יבש 3

שאלה 1:

א. המימוש הנ"ל לא נכון, ניתן דוגמא נגדית המסבירה זאת. נניח כי ישנם שני חוטים A,B המגיעים לקריאה global_lock() ביחד ומשם מתבצע מעבר למצב גרעין כאשר כל אחד מהם עובר לעבוד על מעבד שונה. ייתכן מצב שבו שני החוטים יגיעו לתנאי של לולאת ה-while לפני שהחוט השני יספיק לשנות את המשתנה הגלובאלי של המנעול לערך 1 ולכן שניהם יעברו את הלולאה וימשיכו כך שיחזרו למצב משתמש ביחד וישנו את ערך x בו זמנית.

ב. דני צודק. במערכת שבה מעבד ליבה יחידה רק תהליך אחד יוכל לרוץ בקרנל בנקודת זמן מסוימת ולכן לא ייתכן מצב שבו שני חוטים יריצו את פעולות הגרעין היכולות לפתוח ולסגור את המנעול במקביל ולכן לא ייגרם מצב של race condition בין אותם תהליכים.

ג. נתאר תרחיש בעייתי שהיה יכול להתקיים במערכת של דני ולהוביל למצב של תקיעת המערכת. נניח כי ישנם שני חוטים A,B כך ש-A מתחיל ראשון ומגיע לקריאה global_lock() ומשם לקריאת המערכת בגרעין sys_global_lock() שם הוא נועל את ה-spinlock ממשיך ויוצא חזרה לקוד המשתמש שם הוא מבצע את x++. לפני שהחוט A מספיק להגיע לקריאה לפונקציה global_unlock() חוט B מתחיל את פעולתו ומגיע לקראיה global_lock() ומשם לקריאת המערכת בגרעין sys_global_lock() שם הוא מגלה כי ה-spinlock נעול ומחכה ב-busywait לפתיחת המנעול אך חוט A אינו יכול להיכנס ולבצע את קריאת הגרעין sys_global_unlock() ולשחרר את ה-spinlock שכן המעבד בעל ליבה אחת והגרעין עובד ללא הפקעות וחוט B הוא זה שכרגע עובד בגרעין (תקוע ב-busywait), לכן הגענו למצב של deadlock ולתקיעה של המערכת.

ד. אם נשתמש בסמפור במקום ב-spinlock המחשב של דני לא היה נתקע אך מטרת הנעילה הייתה עדיין מושגת זאת מכיוון שסמפור בניגוד ל-spinlock אינו מבצע busy wait אלא מעביר את החוט להמתנה ובמקרה זה מוותר על ההחזקה בליבה היחידה של הגרעין וכך התהליך שנעל את המנעול יוכל תמיד לקבל גישה לריצה בגרעין על הליבה היחידה ולשחרר את המנעול.

.ה

```
Mutex m; // initially unlocked

pid_t global_lock_thread = -1; // global variable to keep the TID of the last thread
                                  who locked the global lock

pid_t global_lock_process = -1; // global variable to keep the PID of the process
                                  who locked the global lock

int sys_global_lock()
{
    lock(m);
    // if the last thread who locked the global lock is the current thread return error
    // code
    if (current->pid == global_lock_thread &&
       current->tgid == global_lock_process)  // check current process and thread are
                                              // the same ones who locked the lock
    {
         unlock(m);
         return EPERM;
    }
    while (global_lock != 0)
    {
         cond_wait(global_queue, m, (global_lock==0));
    }
    global_lock_thread = current->pid; // update current locking thread
    global_lock_process = current->tgid; // update current locking process
    global_lock = 1; // lock the lock
    unlock(m);
}

 int sys_global_unlock()
 {
    lock(m);
    If (current->tgid != global_lock_process || current->pid != global_lock_thread)
    {
         unlock(m);
         return EPERM;
    }
    global_lock = 0; // Free the lock
    global_lock_thread = -1; // update current locking thread to -1 which means the
                             // lock is now free and not locked by any thread
    global_lock_process = -1; // update current locking process to -1 which means
the                           // lock is now free and not locked by any process
    cond_signal(global_queue);
```

```
    unlock(m);
}
```

<u>שאלה 2:</u>

א. המפתחים הראשונים של לינוקס בחרו לממש גרעין שאינו ניתן להפקעה שכן כאשר יש מעבד יחיד ומתרחשת פסיקת חומרה בתוך פסיקת חומרה יכול להיווצר deadlock במידה והייתה מתאפשרת הפקעה של הגרעין כפי שלמדנו בתרגול, כאשר אין הפקעה נאלצת פסיקת החומרה המקוננת להמתין ולהיות מטופלת בהמשך. במערכת מרובת מעבדים היתרון מצטמצם שכן כעת ניתן לטפל בפסיקות חומרה מקוננות על ליבות שונות במקביל תוך הגנה על מידע משותף באמצעות spinlock.

ב. תהליך אינטראקטיבי עלול לסבול מגרעין שאינו ניתן להפקעה, שכן תהליך שכזה מבוסס על זמני ריצה קצרים על המעבד של התהליך לבין כך שהוא מחכה ל-I/O כלשהו לצורך הביצוע של הפעולות הללו (בזמן זה יכול לוותר על המעבד), דבר הדורש זמינות גבוהה של המעבד בשביל הזמן שבו המידע הרלוונטי יהיה זמין וניתן יהיה לבצע את הפעולות. במידה והגרעין אינו ניתן להפקעה גם לאחר קבלת המידע הדרוש לביצוע הפעולות ובמידה ותהליך אחר רץ בגרעין, לא יוכל התהליך האינטראקטיבי לחזור ולפעול עם המידע שהתקבל/נטען בזמן תגובה מספק דבר שיגרום לפגיעה בתגובתיות התהליך האינטראקטיבי.
תהליך חישובי אינו נפגע מגרעין שאינו ניתן להפקעה שכן הוא אינו דורש תגובתיות כלומר כל המידע הדרוש לו קיים מלכתחילה ולכן אין צורך להפקיע את הגרעין מביצוע החישובים הנדרשים.

ג. נחשב:

16 bits per sample x 44,100 samples per second = 705,600 bits per second x 2 channels = 1,411,200 bits per second of stereo.

64 kibibytes = 524,288 bits / 1,411,200 bits per second of stereo = 0.371 seconds

ד. אופיו של התהליך הוא אינטראקטיבי. לפי רוחב פס של 10Mib/s ניתן למלא את הבאפר ב-0.06 seconds (64Kib/10Mib) ולפי ההנחה שזמן הקריאה מהדיסק הוא הדומיננטי ביותר בפעולת נגן המוזיקה נוכל להגיד שלאחר 0.06 seconds התהליך יוותר על זמן המאבד ונדרש אליו שוב לאחר סיום הריקון של הבאפר, 0.371 seconds מתחילת נגינת הדגימות האחרונות, כלומר המעבד יספיק למלא את ה-buffer לפני שיידרש שוב לטעון ולנגן את התוכן בו.

ה. נתייחס לכל אחד מהגורמים:

- **הקטנת העומס על המערכת (פחות תהליכים שרצים יחד עם נגן המוזיקה)** – יכול לעזור שכן אם יש תהליכים בעליי עדיפות גבוהה יותר שזקוקים לגרעין הם יתפסו אותו לפני נגן המוזיקה וכך יגרמו לקטיעות. אם נוכל להיפטר מחלק מאותם תהליכים נקצר את העיכוב בזמן הגישה של הנגן לגרעין ובהתאם את העיכוב בשמע. במידה ואין תהליכים נוספים הזקוקים לגרעין הדבר לא יכול לסייע.
- **שימוש במעבד מהיר יותר** – יכול לעזור שכן יאפשר סיום טיפול מהיר יותר של כל תהליך הנכנס למצב גרעין ובהתאם לכך פינוי יותר מהיר של הגרעין לטיפול

בתהליכים אחרים. דבר זה יאפשר לתהליך הנגן לעבוד במצב הגרעין לעיתים יותר תכופות ולכן יקטין את העיכוב בשמע.

- **שימוש בדיסק מהיר יותר** – לא יכול לעזור שכן דיסק מהיר יותר יקצר אומנם את הזמן שהנגן יצטרך לגשת לקוד הגרעין אך הדבר לא ישנה עם תהליכים אחרים בעליי עדיפות גבוהה יותר קיימים במערכת ויקבלו גישה לקוד הגרעין לפני התהליך של הנגן ויעכבו בכך את הגישה שלו דבר שיגרום לאותם עיכובים בשמע.
- **מעבר למערכת עם יותר מעבדים** – יכול לעזור שכן יהיו יותר מעבדים שנוכל להקצות וכך בכל מעבד יהיו פחות תהליכים ש"יילחמו" על מצב גרעין כלומר יתאפשרו גישות יותר תכופות למצב גרעין לכל התהליכים כולל לתהליך של נגן המוזיקה דבר שיתבטא אצלו בקטיעות פחות תכופות.
- **שיפור העדיפות של נגן המוזיקה ע"י הקטנת הערך nice** – לא יכול לעזר שכן עדיפויות לתהליכים מאפשרות לחלק את זמן המעבד בצורה שונה בין תהליכים אך הדבר לא משנה במקרה זה שכן הגרעין לא ניתן להפקעה ולכן אין משמעות לחלוקה של זמן המעבד עליו.
- **הפיכת נגן המוזיקה לתהליך זמן-אמת** – יכול לעזור שכן הפיכת התהליך לתהליך זמן אמת תקבע אותו ככזה שנדרש לעמוד באילוצים קשיחים על זמן התגובה ללא תלות בעומס על המערכת וייכנס לתור הריצה של תהליכי זמן האמת וברגע שיזומן ממנו ירוץ עד סופו. במידה ונכניס את התהליך של הנגן לראש תור הריצה בעל העדיפות הגבוהה ביותר נאפשר לו בכל פעם לקבל עדיפות לכניסה לגרעין וכך הוא תמיד יתפוס אותו כשהוא צריך ללא זמן המתנה (לכל היותר יחכה לתהליך רגיל יחיד שיסיים את ריצתו על הגרעין ואז "יידחף" לגרעין לפני התהליכים הרגילים האחרים).

ו. שלושה חסרונות עיקריים בהצעה של אינגו:

1. בדיקות חוזרות האם יש תהליכים אחרים לעבור אליהם הן דבר שמבזבז זמן מעבד בפני עצמו.
2. מגדיל את התקורה על החלפות הקשר בין תהליכים בעקבות נקודות הבדיקה הללו, הדבר עלול לעלות בזמן מעבד יקר.
3. שימוש בהצעה זו יכול להקטין את ה-throughput שכן ייתכן שתהליכים שונים יגיעו ממש עד לסיומם אך לפני שיספיקו לסיים יעברו בנקודת מעבר לתהליך אחר וכך הלאה וכך הלאה כך שיצטברו תהליכים שנמנע מהם לסיים (למרות שהם מאוד קרובים לכך).

ז. א. שתי בעיות שעלולות להיווצר בגישה למשתנים המוגדרים פר-מעבד במידה ולא נמנע הפקעות:

1. ייתכן ועדכון המשתנה [()per_cpu_array[smp_processor_id בתהליך אחד במעבד רלוונטי לתהליך אחר שרץ במקביל אליו. במידה ונאפשר הפקעה של המעבד ייתכן ולפני עדכון אותו משתנה תתרחש הפקעה ונעבור לתהליך אחר אשר מצפה למשתנה מעודכן בניגוד למה שהתרחש בפועל (הדבר יכול לגרום להתנהגות לא צפויה של התהליך). לדוגמא נניח וקיים משתנה המוגדר פר מעבד וסופר את מספר התהליכים שהתחילו לרוץ על המעבד. נניח ותהליך מספר 1 התחיל לרוץ על המעבד וטרם ההגעה שלו לקוד המעדכן את המשתנה התרחשה הפקעה לתהליך 2. נניח ותהליך 2 רוצה להדפיס את מספר התהליכים שהתחילו לרוץ על המעבד אזי גודל מונה זה (המשתנה הגלובאלי של המעבד) אמור להיות 2 אך מכיוון שבתהליך 1 לא התרחש עדיין עדכון המונה הערך לא יהיה מעודכן וההדפסה בהתאם.

2. מכיוון שלמערכת ההפעלה אין שליטה על הקצאת מעבדים, לאחר הפקעה לא נוכל לדעת לאיזה מעבד יוקצה התהליך בעת שיגיע תורו להמשיך בפעולתו לכן במידה והתהליך משתמש ומעדכן משתנים של המעבד ומצפה בהמשך לעבוד איתם, במידה ותתרחש הפקעה בנקודה כלשהי לא מובטח כי התהליך יחזור לאותו מעבד עם המשתנים אותם הוא מצפה לראות ולכן כשייגש אליהם ייתקל בערכים שונים. נניח לדוגמא כי קיים משתנה לכל מעבד ששומר את תוצאת החישוב האריתמטי האחרון שבוצע בו. בהינתן תהליך A שמתחיל את ריצתו על מאבד 0 ומחשב כמה זה 2+2=4 ושומר את תוצאה זו על המשתנה של המעבד תוך ציפייה להשתמש בערך זה בהמשך באמצעות לקיחתו מהמשתנה הגלובאלי. במידה ולאחר החישוב תתרחש הפקעה של התהליך והוא יעבור למעבד אחר (שבו לצורך הדוגמא נניח ותהליך B כלשהו עשה חישוב אריתמטי שתוצאתו 6 ולא 4), בהמשך החישוב שלו התהליך A כבר יסתמך על ערך שהוא לא חישב כלומר הוא איבד את הערך שחישב ודבר זה יכול לפגוע במטרת תהליך A.

ב. בגרעין 2.4 לא הייתה בעיה בגישה למשתנים המוגדרים פר-מעבד שכן לא הייתה הפקעה של המעבד ולכן לא נוצרו כלל הבעיות שציינו לעיל מבחינת גישה למשתנים אלו.