תרגיל בית 5 - יבש 3

מבנה מערכות הפעלה

046209

מגישים:
דור משעלי 311510630
כריסטיאן שקור 208157826

שאלה 1.

נתון המימוש העקרוני הבא להגנה בעזרת מוניטור על גישה למשאב משותף, כך שתתאפשר גישה לתהליכון כותב יחיד או למספר כלשהו במקביל של תהליכונים קוראים. הסבירו מה לא נכון במימוש להלן.

Pseudo code functions:

wait(q) – wait in the queue until signaled
signal(q) – signal a thread waiting in queue

```
monitor mon(read_start, read_end, write_start, write_end)
{
    bool  busy = false;
    queue  q;

    read_start()  // Entry section for a read process
    {
        if (busy) wait(q);
    }

    read_end()   // Exit section for a read process
    {
        if (!queue.isEmpty(q)) signal(q);
    }

    write_start()   // Entry section for a write process
    {
        if (busy) wait(q);
        busy = true;
    }

    write_end()   // Exit section for a write process
    {
        busy = false;
        if (!queue.isEmpty(q)) signal(q);
    }

}
```

תשובה

במימוש המוניטור המוצג בתרגיל שתי בעיות. בעיה ראשונה, נתאר מצב בו תהליך A נכנס לתור בשביל קריאה. לפני הכניסה ל $section$ של $read\_start()$ המשתנה $busy = false$ ולכן לא נכנס ל - if . כרגע תהליך A קורא. באותו הזמן שתהליך A קורא תהליך B מנסה לכתוב, הוא יצליח לכתוב כי לא יכנס ל if. קורה מצב בעייתי. תהליך אחד מנסה לקרוא מידע בעוד שהשני כותב.

שאלה 2.

בשאלה זו נדון בבעיית הקוראים כותבים, נתון הפתרון הבא לבעיית הקוראים כותבים:

```
struct readers_writers {
      mutex m;
      mutex writers;
      int readers;

       init() {
            readers = 0;
            m = unlocked, writers = unlocked;
      }

      enter_reader() {
            lock(m);
            readers++;
            if(readers == 1)
                 lock(writers);
            unlock(m)
      }

      enter_writer() {
            lock(writers);
      }

      leave_reader() {
             lock(m);
             readers --;
             if(readers == 0)
                  unlock(writers);
             unlock(m)
      }

      leave_writer() {
            unlock(writers);
      }
};
```

א. האם הפתרון פותר בצורה נכונה את בעיית הקוראים כותבים? אם כן ציינו כיצד ,אם לא ציינו מדוע?

כן, הפתרון מאפשר לכמה קוראים במקביל להיות נגישים למשאב המשותף בעוד שהוא לא מאפשר לכותבים להיכנס . בנוסף בזמן שיש כותב אזי המנעול נעול מה שגורם לקוראים לא לקרוא.

ב. האם הפתרון מקיים הוגנות? אם כן ציינו כיצד, אם לא ציינו מדוע.

לא , הפתרון גורם להרעבה של כותבים , כי מספיק ויש קורא אחד במערכת אזי הפתרון ייתן עדיפות לקורא.

ג. ממשו פתרון לבעיית הקוראים כותבים, כאשר יש לתת לכותבים עדיפות. בפתרון יש להשתמש בשלד הנתון עליכם למצוא מימוש שעושה שימוש ב-mutex בודד (התומך בפעולות lock ו-unlock) ושני condition variables התומכים בפעולות

Wait(mutex m, condition cv)

Signal (condition cv)

Broadcast (condition cv)

כמו כן הינכם יכולים להשתמש במספר בלתי מוגבל של integers.

```
struct readers_writers {
        mutex m;
        cond writers;
        cond readers
        int readers_c;
        int writers_c;
        int writers_w;
        init() {
                readers = 0;
                m = unlocked;
                writers_c=0;
                writers_w=0;
        }
        enter_reader() {
                lock(m);
                readers_c++;
                while(writers_c>0 ||writers_w){
                        wait(readers,m);
                }
                unlock(m);
        }
        enter_writer() {
                lock(m);
                writers_w++;
                while(writers_c>0 ||readers_c>0 ){
                        wait(writers,m);
                }
                writers_c++;
                writers_w--;
                unlock(m);
        }
        leave_reader() {
                lock(m);
                readers_c --;
                if (writers_w){
                        signal(writers);
                }
                unlock(m)
        }
        leave_writer() {
                lock(m);
                writers_c--;
                signal(writers);
                brodcast(readers);
                unlock(m);
        }
};
```

ד. ממשו פתרון הוגן לבעיית הקוראים כותבים. בפתרון יש להשתמש בשלד הנתון עליכם למצוא מימוש שעושה שימוש בשלושה mutexes (התומכים בפעולות lock ו unlock) ומספר בלתי מוגבל של integers.
בסעיף זה ניתן להניח כי סדר נעילת המנעול זהה לסדר שבו משתחררים התהליכים ממנעול זה (מנעול ממומש על ידי תור כפי שראיתם בהרצאה).

```
mutex m;
mutex writers;
mutex queue;
int readers;
init() {
        readers = 0;
        m = unlocked, writers = unlocked, queue =unlock;
}
enter_reader() {
        lock(queue);
        lock(m);
        readers++;
        if(readers == 1)
                lock(writers);
        unlock(m)
        unlock(queue);
}
enter_writer() {
        lock(queue);
        lock(writers);
        unlock(queue);
}
leave_reader() {
        lock(m);
        readers --;
        if(readers == 0)
                unlock(writers);
        unlock(m)
}
leave_writer() {
        unlock(writers);
}
```

שאלה 3.

נתון המימוש בפסודו קוד הבא:

```
Monitor Mycounter
{
 int counterValue;
 int getValue() { return counterValue;}
 void setValue(int value){ counterValue = value; }
 void doubleValue(){counterValue += this.getValue();}
}
```

א. באילו מקרים עלול להיווצר deadlock בשימוש במחלקה הזו? רשמו דוגמת קוד המשתמשת במחלקה וגורמת ל-deadlock?

תשובה

בעת השימוש בפונקציית $void\ doubleValue()$ עלול להיווצר $deadlock$. נתאר דוגמת קוד המביאה למקרה זה. מכיוון שמדובר ב $monitor$ אז עבור כל פונקציה שנכנס אליה ב $class$ נבצע $lock$. נניח כי השתמשנו בפונקציה $void\ doubleValue()$, לכן נבצע $lock$ ואז בתוכה נקרא לפונקציה $getValue$ גם פה נבצע $lock$ מה שיגרום ל $deadlock$.

ב. הציעו פתרון לתיקון המחלקה עצמה (ללא הורדת ה monitor)

תשובה

נשתמש במשתמש באופן ישיר, ללא קריאה לפונקציה

$void\ doubleValue(){\ counterValue += counterValue;\ }$

שאלה 4.

ציינו האם הטענה נכונה או לא. הסבירו את תשובתכם.

א. כדי לסיים חוט של threads level user מומלץ לקרוא ל exit(0).

תשובה

לא נכון. קריאה ל $exit(0)$ תגרום לסיום התהליך כולו, סיום התהליך יביא לסיום שאר החוטים למרות שרצינו לסיים רק חוט אחד.

ב. במעבד בעל ליבה אחת, אפשר לממש מנעול בעזרת חסימת/אפשור פסיקות.

תשובה

נכון. חסימת פסיקות לפני קטע קריטי ינעל את התהליך ואפשור פסיקות בסיום יפתח את הנעילה. דבר זה מבטיח שהתהליך לא יופרע עד שלא יסתיים הקטע הקריטי ולא תהיה החלפת הקשר.

ג. במעבד מרובה ליבות, אפשר לממש מנעול בעזרת חסימת/אפשור פסיקות.

לא נכון. חסימת/אפשור פסיקות זה מקומי לליבה אחת. במעבד מרובה ליבות תתכן גישה לקטע קריטי של תהליך אחר.