עבודה 3 מערכות הפעלה:

שאלה 1:

א. נדבר על שני סוגי זיכרון שזמינים עבור פרוסס וטארד – stack וheap,
לכל טראד יש את המקום שלו בstack אבל כל הטראדים המשותפים לאותו פרוסס יחלקו את הheap , בגלל זה טראדים נקראים "קלים יותר" בגלל שיש לכל אחד מהם את הstack שלו אבל יכולים לגשת למידע משותף.
בגלל שטראדים שבתוך הפרוסס חולקים את אותו מרחב כתובות כמו הפרוסס, הגישה בין הטראדים קלה יותר שזה יתרון. (מצד שני חיסרון בכך יכול להיות שכאשר יש בעיה בטראד אחד בתוך הפרוסס דבר זה יכול להשפיע על שאר הטראדים הנמצאים תחת אותו פרוסס).

ב. חסם עליון למהירות התוכנית כתלות במספר המעבדים:

$$SPEEDUP \leq \frac{1}{S + \frac{1-S}{N}}$$

S- חלק סיראלי של התוכנית
N- כמות המעבדים
SPEEDUP- מהירות מקסימאלית של תוכנית

עבור 30% ו70% וCORE 2:

$$SPEEDUP \leq \frac{1}{0.3 + \frac{1-0.3}{2}} = \mathbf{\frac{20}{13}}$$

עבור 30% ו70% וCORE 4:

$$SPEEDUP \leq \frac{1}{0.3 + \frac{1-0.3}{4}} = \mathbf{\frac{40}{19}}$$

עבור 30% ו70% וCORE 8:

$$SPEEDUP \leq \frac{1}{0.3 + \frac{1-0.3}{8}} = \frac{\mathbf{80}}{\mathbf{31}}$$

<u>עבור 30% ו70% ו64 CORE:</u>

$$SPEEDUP \leq \frac{1}{0.3 + \frac{1-0.3}{64}} = \frac{\mathbf{640}}{\mathbf{199}}$$

<u>עבור 8% ו92% ו2 CORE:</u>

$$SPEEDUP \leq \frac{1}{0.08 + \frac{1-0.08}{2}} = \frac{\mathbf{50}}{\mathbf{27}}$$

<u>עבור 8% ו92% ו4 CORE:</u>

$$SPEEDUP \leq \frac{1}{0.08 + \frac{1-0.08}{4}} = \frac{\mathbf{100}}{\mathbf{31}}$$

<u>עבור 8% ו92% ו8 CORE:</u>

$$SPEEDUP \leq \frac{1}{0.08 + \frac{1-0.08}{8}} = \frac{\mathbf{200}}{\mathbf{39}}$$

<u>עבור 8% ו92% ו64 CORE:</u>

$$SPEEDUP \leq \frac{1}{0.08 + \frac{1-0.08}{64}} = \frac{\mathbf{1600}}{\mathbf{151}}$$

ג. השימוש בטראד פול הופך תוכניות ליותר יעילות מכיוון שהוא :
save initiation time , less overhead
לדוגמה תוכנית שרצה יותר טוב עם בריכת טראדים:
יוטיוב- מקבל פניות ממשתמשים כל הזמן בכל המקרים זה אותה עבודה של שליחת סרטון מכיוון שזו אותה הפונקציה לא צריך כל הזמן ליצור טראד חדש.

שאלה 3:

א.

```
1 reference
class mySemaphore
{
    int size;
    static int count;
    Mutex[] mutexs;
    Thread[] myThreads;

    0 references
    public mySemaphore(int theSize)
    {
        size = theSize;
        count = 0;
        mutexs = new Mutex[size];
        myThreads = new Thread[size];
        for (int i = 0;i < size; i++)
        {
            mutexs[i] = new Mutex();
        }

    }
    0 references
    public void wait()
    {
        while (count >= size) { }
        int i = 0;
        for (i = 0;i < mutexs.Length; i++)
        {
            mutexs[i].WaitOne(1);
        }
        myThreads[i] = Thread.CurrentThread;
        count++;
    }

    0 references
    public void Release()
    {
        for(int i = 0; i < size; i++)
        {
            if(myThreads[i] == Thread.CurrentThread)
            {
                mutexs[i].ReleaseMutex();
                count--;
                break;
            }
        }
    }
}
```

ב. נשמור כמו במין עץ בינארי עבור כל node:

b[level,2node+1], b[level,2node], turn[level,node]

עבור כל פרוסס:

Level ,node ,id

Program for process i:

```
Node:=i
For level 0 to log n-1 do:
        Id=node mod 2;
        Node= [node/2]
         b[level,2node+id]
         turn[level ,node] = id
         await (b[level,2node+1-id] = false or turn[level ,node]=1-id)
end do
```

CS

```
For level = log n-1 down_to 0 do:
        Node = [ i / 2^level]
           b[level,2node+id]
end do
```

ג. נרשום פסאדו קוד עבור כל אחד מה statements :

1. אתחול ארבעה Semaphore עם ערך 0 ונקרא להם S1,S2,S3,S4

```
statement S1:
#code Statement 1
Signal(S1);

statement S2:
Wait(S1);
#code Statement 2
Signal(S2);

statement S3:
Wait(S2);
#code Statement 3
Signal(S3);

statement S4:
Wait(S3);
#code Statement 4
Signal(S4);

statement S5:
Wait(S4);
#code Statement 5
```

ד. deadlock מתאר סיטואציה שבה שני טראדים או יותר נחסמים לנצח וזאת מכיוון שהם מחכים אחד לשני, deadlock מופיע כאשר מספר טראדים צריכים את אותו מנעולים אך משיגים אותם בסדר שונה. תוכנית עם כמות טראדים גדולה עלולה לסבול ממצב של deadlock שמילת המפתח המסונכרנת גורמת לחסימת הטראד המבוצע בזמן ההמתנה למנעול או למוניטור המשויכים לאובייקט שצוין.

דוגמה לקוד עם deadlock:

```
1. Semaphore s1 = new semaphore(0,1)
2. Semaphore s2 = new semaphore(0,1)
3. Two processes P1 and P2;

P1:

4. Wait(s1)
5. Wait(s2)

P2:

6. Wait(s2)
7. Wait(s1)
```

הסבר: P1 מבצע את שורה ארבע ומיד לאחר מכן יש P2 context switch מבצע את שורה שש וכעת ישנו מצב של דד-לוק מכיוון שP1 מחכה למנעול שP2 תפס ואילו P2 מחכה למנעול שP1 תפס ואף פרוסס לא משחרר את המנעול שלו.

ה. השוואה בין שני האלגורתמים:

נקודות להשוואה	Dekkers	Petersonss:
כמות פרוססים מקסימאלית שעליהם ניתן לבצע את האלגוריתם	אי אפשר להרחיב אותו ליותר משני טראדים במקסימום ולכן במערכות גדולות דבר שיכול לגרום לאי נוחות.	מינימום 2 הכי הרבה n , דבר המקנה לו ייתרון.
מצב שבו פרוסס אחד מקבל יותר זמן באזור הקריטי	ייתכן מצב כזה	לא ייתכן מצב כזה
	כל טארד יכול לצאת לפועל רק בסנכרון מאוד קפדני	משתמש רק בזיכרון משותף לסינכרון רק פעולות הLODE והSTORE צריכות להיות אטומיות

שאלה 4:

בסעיף זה השתמשנו בסוג מנעול ששמו `ReaderWriterLockSlim` והוא עוזר לנו ליעל את ריצת התוכנית שלנו, מכיוון שהוא מאפשר להרבה קוראים לקרוא את אותו מסמך יחד ובעצם מפחית את העומס על אותו מנעול (עדין כל כותב צריך לקבל תור בנפרד).

החלטנו שעבור סכום מנעולים מינמאלי נשתמש במנעול עבור כל פונקציה וכאשר המשתמש יקרא לפונקציה הוא בעצם ינעל אותה ויוכל לעשות בה שימוש ללא חשש מכניסת טראדים אחרים השתמשנו באסטרטגיה זאת מכיוון שרצינו להשקיע את רוב הזמן על שאלה 5 ולעשות אותה בצורה המיטבית ולכן בחרנו לנעול את כל הפונקציות בשאלה זו.

שאלה 5:

א.הסבר:

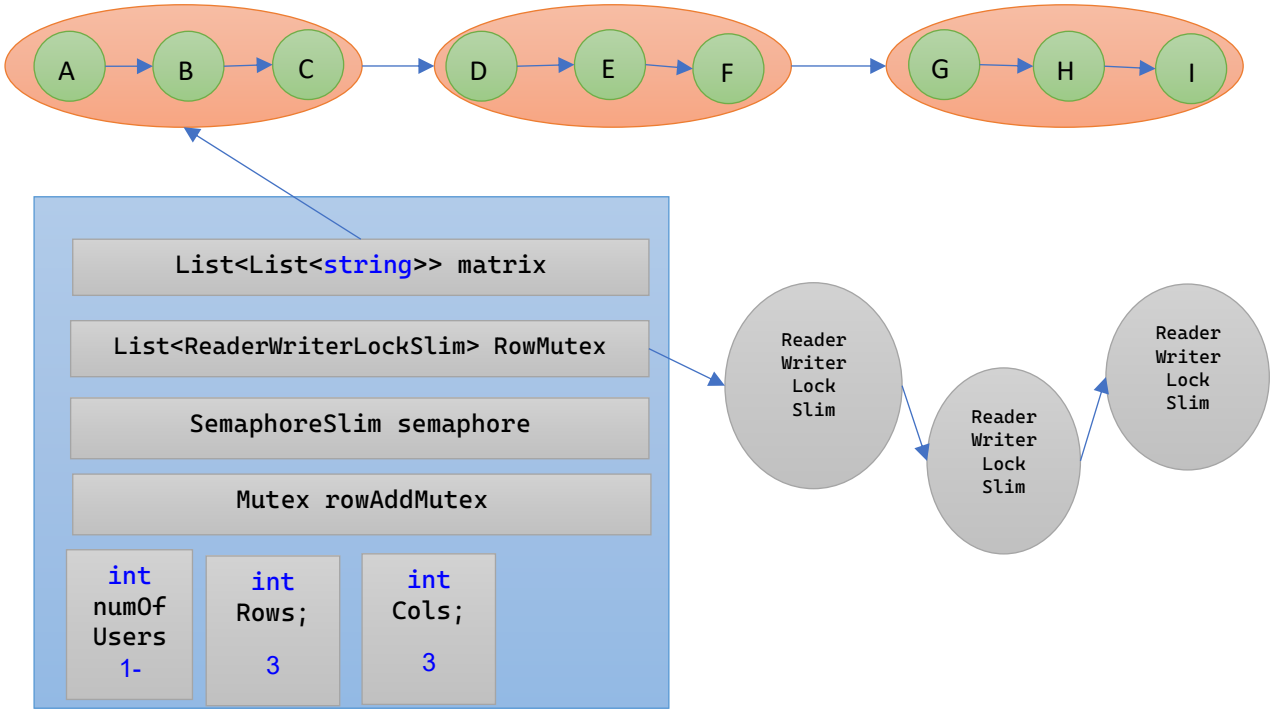
האובייקט שלנו מחולק לפי שורות כאשר אנו רוצים לבצע פעולות על האובייקט שלנו אנו תמיד נועלים אך ורק 2 שורות לכל היותר בכל הפונקציות (דבר זה עוזר לתוכנית שלנו לרוץ בצורה יעילה יותר מכיוון שאיננו נועלים את כל הפונקציה או את כל השורות וגם לא נועל גם עמודות וגם שורות - כל פעם נועל רק 2 שורות בכל התוכנית).
בנוסף אנו משתמשים בסוג מנעול ששמו `ReaderWriterLockSlim` והוא עוזר לנו ליעל את ריצת התוכנית שלנו, מכיוון שהוא מאפשר להרבה קוראים לקרוא את אותו מסמך יחד ובעצם מפחית את העומס על אותו מנעול (עדין כל כותב צריך לקבל תור בנפרד).

הכנסנו סמופור שאחראי על כל האובייקט מי נכנס אליו ומי יוצא ומתוחזק בתוך כל אחת מהפונקציות, הכנסנו עוד מנעול על חלק קטן מפונקציית ההוספת שורה כי רצינו להפוך חתיכת קוד מאוד קטנה לאטומית (נציין שהתוכנית שלנו לא נתקעת על זה מכיוון שחלק קוד זה קטן).

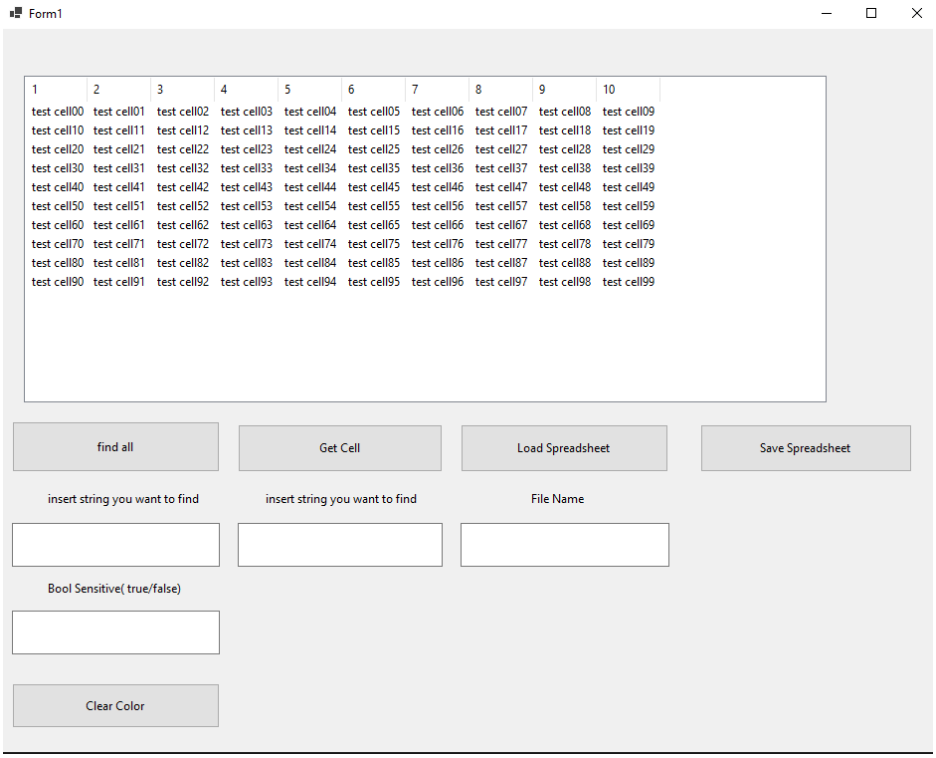
אנחנו משתמשים בכמות מפתחות ככמות השורות שלנו בתוכנית והיא מיוצגת לפי רשימה שאנחנו מחזיקים באובייקט שלנו.

למשל עבור הSpreadsheetApp הבא :

A	B	C
D	E	F
G	H	I



סעיף ג':



למשל הפעלת הפונקציה Get Cell אשר צובעת את החיפוש: