Homework 3 Dry

Due Date: 12/01/2022 23:30

Teaching assistant in charge: Arad Kotzer

Important: the Q&A for the exercise will take place at a public forum Piazza only. Critical updates about the HW will be published in pinned notes in the piazza forum. These notes are mandatory and it is your responsibility to be updated. A number of guidelines to use the forum:

- Read previous Q&A carefully before asking the question; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You're not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding hw3 , put them in the hw3 folder

Only the TA in charge can authorize postponements. In case you need a postponement, fill in this form - https://forms.gle/W1XaKRp3J3tELPsS6

Dry part submission instructions:

1. Please submit the dry part to the electronic submission of the dry part on the course website.
2. The dry part submission must contain a single dry.pdf file containing the following:
    a. The first page should contain the details about the submitters - Name, ID number and email address.
    b. Your answers to the dry part questions.
3. Only typed submissions will be accepted. Scanned handwritten submissions will not be accepted.
4. Only PDF format will be accepted.
5. You do not need to submit anything in the course cell.
6. When you submit, **retain your confirmation code and a copy of the PDF**, in case of technical failure. It is **the only valid proof** of your submission.

יש לנמק כל תשובה אלא אם במפורש נאמר אחרת, תשובות ללא נימוק לא יתקבלו.

שאלה 1 - גרעין ניתן להפקעה (52 נק')

כפי שלמדנו בתרגולים, גרעין לינוקס 2.4 אינו ניתן להפקעה (non-preemptible).

א. (8 נק') מדוע המפתחים הראשונים של לינוקס בחרו לממש גרעין שאינו ניתן להפקעה?
רמז: לינוקס פותחה בתחילת שנות ה-90, כאשר מחשבים אישיים היו בעלי מעבד יחיד. איזה יתרון מעניק גרעין שאינו ניתן להפקעה במערכת עם מעבד יחיד? מדוע, לעומת זאת, היתרון מצטמצם במערכת מרובת מעבדים?

ב. (8 נק') גרעין לינוקס מסווג תהליכים רגילים לשתי קבוצות: חישוביים ואינטראקטיביים.
איזה סוג של תהליכים עלול לסבול (מבחינת ביצועים) מגרעין שאינו ניתן להפקעה?
התייחסו לשני סוגי התהליכים והסבירו מדוע סוג אחד נפגע וסוג שני אינו נפגע.

לקראת סוף שנות האלפיים, בתקופה בה אנשים הסתובבו עם ווקמן בכיס ואף אחד עוד לא חלם על סמארטפונים, גרסת גרעין לינוקס האחרונה הייתה 2.4---הגרסה הנלמדת בתרגולים. באותם שנים, קבוצה של מפתחי לינוקס הבחינו שנגן המוזיקה שלהם מקרטע (כלומר שומעים "קפיצות" במהלך הנגינה) בזמן שהמערכת שלהם עמוסה, למשל בזמן שהם מהדרים את גרעין לינוקס ברקע.
לצורך המשך השאלה, נסביר על קצה המזלג איך עובד נגן מוזיקה: רצועת שמע ("שיר") מורכבת מהרבה דגימות (samples) שנשלחות להתקן חומרה מיוחד - כרטיס קול. כרטיס הקול ממיר את המידע הדיגיטלי (הדגימות) לאות אנלוגי שהרמקולים יכולים להשמיע. על מנת לשפר את הביצועים, כרטיס הקול קורא את הדגימות מחוצץ (buffer) שאותו ממלאת מערכת ההפעלה בתור האחראית על גישה להתקני קלט/פלט. לאחר שנגן המוזיקה מסיים למלא את החוצץ, הוא מוותר על המעבד ועובר להמתין לפרק זמן מדוד, עד שיצטרך למלא שוב את החוצץ.

ג. (6 נק') בהנחה שברגע נתון החוצץ של כרטיס הקול מלא במידע, מה משך הזמן המקסימלי שבו יכולה מערכת ההפעלה להריץ תהליכים אחרים לפני שיהיה עליה לחזור ולמלא את החוצץ כך שהמשתמש לא ישמע "קפיצות"?
נתון כי: החוצץ של כרטיס הקול הוא בגודל KiB64, תדירות הדגימות ברצועת השמע היא KHz44.1, וכל דגימה מכילה מידע על שני ערוצים (stereo) ברוחב 16 ביט כל אחד.

ד. (7 נק') בהנחה שזמן הקריאה מהדיסק הוא הדומיננטי ביותר בפעולת נגן המוזיקה, מה אופיו של התהליך---חישובי או אינטראקטיבי? הניחו כי רוחב פס של דיסק בשנת 2000 היה מסדר גודל של MiB/s10.

כדי להתגבר על הבעיה של נגן המוזיקה המקרטע, אפשר כמובן לקצר את מסלולי הבקרה בגרעין לינוקס 2.4. למרבה הצער, הפתרון הזה נאיבי: מפתחי לינוקס גם כך מנסים לקצר את זמן השהייה בגרעין כדי להקטין את התקורה של מערכת ההפעלה. אם זה היה כל כך פשוט, הם כבר היו עושים זאת...

ה. (13 נק') אינגו מולנאר (Ingo Molnar), מפתח גרעין ידוע, הציע להוסיף "נקודות הפקעה אפשריות" במסלולי בקרה ארוכים בגרעין לינוקס 2.4, כלומר נקודות בקוד בהן הגרעין שואל "האם תהליך אחר צריך לרוץ במקומי?" ומבצע החלפת הקשר במידה והתשובה חיובית (למשל, אם תהליך בעדיפות גבוהה יותר ממתין לריצה.)
מנו שלושה חסרונות בהצעה של אינגו.

ו. (10 נק') בעיית סנכרון נוספת שקיימת רק בגרעין ניתן להפקעה היא בגישה למשתנים המוגדרים פר-מעבד (per-CPU variables). למשל, המערך per_cpu_array (לא קיים באמת בקוד הגרעין):

```
struct per_cpu_struct per_cpu_array[NR_CPUS];
```

מגדיר רשומה נפרדת לכל אחד מהמעבדים במערכת, אליה ניגשים באופן הבא:

```
something = per_cpu_array[smp_processor_id()];
/* execute more work here... */
per_cpu_array[smp_processor_id()] = something_else;
```

כדי להגן על משתנים המוגדרים פר-מעבד, גרעין לינוקס מונע הפקעות בזמן שניגשים אליהם: בכניסה לקטע הקריטי המונה preempt_count מוגדל ב-1, וביציאה מהקטע הקריטי המונה מוקטן ב-1.

א) תארו שתי בעיות שעלולות להיווצר בגישה למשתנים המוגדרים פר-מעבד במידה ולא נמנע הפקעות באמצעות תרחישים מדויקים של שתי בעיות שונות שיכולות להתרחש בגישה למערך `per_cpu_array`.

ב) מדוע בגרעין 2.4 לא הייתה בעיה בגישה למשתנים המוגדרים פר-מעבד?

לסיכום: החל מגרסה 2.6, גרעין לינוקס הציג ביצועים משופרים בזכות היכולת להפקיע את המעבד מתהליכים שרצים ב-kernel mode. על- מנת למדוד את השיפור בביצועים, Robert Love מדד את ההשהיה של קריאות מערכת write() מתוך נגן המוזיקה באמצעות כלי שפיתח Benno Senoner. תוכלו להיווכח בשיפור ע"י התבוננות בגרפים הבאים: גרעין 2.4 לעומת גרעין 2.6. הגרפים מציגים את ההשהיה של קריאת המערכת write() לאורך זמן פעולת נגן המוזיקה, והקו האדום הוא סף ההשהיה שבו האוזן האנושית יכולה להבחין. תוכלו לשים לב לקפיצות שחוצות את הקו האדום כאשר משתמשים בגרעין 2.4, ולהשוות את הגרף לזה שנמדד עבור גרעין 2.6.

שאלה 2 - סינכרון (48 נק')

המצאת המושג "פקולטה נחשבת" החמירה את הסכסוך בין הסטודנטים במדמ"ח ובהנדסת חשמל, ולכן הוגדר כי כאשר סטודנט מאחת הפקולטות רוצה להיכנס לחדר מסויים עליו לציית לכלל הבא: אם יש סטודנטים מפקולטה אחרת בחדר אזי אסור לסטודנט להיכנס ועליו להמתין עד שיעזבו (לעומת זאת, מספר סטודנטים מאותה פקולטה יכולים לשהות בחדר באותו הזמן).

סמני נכון / לא נכון (אין צורך להסביר):

1. **(3 נק')** יכולים להיות שני סטודנטים מפקולטות שונות באותו חדר במקביל: **נכון / לא נכון**
2. **(3 נק')** יכולים להיות שני סטודנטים מפקולטות זהות בחדר במקביל: **נכון / לא נכון**
3. **(3 נק')** סטודנטי פקולטה אחת עלולים להרעיב (כניסת) סטודנטי פקולטה אחרת: **נכון / לא נכון**

בסעיפים הבאים מוצג קוד למימוש כניסה ויציאה של סטודנטים אל ומחדר מסוים, כאשר נתון כי:

- כל חוט מייצג סטודנט.
- בכניסה לחדר הסטודנט קורא ל onArrival(int faculty), שמקבלת את פקולטת הסטודנט.
- ביציאה מהחדר הסטודנט קורא ל onLeave(int faculty) שמקבלת את פקולטת הסטודנט.
- הערכים 0 ו-1 של faculty מייצגים את הפקולטה להנדסת חשמל ומדמ"ח, בהתאמה.
- (הניחו שאמצעי הסנכרון עברו אתחול תקין והתעלמו מבעיות קומפילציה אם ישנן, שכן מטרת השאלה אינה לבדוק שגיאות אתחול/תחביר).

```
1.  #include <pthread.h>
2.
3.  int students = 0;
4.  mutex_t global;
5.
6.  void onLeave(int faculty) {
7.          mutex_lock(&global);
8.          students--;
9.          mutex_unlock(&global);
10. }
```

```
11. void onArrival(int faculty) {
12.         mutex_lock(&global);
13.         while (students > 0) {
14.                 mutex_unlock(&global);
15.                 sleep(10);
16.                 mutex_lock(&global);
17.         }
18.         students++;
19.         mutex_unlock(&global);
20. }
```

4. **(12 נק')** בהתייחס לקוד הנ"ל, הקיפי את כל התשובות הנכונות (עשויה להיות יותר מאחת).
עבור כל תשובה שהקפת, תארי דוגמת הרצה המובילה לתשובה זו.

a. קיימת בעיית נכונות עקב race condition למשאבים משותפים.
b. קיימת בעיית DeadLock / Livelock בקוד.
c. הקוד משתמש ב-Busy Wait שפוגע בנצילות המעבד.
d. הקוד מפר את כלל הכניסה לחדר (שהוגדר בתחילת השאלה).

נימוק:

המימוש של כניסה ויציאה שונה כך שישתמש במשתני תנאי:

```
int students[2] = {0};      // 2 counters
cond_t conds[2];            // 2 condition variables
mutex_t global;
void onArrival(int faculty) {
        mutex_lock(&global);
        int other = faculty ? 0 : 1;
        while(students[other] > 0)
                cond_wait(&conds[faculty] , &global);
        students[faculty]++;
        mutex_unlock(&global);
}
void onLeave(int faculty) {
        mutex_lock(&global);
        students[faculty]--;
        int other = faculty ? 0 : 1;
        cond_broadcast(&conds[other]);
        mutex_unlock(&global);
}
```

אך דני (עתודאי במדמ"ח) טען שקוד זה גורם לחוטים להתעורר שלא לצורך ומיד לחזור למצב המתנה.

5. **(7 נק')** הסבירי את טענתו של דני באמצעות דוגמת ריצה קונקרטית.

6. **(8 נק')** כיצד ניתן לתקן את הבעיה שהציג דני בסעיף הקודם?

דני ניסה לשפר עוד את יעילות הקוד והחליט להשתמש בשני מנעולים: מנעול ראשון בעבור סטודנטים הנכנסים לחדר, ומנעול שני בעבור סטודנטים היוצאים מהחדר. להלן המימוש החדש (השינויים בקוד מודגשים):

```
int students[2] = {0};              // 2 counters
cond_t conds[2];                    // 2 condition variables
mutex_t m_arrival, m_leave;         // there are *2* locks now
void onArrival(int faculty){
        mutex_lock(&m_arrival);
        int other = faculty ? 0 : 1;
        while(students[other] > 0)
                cond_wait(&conds[faculty] , &m_arrival);
        int tmp = students[faculty];
        students[faculty] = tmp + 1;
        mutex_unlock(&m_arrival);
}
void onLeave(int faculty){
        mutex_lock(&m_leave);
        int tmp = students[faculty];
        students[faculty] = tmp – 1;
        int other = faculty ? 0 : 1;
        cond_broadcast(&conds[other]);
        mutex_unlock(&m_leave);
}
```

.7 **(12 נק')** בהתייחס לקוד הנ"ל, הקיפי את כל התשובות הנכונות (עשויה להיות יותר מאחת).
עבור כל תשובה שהקפת, תארי דוגמת הרצה המובילה לתשובה זו.

a. יתכנו 2 סטודנטים מפקולטות שונות בתוך החדר ביחד, עקב race condition למשאב משותף.
b. יתכן סטודנט שלא נכנס לחדר למרות כלל הכניסה שמתיר זאת, עקב race condition למשאב משותף.
c. קיימת בעיית DeadLock / Livelock בקוד.
d. סיגנלים עלולים ללכת לאיבוד.

נימוק: