש לרשימה (תוך כדי שמירה על סדר ממויין 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 (כאשר הסריקה מתחילה מראש הרשימה).

- **פרמטר קלט list**: מצביע לרשימה ממנה יוסר איבר אשר ערכו שווה ל- value.
- **פרמטר קלט value**: ערכו של האיבר 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 אשר על איבריה יופעל ה- predicate.
- **פרמטר קלט predicate**: מצביע לפונקציה אשר תופעל על כל איבר ברשימה. אם איבר ברשימה מקיים את תנאי ה- predicate, אז הפונקציה תחזיר 1, אחרת, יוחזר 0.

דרישות מימוש
בבדיקת התרגיל, ייבדקו הדגשים הבאים: **(כל הדגשים ייבדקו באופן אוטומטי)**

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

תמיכה במקביליות
אסור לנעול את **כל** הרשימה כאשר חוט מסויים ניגש לעדכן את איברי הרשימה, ובכך למנוע מחוטים אחרים לעדכן נתונים בלתי תלויים אחרים של הרשימה.

למשל, נניח שקיימת רשימה עם 100 איברים, אשר ערכם נע מ- 1 עד 100 (בסדר ממויין עולה). כעת, נניח שחוט A מבקש להסיר את האיבר ברשימה אשר ערכו שווה ל- 80:

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

דרך מימוש אפשרית כדי לתמוך בדרישות המקביליות הנ"ל, היא hand over hand locking.

דליפות זיכרון (Memory Leaks)

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

קומפילציה, הרצה ובדיקה

קבצים מצורפים

בקובץ ה- ZIP של התרגיל מסופקים לכם הקבצים הבאים:

1. concurrent_list.c
2. concurrent_list.h (אסור לערוך קובץ זה)
3. Makefile (אסור לערוך קובץ זה)
4. test.c (אסור לערוך קובץ זה)
5. minimal.supp (אסור לערוך קובץ זה)
6. valgrind-3.4.1.tar.bz2 (אסור לערוך קובץ זה)

אנא וודאו כי כל הקבצים נמצאים באותה הסיפרייה כאשר אתם עובדים על התרגיל.

להלן הסבר על כל אחד מן הקבצים:

הקובץ concurrent_list.c

הקובץ concurrent_list.c הינו קוד שלד שמהווה נקודת התחלה לכתיבת התרגיל – השתמשו בו.

הקובץ concurrent_list.h

הקובץ concurrent_list.h מכיל את הגדרות הפונקציות ומבני הנתונים שעליכם לממש, כפי שמופיע בתיאור התרגיל לעיל – אסור לבצע שינויים בקובץ זה, וגם אין להגישו – קובץ זה יתווסף אוטומטית כאשר נבדוק את הגשותיכם.

הקובץ test.c

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

1. **create_list**

   פקודה זו תיצור רשימה מקושרת מקבילית חדשה, ע"י קריאה לפונקציה create_list אשר מימשתם ב- concurrent_list.c.
2. **delete_list**

   פקודה זו תיצור חוט חדש אשר ימחק את הרשימה שנוצרה ע"י create_list. החוט יקרא לפונקציה delete_list אשר מימשתם ב- concurrent_list.c.

3. **print_list**
פקודה זו תיצור חוט חדש אשר ידפיס את הרשימה שנוצרה ע"י create_list. החוט יקרא לפונקציה print_list אשר מימשתם ב- concurrent_list.c.
4. **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 אשר מימשתם ב- concurrent_list.c.
5. **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 אשר מימשתם ב- concurrent_list.c.
6. **count_greater #value**
פקודה זו תיצור חוט חדש אשר ידפיס את מספר האיברים ברשימה שנוצרה ע"י create_list אשר ערכם גדול מ- #value. החוט יקרא לפונקציה count_list אשר מימשתם ב- concurrent_list.c, עם פונקציית predicate מוגדרת מראש (אתם מוזמנים לבחון את הקוד).
7. **join**
פקודה זו תגרום לחוט הראשי (החוט שמריץ את main של test.c) למתין לסיום כל החוטים שנוצרו עד כה.
8. **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, כדי להבין כיצד הפקודות מפורשות.

הקובץ Makefile
הקובץ Makefile הינו קובץ המגדיר כיצד להדר את התרגיל באמצעות GCC. כדי להדר את התרגיל, עליכם להריץ את הפקודה make באמצעות ה- shell. כתוצאה מהרצת פקודה זו, GCC יהדר את הקובץ test.c ביחד עם המימוש של concurrent_list.c. אם פעולת ההידור מצליחה, אז יווצר קובץ הרצה בשם test, אותו תוכלו להריץ כדי לבדוק את המימוש שלכם ע"י הזנת פקודות טקסט כמפורט לעיל.

אתם מוזמנים להרחיב את התוכנית test.c כדי לבדוק דגשים נוספים שהמימוש הנוכחי של test.c לא מטפל בהם (למשל, בדיקה מקיפה של תמיכה במקביליות, כפי שמפורט בדרישות המימוש).

ניתן לראות דוגמת הרצה של test בוידאו בקישור הבא.

הקובץ minimal.supp
קובץ זה משמש בכדי לבדוץ באופן אוטומטי האם קיימות דליפות זיכרון במימוש של concurrent_list.c (פרטים בהמשך).

הקובץ valgrind-3.4.1.tar.bz2
קובץ ZIP המכיל כלי אשר באמצעותו תתבצע בדיקה האם אין דליפות זכרון בתוכנית. אנא עקבו אחר ההוראות בקישור הבא כדי לבין כיצד להתקין את כלי זה במוכנה הוירטואלית.

בדיקת דליפות זיכרון
תחילה, ודאו כי התקנתם את הכלי valgrind, כפי שמפורט בוידאו בקישור לעיל.

כדי לבדוק האם קיימת דליפת זכרון בתוכנית, הריצו את התוכנית test באמצעות הפקודה הבאה:

```
valgrind --leak-check=full --suppressions=minimal.supp ./test < stress.in
```

אם ברצונכם להריץ את התוכנית עם קובץ בדיקה עם פקודות מוכנות מראש, הריצו את הפקודה הבאה:

```
valgrind --leak-check=full --suppressions=minimal.supp ./test < test_file.in
```

במידה וקיימת דליפת זיכרון בתוכנית, הפלט של valgrind יפרט איזו שורה בקובץ concurrent_list.c גרמה להקצאת בלוק הזכרון שלא שוחרר. הפלט ייראה בדומה לפלט הבא:

```
==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)
```

הגשה

ההגשה הינה אלקטרונית דרך Moodle. עקבו אחר השלבים הבאים:

1. עליכם ליצור קובץ zip (השתמשו ב-zip או gzip בלבד) בשם hw3_id1_id2 כאשר id1,id2 מייצגים את מספרי תעודות הזהות של המגישים.
2. תכולת קובץ ה zip צריכה להיות התכולה הבאה (ללא תתי ספריות!):
   - concurrent_list.c
   - קובץ בשם submitters.txt שמכיל את מספרי הזהות והשמות של מגישי התרגיל מופרדים על ידי פסיק במבנה הבא (לדוגמה):

```
Bill Gates,bill@microsoft.com,123456789

Linus Torvalds,linus@gmail.com,234567890
```

3. את קובץ ה- zip יש ליצור ע"י הרצת הפקודה הבאה:

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

4. הגישו את קובץ ה- zip דרך Moodle.