

אפשר לממש סמפורים על ידי שימוש בתור של מיוטקסים:

```
using System;
using System.Threading;
public class MySemaphore
{
    private readonly Mutex mutex = new Mutex();
    private readonly int maxCount;
    private int currentCount;
    public MySemaphore(int initialCount, int maxCount)
    {
        this.currentCount = initialCount;
        this.maxCount = maxCount;
    }
    public void WaitOne()
    {
        while (true)
        {
            mutex.WaitOne();
            if (currentCount > 0)
            {
                currentCount--;
                mutex.ReleaseMutex();
                return;
            }
            mutex.ReleaseMutex();
        }
    }
    public bool Release(int releaseCount = 1)
    {
        mutex.WaitOne();
        if (currentCount + releaseCount > maxCount)
        {
            mutex.ReleaseMutex();
            return false;
        }
        currentCount += releaseCount;
        mutex.ReleaseMutex();
        return true;
    }
}
```

האלגוריתם של פטרסון יכול להיות מורחב ל3 תהליכים או יותר, פסודו קוד לדוגמה

```
// Shared variables
boolean flag[3] = {false, false, false}
int turn = 0
// Process i (where i is 0, 1, or 2)
While(true){
    flag[i] = true
    turn = i
    while ((flag[(i+1)%3] || flag[(i+2)%3]) && turn == i);
    // Critical section
}
```

```

flag[i] = false
// Remainder section
}

```

ניתן לראות שההרחבה ל3 יחסית דומה לאלגוריתם המקורי של פטרסון, כעת יש מערך של דגלים במקום שני דגלים, וההרחבה של משתנה התור ל3 אפשרויות. במהלך בדיקה לגבי הסעיף הזה נתקלנו בFilter algorithm, שהוא אלגוריתם קיים שמרחיב את אלגוריתם פטרסון ל3 ומעלה. לכן אנחנו יכולים להגיד בוודאות שאכן ניתן לעשות זאת.

ג.1

האלגוריתם של דקר הוא אלגוריתם שמטרתו פתרון של בעיית mutual exclusion.

יתרונות של דקר:

- לא תלוי בחומרה
- מבטיח mutual exclusion

חסרונות דקר:

- מוגבל לשני תהליכים
- פחות יעיל בגלל שיש busy-waiting
- יותר מסובך לבצע ולהבין

יתרונות של מיוטקסמפור:

- יכול להתמודד עם כמה תהליכים או תרדים
- יעיל יותר כי לא מתרחש busy-waiting
- קל יותר להבין ולבצע
- הרבה פעמים יש תמיכה ספציפית של החומרה לכן יעיל יותר

חסרונות מיוטקסמפור:

- צריך תמיכה של מערכת ההפעלה
- יכול לייצר overhead (הכוונה היא לכל מה שתהליך\תרד משכפל בעת יצירה) לתהליכים קריטיים קצרים באופן לא יעיל.

ד.1

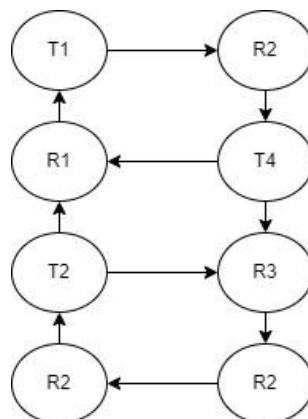
נקח לדוגמה מצב בו אנו מדמים מערכת בנק, ונניח שיש לנו 4 תרדים שכל אחת מטרתו שונה:

- T1 מעביר כסף
- T2 מעדכן את העו"ש בחשבון
- T3 מייצר דו"ח
- T4 מעבד בקשות הלוואה.

ונניח כי קיימים לנו ארבעת המשאבים הנ"ל אשר מתקשרים לתרדים שהצגנו למעלה:

- R1 בסיס נתונים של כל החשבונות
- R2 יומן טרנזקציות
- R3 בסיס נתונים לדוחות
- R4 מערכת דירוג אשראי

גרף אלוקציות שבו מתרחש דדלוק

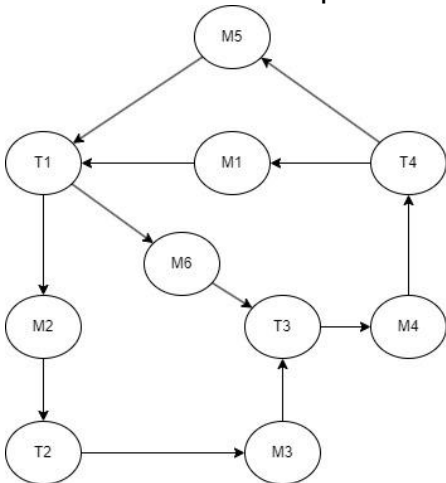


טבלאת אלוקציות:

Thread\Resource	R1	R2	R3	R4
T1	1	0	0	0
T2	0	1	0	0
T3	0	0	1	0
T4	0	0	0	1

ה.1

מצב של דדלוק עם 4 תרדים ושישה מיוטקסים



טבלאת אלוקציה

Thread\Mutex	M1	M2	M3	M4	M5	M6
T1	1	0	0	0	1	0
T2	0	1	0	0	0	0
T3	0	0	1	0	0	1
T4	0	0	0	1	0	0

2 SharableSpreadSheet:

א.

SharableSpreadSheet
-spreadSheet: ConcurrentDictionary<(int,int),string>
-nRows:int
-nCols:int
-searchSemaphore:SemaphoreSlim
-structureLock:ReaderWriterLockSlim
+methods

ב. בתוכנית יש מספר מנעולים –

1. אובייקט מסוג SemaphoreSlim שמטרתו להגביל את כמות החיפושים המקביליים. משומש במתודות SearchString, SearchInRow, SearchInCol, SearchInRange, FindAll. התוכנית והוא מאותחל בהתאם לפרמטר nUsers.
2. אובייקט מסוג ReaderWriterLockSlim שמטרתו להגן של שינויים במבנה הטבלה. משומש במתודות ExchangeRows, ExchangeCols, AddRow, AddCol ויש אחד כזה בכל התוכנית.

3. אובייקט מסוג `ConcurrentDictionary<int,int>,string` שמקיים מנעולים באופן מרומז (מעצם היותו מילון), הוא זה שמאפשר גישה לתאים ספציפיים בטבלה באופן שהוא `thread-safe`. משומש במתודות `set` ו-`get` ובאופן לא ישיר בכל מתודה אחרת שמנסה לגשת לתאים. האובייקט עצמו יחיד, אך הוא מקיים מנעול לכל זוג מפתח-ערך, ככה שיוכלו להשתמש במילון בו זמנית אבל בכל גישה ננעל התא הספציפי. לסיכום, בסך הכל 3 אובייקטים שמשמשים כמנעולים בתוכנית - `SemaphoreSlim` שמשמש להגבלת כמות החיפושים המקביליים ומיצוא המשאבים. `ReaderWriterLockSlim` שמאפשר קריאות מקבילות של הטבלה ומבטיח גישה בטוחה (מבחינת תרדים) לכל שינוי במבנה הטבלה. `ConcurrentDictionary` מאפשר נעילה באופן יעיל ובטוח לכל תא ספציפי בו בלי להגדיר באופן מפורש בקוד.