Homework 3 Dry

Due Date: 17/06/2021 23:30

Teaching assistant in charge:

- Bar Raveh

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/aD9uB2P7wzGPDgqz8

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, תשובות ללא נימוק לא יתקבלו.

שאלה 1 - סנכרון בגרעין (30 נק')

שאלה זו דנה בסנכרון ובמנעולים בגרעין **שאינו ניתן להפקעה**.
דני, מפתח לינוקס, החליט להוסיף תמיכה בגרעין עבור מנעולים של קוד משתמש (user level). לשם כך, הוא הגדיר בגרעין מנעול יחיד (גלובלי) המשותף לכלל התהליכים הרצים במערכת, וכן קריאות מערכת מתאימות בכדי לאפשר לתהליכים לתפוס ולשחרר את המנעול:

```
volatile int global_lock = 0;
wait_queue_head_t global_queue;

int sys_global_lock() {
      while (global_lock != 0) {
            // atomically puts the process to sleep if
            // global_lock is locked:
            wait_event_interruptible(global_queue, &global_lock);
      }
      global_lock = 1;
}

int sys_global_unlock() {
      global_lock = 0;
      wake_up_interruptible(global_queue); // wakes up all
            // processes waiting in global_queue
}
```

למשל, שני חוטים שונים של אותו תהליך יכולים להשתמש בקריאות המערכת הנ"ל על-מנת לתאם ביניהם את הגישה למשתנה משותף X באופן הבא:

```
global_lock();   // wrapper function for sys_global_lock
X++;
global_unlock(); // wrapper function for sys_global_unlock
```

כמו כן, המנעול הגלובלי נגיש אך ורק לקריאות מערכת, כלומר אינו נגיש לשגרות טיפול בפסיקות וחריגות.

א. (6 נק') האם המימוש הנ"ל נכון? (כלומר אכן תומך בהפרדה הדדית עבור המשתנה המשותף X?)
במידה וכן - נמקו. במידה ולא - תארו במדויק תרחיש בעייתי שיכול להתרחש.

ב. (6 נק') לדעת דני, הקוד נכון במערכת שהינה בעלת ליבת מעבד יחידה. האם הוא צודק? נמקו.

כדי לשפר את התמיכה במערכת מרובת-ליבות, יוסי הציע להוסיף spinlocks לקוד של דני באופן הבא:

```
spinlock_t aux_lock;
int sys_global_lock() {
     spin_lock(aux_lock);
     while (global_lock != 0)
          wait_event_interruptible(global_queue, &global_lock);
     global_lock = 1;
}

int sys_global_unlock() {
     global_lock = 0;
     wake_up_interruptible(global_queue);
     spin_unlock(aux_lock);
}
```

ג. (6 נק') כאשר יוסי בחן את הקוד המשופר (על מחשב בעל מספר ליבות) המנעול אכן עבד נכון, אך דני טען שכשהריץ את אותו הקוד (על מחשב בן ליבה אחת) מערכת ההפעלה נתקעה. מה גרם לתקלה במערכת של דני? תארו במדויק תרחיש בעייתי שיכול היה להתרחש במערכת של דני.
הניחו כי קוד המשתמש שהריץ דני הוא חוקי ותקין (למשל, אין נעילה כפולה מאותו התהליך).

ד. (6 נק') האם המחשב של דני היה נתקע אילו היינו משתמשים בסמפור במקום ב-spinlock? (כלומר, חישבו על aux_lock כסמפור והחליפו את הקריאות ל-spin_lock/spin_unlock ב- sem_down/sem_up)?

כעת נדון בקוד שונה במעט אשר משתמש בשני המאקרו-ים הבאים:

- cond_wait - מקבל שלושה פרמטרים: (i) תור המתנה, (ii) מנעול מסוג mutex המספק מניעה הדדית, ו- (iii) תנאי בולאני. פעולתו של cond_wait היא אטומית: הוא בודק את התנאי הבוליאני ואם התנאי לא מתקיים משחררים את ה- mutex ונכנסים להמתנה (בשינה) על התור. כאשר cond_wait חוזרת, מתבצעת נעילה חוזרת של ה-mutex.
- cond_signal - מקבל תור המתנה ומעיר תהליך יחיד מתור זה.

המימוש הבא יוצר מנעול תקין:

```
Mutex m;   // initially unlocked

int sys_global_lock() {
     lock(m);
     while (global_lock != 0) {
          cond_wait(global_queue, m, (global_lock==0));
     }
     global_lock = 1;
     unlock(m);
}

int sys_global_unlock() {
     lock(m);
     global_lock = 0;          //Free lock
     cond_signal(global_queue);
     //Wakes a single process waiting in global_queue
     unlock(m);
}
```

דנה שמה לב שעל אף שהמימוש תקין, לעתים הוא עלול לגורם למצבי קיפאון בעקבות נעילה כפולה מאותו החוט (קריאה של מספר פעמים ברצף לפונקציה global_lock) או בעקבות שחרור המנעול ע"י חוט שלא נעל אותו.

ה.6) נק') הציעו שיפור למנעול כפי שהוא כתוב לעיל, המאפשר לאותו החוט לקרוא לפונקציית הנעילה כמה פעמים ברצף מבלי להיתקע (בכל נעילה נוספת לאחר הראשונה יוחזר הערך EPERM), ושעבור חוט שמנסה לשחרר מנעול שלא נעל בעצמו יוחזר ערך השגיאה EPERM. על הקוד שלכם להיות כתוב בשפת C. הקפידו על תיעוד שמנמק את הנכונות של הקוד שלכם.

שאלה 2 - גרעין ניתן להפקעה (70 נק')

הוסר סעיף ו.א וסעיף ז. (נשאר סעיף ו.ב, וסעיף ח' הפך להיות סעיף ז')
כפי שלמדנו בתרגולים, גרעין לינוקס 2.4 אינו ניתן להפקעה (non-preemptible).

א. 7 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. (7 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.

ה. (20 נק') הבעיה של נגן המוזיקה המקרטע נובעת מכך שגרעין לינוקס אינו ניתן להפקעה.
בהנחה שהמערכת מריצה יחד עם נגן המוזיקה קומפילציה של גרעין לינוקס, אילו מהצעדים הבאים יכול לצמצם את הבעיה? נמקו עבור כל אחד מהגורמים.

- הקטנת העומס על המערכת (פחות תהליכים שרצים יחד עם נגן המוזיקה).
- שימוש במעבד מהיר יותר.
- שימוש בדיסק מהיר יותר.
- מעבר למערכת עם יותר מעבדים.
- שיפור העדיפות של נגן המוזיקה ע"י הקטנת הערך nice.
- הפיכת נגן המוזיקה לתהליך זמן-אמת.

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...

ו. (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.

ז. (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.