

Operating Systems – 234123

Homework Exercise 3 – Dry

Teaching Assistant in charge:

Mohammad Agbarya

Assignment Subjects & Relevant Course material

Threads, Synchronization, pthread Library

1. Only typed submissions in PDF format will be accepted. Scanned handwritten submissions will not be graded.
2. The dry part submission must contain a single PDF file named with your student IDs –
123456789_300200100.pdf
3. The submission should contain the following:
 - a. The first page should contain the details about the submitters - Name, ID number and email address.
 - b. Your answers to the dry part questions.
4. Submission is done electronically via the course website, in the **HW3 Dry** submission box.

Grading

1. All question answers must be supplied with a full explanation. Most of the weight of your grade sits on your explanation and evident effort, and not on the absolute correctness of your answer.
2. Remember – your goal is to communicate. Full credit will be given only to correct solutions which are clearly described. Convolved and obtuse descriptions will receive low marks.

Questions & Answers

- The Q&A for the exercise will take place at a public forum Piazza **only**. Please **DO NOT** send questions to the private email addresses of the TAs.
- Critical updates about the HW will be published in **pinned** notes in the piazza forum. These notes are mandatory and it is your responsibility to be updated.

A number of guidelines to use the forum:

- Read previous Q&A carefully before asking the question; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You're not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding **hw3-dry**, put them in the **hw3-dry** folder.

Late Days

- Please **DO NOT** send postponement requests to the TA responsible for this assignment. Only the **TA in charge** can authorize postponements. In case you need a postponement, please fill out the attached form:

<https://docs.google.com/forms/d/171TPPZSpFGlxBt2bGK7zU-Id4rf6B4LYTgKuOqEJSgo>

הנחיות בנוגע לתרגיל הבית הנוכחי:

1. שימו לב, הקוד הנתון בחלק מקטעי הקוד אינו קוד פורמלי, ולכן אין להתייחס בפתרונכם לבעיות קומפילציה כאלה או אחרות. יש לזהות את מהות השאלה ולענות לפיה.
2. יש להסביר כל סעיף עליו אתם עונים. הסבר שכזה תורם לכם להבין יותר טוב את התרגיל, ותורם לנו בלהבין יותר טוב את פתרונכם. מרבית הניקוד יינתן על סמך הסבר זה.

חלק ראשון (30 נק'): זיהוי כשלי סנכרון

תזכורת: תכונות הקטע הקריטי

תכונות הכרחיות:

1. **Mutual Exclusion – מניעה הדדית** – בכל רגע נתון, לא יכול להיות יותר מחוט אחד בתוך הקטע הקריטי (הדבר שקול לכך שהקטע הקריטי הופך לנקודת **סריאליזציה** במסלולי הביצוע).
2. **Progress – התקדמות** – אם יש חוטים שרוצים לבצע את הקטע הקריטי, לבסוף חוט **כלשהו** יצליח להיכנס. ישנה התקדמות – אין Deadlock/Livelock.

תכונות רצויות:

3. **Fairness – הוגנות** – אם יש חוט **שרוצה** לבצע את הקטע הקריטי, **הוא** לבסוף יצליח. אין הרעבה. הרחבות על הדרישה:
 - **Bounded Waiting – הגדרת סוף** למספר הפעמים שחוטים אחרים ייכנסו לקטע הקריטי לפני החוט הנוכחי.
 - **Order** – יש סדר ברור וידוע לזמני הכניסה של החוטים הנכנסים לקטע הקריטי. דוגמה לסדר אפשרי: FIFO.

1.

1. (5 נק') אילו תכונות של הקטע הקריטי מפר המימוש הבא כאשר משתמשים בו במערכת עם נפילות חוטים, ולמה? הניחו שהקוד רץ על מעבד יחיד.
נפילת חוטים: חוט יכול ליפול באופן פתאומי, כתוצאה מחריגה למשל.

Atomic Swap מקבלת כתובת של תא וערך חדש, ומחליפה **באופן אטומי** את תכולת התא עם הערך החדש, ומחזירה את הערך הישן.

2. (5 נק') במימוש קיימת בעיית Performance, הגורמת לחוסר יעילות של זמן המעבד. ניתן להניח שהקטע הקריטי עליו המנעול מגן הינו קטע ארוך וכבד חישובית. היכן היא? האם הבעיה עדיין קיימת אם הקטע היה קצר ומהיר?

```
class lock {
    bool lockVal;
public:
    lock(bool initVal) { lockVal = initVal;}
    void lock(){
        while(AtomicSwap(&lockVal,0)==0){}
    }
}
```

```

void unlock(){
    lockVal = 1;
}

```

2. (5 נק') מממשים מנעול חדש הכולל:

- Mutex סטנדרטי
- מונה count
- סף איטרציות MAX_ITER

תיאור מנגנון הנעילה: בזמן ניסיון נעילה (דהיינו, קריאה לפונקציה lock()), המנעול תומך ב-Timeout אותו ממשים על ידי מונה בצורה הבאה: במידה והחוסים במערכת מנסים לתפוס את המנעול MAX_ITER פעמים, אך המנעול אינו שוחרר במהלך ניסיונות אלו, המנעול ישוחרר. שימו לב ש-MAX_ITER הינו define גלובלי הידוע לכל החוסים. תיאור בפסודו קוד של מימוש הפונקציה lock() נתון בקטע הקוד הבא:

```

while ( mutex is locked ) {
    if ( mutex wasn't released yet )
        count++
    else
        count=0
    if ( count == MAX_ITER )
        unlock mutex
}

```

הניחו מערכת עם מעבד יחיד ואפשרות לנפילת חוסים פתאומית. הניחו שה-Mutex מומש בעזרת תור ושומר על סדר הכניסות אליו (FIFO). זהו אגב, נקרא מנעול "הוגן". אילו תכונות של הקטע הקריטי מופרות פה?

3. (5 נק') בהנחה שהקוד מורץ על מעבד יחיד, הסבר מה ידפיס הקוד הבא, ולמה? התשובה צריכה להיות מורכבת מערך מקסימלי אפשרי וערך מינימלי אפשרי, עם תרחיש אפשרי לכל אחד. ניתן להניח שפעולות load ו-store מתבצעות באופן אטומי.

```

int sum=0;
pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
int ids[10]={1,2,3,4,5,6,7,8,9,10};

void* thread_workload(void *threadID){
    int* p_val = (int*) threadID;
    pthread_mutex_lock(&mutex);
    sum += *p_val;
}

```

```

        pthread_mutex_unlock(&mutex);
    }

int main(){
    pthread_t t;
    int i;
    for(i=0;i<10;++i)
        pthread_create(&t,NULL, thread_workload,(void*)(ids+i));
    pthread_join(t,(void**)&i);
    printf("%d\n", sum);
    return 0;
}

```

4. (5 נק') בהנחה שהקוד מורץ על מעבד יחיד, הסבר מה ידפיס הקוד הבא, ולמה? התשובה צריכה להיות מורכבת מערך מקסימלי אפשרי וערך מינימלי אפשרי, עם תרחיש אפשרי לכל אחד. ניתן להניח שפעולות load ו store מתבצעות באופן אטומי.

```

int result;
void* do_calc();
int i;
for(i=0; i<100 ; ++i)
    result=result+1;

int main(){
    pthread_t threads[2];
    int i;
    result =0;
    for(i=0;i<2;++i)
        pthread_create(&threads[i],NULL,do_calc,NULL);
    for(i=0;i<2;++i)
        pthread_join(threads[i],NULL);
    printf("%d\n", result); return 0;
}

```

5. (5 נק') הסבירו למה אין צורך להגן על sum בעזרת משתנה סנכרון כמו Mutex או Semaphore. הניחו ש-sum הינו משתנה גלובלי.

```

int sum = 0;

if( fork() ) {
    sum = sum+5;
}

```

```
} else {  
    sum = sum +1;  
}
```

חלק שני (30 נק'): Singlephore

לרוב מנגנוני הסנכרון עליהם למדתם, קיימים לפחות שתי פעולות. מנעולים פשוטים תומכים ב-lock ו-unlock. משתני תנאי תומכים ב-wait ו-signal, וסמפורים ב-up ו-down או בשם המקורי בספרות, P ו-V. בתרגיל זה תעבדו עם מנגנון סנכרון שלו תמיכה רק בפעולה אחת ויחידה, ונקרא – singlephore.

הגדרת פעולות של המנגנון:

```
typedef struct singlephore {
    int value;
} singlephore;

// Initialize the singlephore to value 0.
void singlephore_init(singlephore * h) {
    h->value = 0;
}

// Block until the singlephore has value >= bound, then atomically increment its
// value by delta.
void H(singlephore * h, int bound, int delta) {
    // This is pseudocode; a real singlephore implementation would block, not
    // spin, and would ensure that the test and the increment happen in one
    // atomic step.
    while (h->value < bound) {
        sched_yield();
    }
    h->value += delta;
}
```

ברגע שה-singlephore אותחל, קוד אפליקציה יגש אליו רק דרך הפעולה H.

1. (10 נק') ממש מנעול למניעה הדדית בעזרת singlephore. מלא את תבניות הקוד הבאות:

```
typedef struct mutex {
    singlephore h;
} mutex;

void mutex_init(mutex* m) {
    //TODO
}

void mutex_lock(mutex* m) {
    //TODO
}

void mutex_unlock(mutex* m) {
    //TODO
}
```


2. (15 נק') ממש משתנה תנאי בעזרת singlephore ו-mutex (שכבר מימשתם). מלא את תבניות הקוד הבאות: (שימו לב, הסעיף הבא יכול לעזור לפתרון סעיף זה).

```
typedef struct condvar {
    mutex m;
    singlephore h;
    //TODO
} condvar;

// Initilize the condition variable
void cond_init(condvar* c) {
    //TODO
}

// Signal the condition variable
void cond_signal(condvar* c) {
    //TODO
}

// Block until the condition variable is signaled. The mutex m must be locked by the
// current thread. It is unlocked before the wait begins and re-locked after the
// wait
// ends. There are no sleep-wakeup race conditions: if thread 1 has m locked and
// executes cond_wait(c,m), no other thread is waiting on c, and thread 2 executes
// mutex_lock(m); cond_signal(c); mutex_unlock(m), then thread 1 will always
// recieve the
// signal (i.e., wake up).
void cond_wait(condvar* c, mutex* m) {
    //TODO
}
```

רמזים:

1. אם אין חוט שמחכה על משתנה התנאי c, אז cond_signal(c) לא יעשה דבר.
2. הנח ש-N חוטים ממתינים על משתנה התנאי c. אז N קריאות ל-cond_signal(c) הם תנאי הכרחי ומספיק על מנת להעיר את כולם.
3. יתכן ותוכל להיעזר בסעיף הבא כדי למצוא את הפתרון הנכון
4. ניתן ורצוי להשתמש בקבוע INT_MIN, הערך הנמוך ביותר ש-integer יכול לקבל.

3. (5 נק') ג'ון סנו החרוץ מתלמידי הקורס, סיפק את הפתרון הבא לסעיף ב':

```
typedef struct condvar {
    singlephore h;
} condvar;

void cond_init(condvar* c) {
    singlephore_init(&c->h);
}
```

```
void cond_signal(condvar* c) {  
    H(&c->h, INT_MIN, 1);  
}  
  
void cond_wait(condvar* c, mutex* m) {  
    mutex_unlock(m);  
    H(&c->h, 0, -1);  
    mutex_lock(m);  
}
```

מה לא תקין בפתרון? הראו תרחיש אפשרי בו פתרון זה לא עומד בתנאים של סעיף ב'.

חלק שלישי (40 נק'): Barrier

פרמטרים שנשתמש בהם לאורך כל השאלה:

```
1 n = the number of threads
2 count = 0
3 mutex = Semaphore(1)
4 barrier = Semaphore(0)
```

אחרי התרגול על סנכרון סטודנט חרוץ החליט שהוא ממש barrier באמצעות שני סמפורים, הוא הציע את הפתרון הבא:

```
1
2
3 mutex.wait()
4     count = count + 1
5 mutex.signal()
6
7 if count == n: barrier.signal()
8
9 barrier.wait()
10
11 critical point
```

1) (5 נק') הסבירו היטב למה הפתרון שלו לא עובד, הדגימו שימוש לא נכון במימוש הזה כ-barrier והראו מאיפה הבעיה נובעת.

אחרי שהסטודנט גילה שהפתרון שלו לא עובד הוא החליט לעשות כמה שינויים, הוא הציע את הפתרון הבא:

```

1
2
3 mutex.wait()
4     count = count + 1
5     if count == n: barrier.signal()
6
7     barrier.wait()
8     barrier.signal()
9 mutex.signal()
10
11 critical point

```

(2) (5 נק') הסבירו היטב למה הפתרון החדש שלו לא עובד, הדגימו שימוש לא נכון במימוש הזה כ- barrier והראו מאיפה הבעיה נובעת.

הסטודנט הבין לבד את הבעיות הנ"ל וכתב פתרון עובד סופסוף, אבל הוא שם לב שהמימוש שלו הוא חד פעמי כלומר לא יעבוד אם נשתמש בו פעם שנייה (not-Reusable).
לפתור את הבעיה הזו (Reusability) הסטודנט הציע את הפתרון הבא:

```

1
2
3 mutex.wait()
4     count += 1
5 mutex.signal()
6
7 if count == n: barrier.signal()
8
9 barrier.wait()
10 barrier.signal()
11
12 critical point
13
14 mutex.wait()
15     count -= 1
16 mutex.signal()
17
18 if count == 0: barrier.wait()

```

(3) (5 נק') הסבירו היטב למה הפתרון שלו לא עובד, הדגימו שימוש לא נכון במימוש הזה

כ-barrier-reusable והראו מאיפה הבעיה נובעת.

אחרי שהסטודנט גילה שהפתרון שלו לא עובד הוא החליט לעשות כמה שינויים, הוא הציע את הפתרון הבא:

```
1
2
3 mutex.wait()
4     count += 1
5     if count == n: barrier.signal()
6 mutex.signal()
7
8 barrier.wait()
9 barrier.signal()
10
11 critical point
12
13 mutex.wait()
14     count -= 1
15     if count == 0: barrier.wait()
16 mutex.signal()
```

4) (5 נק') הסבירו היטב למה הפתרון החדש שלו לא עובד, הדגימו שימוש לא נכון במימוש הזה כ-barrier-reusable והראו מאיפה הבעיה נובעת.

הסטודנט הבין לבד שצריך סמפור נוסף כדי לממש barrier-reusable.
5) (20 נק') הציעו לסטודנט ממוש עובד בסגנון הנ"ל.
