.1

```
class Semaphore
{
    Mutex m_resource;
    Mutex m_wait;
    int resource;
    int max_users;
    bool Flag;
    0 references
    public Semaphore(int n)
    {
        this.max_users = n;
        this.resource = n;
        this.Flag = false;
        this.m_resource = new Mutex();  // mutex to manage change in resource
        this.m_wait = new Mutex();  // mutex that will work if we are out of resources
    }
    0 references
    public void Wait()
    {
        this.m_wait.WaitOne();
        this.resource--;
        this.m_resource.ReleaseMutex();
        if(this.resource < 0)
        this.Flag = true;
        while(Flag){ }; // wait until able to enter
        this.m_wait.ReleaseMutex();
    }
    0 references
    public  void signal()
    {
        this.m_resource.WaitOne();
        if (this.resource < this.max_users)
            this.resource++;
        this.m_resource.ReleaseMutex();
        this.Flag = false;

    }
}
```

2. אפשרי להפוך את האלגוריתם של פטרסון לצורה בה הוא יעבוד עם 3 threads. נחלק את המצב לשלבים שבכל שלב יש שני סיבובים.
ראשית נשנה את משתנה Turn להיות מערך כך שיחזיק לנו עבור כל "סיבוב" מי האחרון שנכנס אליו. כאשר בכל "סיבוב" יתבצע תעדוף של ה-thread המתקדם ביותר. בנוסף נשנה את משתה Flag ל-int וכך נדע את מצבו של כל thread.

1) אתחול המערכים: $Flag = \{-1,-1,-1\}, Turn = \{-1,-1,-1\}$
2) סימון מס' thread ב- i (0,1,2)
3) עבור thread i:
   a. לכל k = 0 עד 1 (שני סיבובים)
   b. $Flag[i] = k$ – סמן שאתה במקום ה- k בשלב
   c. $Turn[k] = i$ - סמן שאתה האחרון שהגעת לסיבוב הK
   d. $-\ while\ ((Flag[(i+1)\%3] \geq k\ or\ Flag[(i+2)\%3] \geq k\ )And\ Turn[k] = i)$
   תשאר עד אשר יש thread אחד לפחות באותו מקום כמוך בשלב וגם אם אתה האחרון שהגיע לשלב הנוכחי.
4) כנס ל- Critical Section
5) לאחר סיום הפעולה $Flag[i] = -1$– תודיע שסיימת.

אלגוריתם זה עולה על שלושת דרישות הCS.

1. Progress – אם הטרד הוא היחיד שמחכה להיכנס לCS אז הוא יכנס מכיוון שהוא לא יתקע בלולאת הwhile.
2. Bounded Waiting – אם קיים טרד המחכה להיכנס לCS משמע יש טרד אחר בCS. שהטרד השני יסיים את פעולותיו בCS אחד אחר יכנס. במידה וזה לא הטרד הראשון, אלא הטרד השלישי, בשנייה שהשלישי יסיים את פעולותיו בCS הראשון יכנס.
3. Mutual Exclusion – אם קיים טרד בCS אף טרד אחר לא יכנס לשפ באותו הזמן. מכיוון שבכל סיבוב כל טרד מסמן שהוא נכנס אליו ומוסיף על כך שהוא האחרון שנכנס, ולכן יתקע בלולאת while. כאשר טרד נוסף יגיע לסיבוב שלו, הוא יקדם את הטרד לסיבוב הבא בתנאי שהטרד המקודם לא נכנס לסיבוב זה אחרון. כלומר הטרדים האחרי דוחפים את הטרד הראשון בסיבוב להתקדם קדימה לבד.