מערכות הפעלה
234123

תרגיל יבש: 3

הוגש ע״י:

רון ביידר	302895784	Roni2706@gmail.com
שם	מספר סטודנט	אימייל
דביר פרי	300411659	sdperry@campus.technion.ac.il
שם	מספר סטודנט	אימייל

שאלה 1

בתרגול למדנו על מנעול קוראים-כותבים - מנעול המאפשר למספר קוראים בו-זמנית, או לכתוב יחיד לגשת למבנה הנתונים. מנעול קוראים כותבים רקורסיבי הינו מנעול המאפשר נעילה עצמית ע"י חוט שמחזיק במנעול - בריאה או בכתיבה (כלומר, חוט שנעל את המנעול לקריאה/כתיבה והחליט לבצע שוב את אותה הפעולה, יצליח לתפוס את המנעול בשנית מבלי לשחרר אותו קודם לכן).

סיכום ההתנהגות הרצויה של מנעול רקורסיבי:

	חוט שמחזיק במנעול מנסה לתפוס לקריאה	חוט שמחזיק במנעול מנסה לתפוס לכתיבה	חוט אחר מנסה לתפוס לקריאה	חוט אחר מנסה לתפוס לכתיבה
המנעול משוחרר	-	-	הצלחה	הצלחה
המנעול תפוס לקריאה	הצלחה	כשילון	הצלחה	כישלון
המנעול תפוס לכתיבה	כישלון	הצלחה	כישלון	כישלון

א. האם המנעול קוראים כותבים שלמדנו בתרגול רקורסיבי בקריאה? האם הוא רקורסיבי בכתיבה? הסבירו את תשובתכם.

- המנעול קוראים-כותבים שנלמד בתרגול הינו רקורסיבי בקריאה משום שכאשר המנעול תפוס לקריאה, כל חוט שינסה לתפוס את המנעול לקריאה יצליח, בפרט החוט שמחזיק במנעול בפועל. לכן המנעול מפגין את ההתנהגות הרצויה (על פי הטבלה). המנעול קוראים-כותבים שנלמד בתרגול אינו רקורסיבי בכתיבה משום שחוט שמחזיק במנעול לכתיבה יכשל כאשר ינסה לתפוס את המנעול לכתיבה בשנית.

ב. ממשו מנעול קוראים-כותבים רקורסיבי בקריאה ובכתיבה על-בסיס המימוש המופיע בתרגול. כתבו רק את השינויים במבני הנתונים ותנו מימוש מלא של פונקציות הנעילה והשחרור לקריאה ולכתיבה.

- נציע את המימוש הבא :

נוסיף למבנה את השדה הבא $int\ writer\_tid$ שיחזיק את ה-tid של החוט שכותב בפועל או 0 אם אין כותבים פעילים. נאתחל ערך זה ל-0 בפונקציה $readers\_writers\_init()$.

```
void read_lock() {
        pthread_mutex_lock(&global_lock);
        while (number_of_writers > 0)
                pthread_cond_wait (&readers_condition, &global_lock);
        number_of_readers++;
        pthread_mutex_unlock (&global_lock);
}

void read_unlock() {
        pthread_mutex_lock(&global_lock);
        number_of_readers--;
        if (number_of_readers == 0)
                pthread_cond_signal (&writers_condition);
        pthread_mutex_unlock (&global_lock);

}

void write_lock() {
        pthread_mutex_lock (&global_lock);
        while ((number_of_writers >0 && writer_tid != current->pid) || (number_of_readers > 0))
                pthread_cond_wait (&writers_condition, &global_lock);
        number_of_writers++;
        writer_tid = current->pid;
        pthread_mutex_unlock (&global_lock);

}
```

```
void write_unlock() {
        pthread_mutex_lock (&global_lock);
        number_of_writers--;
        writer_tid = 0;
        if (number_of_writers == 0) {
                pthread_cond_broadcasr (&readers_condition);
                pthread_cond_signal (&writers_condition);
        }
        pthread_mutex_unlock (&global_lock);
}
```

ג. תארו אילו שינויים היו דרושים על מנת להפוך את המנעול שלכם לכזה המאפשר לחוט המחזיק במנעול לקריאה להצליח לנעול את המנעול לכתיבה באופן רקורסיבי, כלומר, חוט שנעל את המנעול לקריאה והחליט לכתוב יצליח לתפוס את המנעול לכתיבה מבלי לשחרר את המנעול מקריאה קודם לכן. האם נעילה כזו תמיד תהיה אפשרית? הסבירו מדוע.

- על מנת לאפשר לחוט המחזיק במנעול לקריאה להצליח לנעול את המנעול לכתיבה מבלי לשחרר את המנעול מקריאה קודם לכן יש צורך לשנות את תנאי הבדיקה בפונקציה $write\_lock()$. הכניסה לרשימת ההמתנה עבור התנאי תהיה עבור מצב בו קיים יותר מקורא אחד, או שקיים בדיוק קורא אחד והוא לא זה שמנסה לנעול את המנעול לכתיבה.

נעילה זו תתאפשר רק כאשר הקורא המבקש לכתוב הוא הקורא היחיד בפועל (אחרת תהיה כתיבה וקריאה בו זמנית ואז כל מטרת המנעול לא עובדת), לכן בפועל לא תמיד נעילה זו תתאפשר.

<u>שאלה 2</u>

בהנחה שהקוד (שלא מצורף פה) רץ על מכונה עם מעבד יחיד. מה האפשרויות לפלט של הקוד? התשובה צריכה להיות מורכבת מערך מקסימלי אפשרי וערך מינימלי אפשרי עם תרחיש והסבר לכל אחד מהם.

- האפשרויות לפלט של הקוד הן כל הערכים בין 100 ל-200.

במקרה קצה בו $threads[0]$ יסיים את כל הקוד אותו הוא מבצע לפני שימסור את המעבד ל-$threads[1]$, נקבל כי תחילה יעדכן $threads[0]$ את ערך המשתנה הגלובלי $result$ ל-100 ולאחר מכן יעדכן $threads[1]$ את ערך המשתנה $result$ ל-200.

במקרה קצב בו מתבצעת החלפת הקשר בין שני החוטים בתדירות גבוהה וכך ש-$threads[0]$ מוסר את המעבד לפני שמעדכן את ערך המשתנה $result$ בכל איטרציה בקוד שלו, וכן $threads[1]$ מוסר את המעבד לפני שמעדכן את ערך המשתנה $result$ בכל איטרציה בקוד שלו. כך נקבל מצב בו שני החוטים משנים את ערך $result$ פעמיים ברצף לאותו הערך. בפועל בכל פעם ש-$threads[1]$ מעדכן את $result$, הוא דורס את העדכון של $threads[0]$. כך שנקבל כי ערכו של $result$ בסוף הריצה הוא 100.

שאלה 3

א. שלושה חוטים, B, C, D, משתפים פעולה על מנת לסכום את הערכים (מספרים שלמים) המאוחסנים במערך X בגודל 20, באופן הבא:

- **B סוכם את האיברים הזוגיים : $X[0], X[2], \dots, X[18]$.**
- **C סוכם את האיברים האי-זוגיים : $X[1], X[3], \dots, X[19]$.**
- **D מחבר את התוצאות של B ו-C.**

השלם את הקוד המופיע להלן (כולל הגדרת משתנים, אם צריך), כך ש:

- **כל חוט מסיים אחרי שגמר את תפקידו.**
- **B ו-C ניגשים לאיברים שונים של המערך בו-זמנית.**
- **אמצעי הסנכרון היחיד המותר בפתרון הוא סמפורים.**
- **החוטים לא מבצעים $busy - wait$.**
- **ניתן להניח שהמערך X מאותחל עם ערכים מתאימים.**
- **אין צורך לטפל בקוד שיוצר את החוטים עצמם.**

- הקוד לאחר השלמתו :

```
int X[20];
int sum, sumB, sumC;
sem_t semB, semC;
sem_init (semB,0,0);        // needs to initialized in the main function
sem_init(semC,0,0);         // (could be implement as one semaphore in case we could initialize it's value to -1)

ThreadB:
    sumB = 0;
    for (i=0; i<20; i+=2)
            sumB += X[i];
    sem_post (&semB);

ThreadC:
    sumC = 0;
    for (i=1; i<20; i+=2)
            sumC += X[i];
    sem_post (&semC);

ThreadD:
    sem_wait (&semB);
    sem_wait (&semC);
    sum = sumB + sumC;
```

ב. למקרים כאלה משתמשים לפעמים בכלי סנכרון כללי יותר שנקרא מחסום ($barrier$).

גרסה פשוטה של המחסום תומכת בפונקציה $barrier\_enter$ עבור מספר תהליכים N ידוע מראש. תהליך הקורא לפונקציה ימתין (המתנה חוסמת) עד שכל N התהליכים יקראו ל-$barrier\_enter$. רק לאחר שכל התהליכים "נכנסו למחסום" (הפעילו את הפונקציה $barrier\_enter$) יוכלו כל N התהליכים להמשיך את הביצוע שלהם.

כתבו פסאודו-קוד לפונקציה $barrier\_enter$ לשימוש חד פעמי (כלומר ניתן להניח שחוט שעבר את המחסום לא ינסה לחכות שוב - אפילו אחרי שכל החוטים עברו את המחסום), אשר משתמש רק במנעול אחד ובמשתנה תנאי אחד. מותר להשתמש במשתנים גלובליים נוספים (אבל לא באמצעי סנכרון נוספים).

- הקוד לאחר השלמתו:

```
Lock L
Condition C
int num//מאותחל ל-0

Function ente_barrier
{
        lock L
        num ++
        if (num < N)
                cond_wait (C) // unlocks L as a side effect
        else
                cond_broadcast (C)
        unlock L
}
```