

# Homework 3 Dry

**Due Date: 1/1/2019 23:30**

Teaching assistant in charge:

1. **Ido Imanuel**

**Important:** the Q&A for the exercise will take place at a public forum Piazza only. 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**, put them in the **hw3** folder

Only the TA in charge can authorize postponements. In case you need a postponement, please fill out the following form: <https://goo.gl/forms/B6hvrILQ2RK88VTH3>

Dry part submission instructions:

1. Please submit the dry part to the electronic submission of the dry part on the course website.
2. The dry part submission must contain a single dry.pdf file containing 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.
3. Only typed submissions will be accepted. Scanned handwritten submissions will not be accepted.
4. Only PDF format will be accepted.
5. You do not need to submit anything in the course cell.
6. When you submit, **retain your confirmation code and a copy of the PDF**, in case of technical failure. It is **the only valid proof** of your submission.

### הנחיות בנוגע לתרגיל הבית הנוכחי:

- א. שימו לב, הקוד הנתון בחלק מקטעי הקוד אינו קוד פרומלי, ולכן אין להתייחס בפתרונכם לבעיות קומפילציה כאלה או אחרות. יש לזהות את מהות השאלה ולענות לפיה.
- ב. 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 ינתן על סמך הסבר זה.**
- ג. חלקו האחרון של תרגיל הבית היבש מיועד לפתרון **לאחר** פתרון של החלק הרטוב של תרגיל בית זה. מאידך, לאלו מכם שמסתבכים עם הרטוב, יתכן וניתן להיעזר בחלק זה עבור רמזים.
- ד. התשובות אותן אנו מחפשים בשאלות "מה יודפס" לא בהכרח יתקבלו עם הרצה אמיתית של הקוד על מחשבכם, כי בשאלה אין אנו נותנים נתונים מפורשים על אורך ה-Time Slice למשל, דבר הקבוע ויחיד אצלכם במחשב האישי. קיימים כמובן שיקולים נוספים. אנו מחפשים תשובה תיאורטית בלבד, עם הסבר מבוסס.
- ה. **ניקוד בונוס:** מטרת תרגיל הבית (יבש ורטוב) היא להציג בפניכם את עולם הסנכרון. זהו עולם רחב ועמוס בתובנות, ולעיתים (כמו למשל, backtracking ממבוא למדמ"ח) קשה להבין היטב את האלגוריתמיקה מאחוריו. חשוב אפוא שתתאמנו על כך, כך שחווית הבחינה תעבור באופן חלק. מכאן, אנו כן ממליצים להתמודד עם סעיפי **כתיבת** מנגנוני הסנכרון בחלק היבש, למרות שמטלה זו קשה יותר **מההבנה** של מנגנוני סנכרון מקולקלים. בגלל הקושי היחסי, סעיפים אלו סומנו כסעיפי בונוס. הניקוד המקסימלי הניתן לצבור בחלק היבש של תרגיל בית זה הוא **114**. במידה וציונכם בתרגיל היבש עלה מעבר ל-100, גם התוספת תשוקלל במיצוע הכללי של שיעורי הבית.

**חלק ראשון: 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 **בלשהו** 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 חסם** 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.

- א. אילו תכונות של הקטע הקריטי מפר המימוש הבא כאשר משתמשים בו במערכת עם נפילות חוטים, ולמה? הניחו שהקוד רץ על מעבד יחיד.  
**נפילות חוטים:** חוט יכול ליפול באופן פתאומי, כתוצאה מחריגה למשל.
- ב. 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(AtomicCompareAndSwap(&lockVal,0)==0){}
    }
    void unlock(){
        lockVal = 1;
    }
}
```

2. ממשים מנעול חדש שעובד כדלקמן. בזמן ניסיון נעילה, 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 ה-Timeout נתון בקטע הקוד הבא:

```
while( mutex is locked ) {  
    if (mutex wasn't released yet)  
        count++;  
    else  
        count =0;  
    if( cnt == 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. בהנחה שהקוד מורץ על מעבד יחיד, הסבר מה ידפיס הקוד הבא, ולמה ?

```
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. 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 store ו-load מתבצעות באופן אטומי (זוהי הנחה בקורס בנוגע לכל פקודות האסמבלר למיניהן)

```
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. הסבירו למה אין צורך להגן על sum בעזרת משתנה סנכרון כמו Mutex או Semaphore.

```
int sum = 0;

if(fork()) {
    sum = sum+5;
} else {
    sum = sum +1;
}
```

### חלק שני: 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.

א. 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
}
```

ב. סעיף בונוס (7 נקודות): ממש משתנה תנאי בעזרת singlephore ו-mutex (שכבר מימשתם). מלא את תבניות הקוד הבאות: (שימו לב, הסעיף הבא אינו סעיף בונוס, אך 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.

ג. ירמיהו 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: ניתוח של החלק הרטוב

חלק זה מבוסס על חלקו הרטוב של תרגיל בית 3, ומיועד לפתרון לאחר סיום חלק זה. במידה והסתבכתם, ניתן גם להיעזר בחלק זה לשם פתרון החלק הרטוב.

1. פיראס החרוץ מתלמידי הקורס ביסס מנגנון סנכרון בין Producer-Consumer שלו הוא קרא "Barrier":

```
class Barrier {
private:
    int working;
public:
    Barrier(){
        working =0;
    }
    increase(){
        working++;
    }
    decrease(){
        working--;
    }
    wait(){
        while(working!=0){}
    }
};
```

השימוש במנגנון היה כדלקמן:

Producer:

1. Init Barrier b
2. Init PCQueue p
3. Init fields *curr, next*
4. for  $t=0 \rightarrow t=n\_generations$   
    for  $i=0 \rightarrow i=M$   
        p.push(job);  
        b.increase();  
    b.wait();  
    swap(*curr, next*);

Consumer (One of N)

- ```
while(1)
    job j= p.pop() // blocked here if queue is empty
    execute j
    b.decrease();
```

**הנחות:**

1. חלק מהפסודו קוד שניתן לכם במסגרת התרגיל הרטוב הושמט. השאלה מתייחסת רק למנגנון הסנכרון.
2. התור מעלה הינו אותו תור יצרן-צרכן שהתבקשתם לממש בתרגיל הרטוב.
3. job הינו struct אשר מתאר לחוט כלשהו את גבולות הגזרה עליהם עליו לרוץ.
4. M הינו מספר העבודות הכולל שיש לחשב ב- *t* generation.

- א. הסבירו את כוונותיו של פיראס – איך היה אמור המנגנון לעבוד?
- ב. במימוש זה מספר בעיות Correctness.
- a. מצאו בעיה אחת של Race Condition בפתרון. הסבירו.
  - b. תארו תרחיש שבו מופר ה-Mutual Exclusion. דהיינו, חישוב הלוח curr טרם הסתיים, וה-Producer מבצע למרות זאת את ה-swap של הלוחות.
  - c. תארו שני תרחישים שונים בהם יתכן Deadlock בפתרון.
- ג. תקנו את class Barrier ואת הפסודו קוד של היצרן-צרכן כך שכל בעיות ה-Correctness יפתרו. אין לשנות את מתווה הפתרון של פיראס באופן מהותי ואין להשתמש בפעולות אטומיות.
- ד. בהתייחסות לסעיף ג: מנגנון סנכרון זה נקרא "מונה משותף", ואינו מוצלח במיוחד ממבט של בביצועים – Performance. הסבר מדוע.
- ה. **בונוס (7 נקודות):** בנו Barrier אחר, המבוסס על שני מערכים בינריים של באורך N, הפותרים יחדיו את בעיית הסנכרון. על המימוש לא להשתמש במנעולים מכל סוג, אך יכול לנצל את הפעולה האטומית CompareAndSwap, הנתונה לכם מטה. הניחו שפעולה זו ממומשת בחומרה, ולא בתוכנה, כיאה לפעולות אטומיות אחרות. הסבירו למה פתרון זה עדיף מבחינת Performance.

```
int CAS(int *ptr,int oldvalue,int newvalue)
{
    int temp = *ptr;
    if(*ptr == oldvalue)
        *ptr = newvalue
    return temp;
}
```

## 2. ביצועים וחוק אמדל

נניח אלגוריתם A המורכב ממספר רב של עבודות  $J_1, J_2, \dots, J_M$  לנחיתכם, מושגים נפוצים לניתוח Performance של האלגוריתם:

- **Latency(A)** – זמן החישוב הכולל של האלגוריתם A
- **Latency(j)** – זמן החישוב הכולל של עבודה j
- **Turnaround Time(j)** – זמן החישוב + זמן ההמתנה בתור של עבודה j
- **Throughput – תפוקה** – מספר העבודות המסתיימות ביחידת זמן

**בשאלה זו יש לצייר מספר גרפים, ולהסבירם. על הגרפים לכלול כותרת, מקרא, שמות צירים והסבר קצר על מה התקבל בגרף ולמה.**

א. הניחו ש-A ניתן למקבול באופן מלא. ציירו גרף של ה-Latency(A) כתלות במספר החוטים N, עפ"י חזונו של אמדל (Amdahl)

ב. הוסיפו לגרף בסעיף (א) מעלה עקומים המתארים גם A סריאלי לחלוטין, ו-A המורכב מחלק של  $s = \{0.25, 0.5, 0.75\}$  שניתן למקבול.

ג. עתה נתפנה לנתח את התוצאות המתקבלות מהאלגוריתם שכתבתם בחלקו הרטוב של התרגיל. נגדיר את A כחישוב לוח משחק יחיד, פעולה הנעשת ע"י N חוטים במקביל. נגדיר חישוב כל Tile כעבודה j. הריצו שלושה עומסים שונים על המערכת: small.txt, mid.txt, big.txt וציירו לכל אחד שני גרפים:

- i. גרף של Average Latency(A) כתלות במספר החוטים N.
- ii. גרף של Average Latency(j) כתלות במספר החוטים N.

- יש תחילה לכבות את דגל ההדפסה print\_on על מנת שהזמנים השונים לא יושפעו מתהליך ההדפסה. יש להעביר לארגומנט האחרון ל-main N במקום Y.
- יש להריץ כל קוניפגורציה ל-Generations 100, עם מספר חוטים משתנה:  $N = 1, 2, 3, \dots, 100$ , ולהשתמש ב-Avg Gen Time ו-Avg Tile Time המתקבל ב-results.csv. ניתן לכתוב סקריפט bash קצר לביצוע דבר זה.
- ניתן לטעון קבצי CSV (Comma Separated File) ישירות לאקסל באופן פשוט וקל ע"י פקודת Import.

ד. נתחו את הגרפים שהתקבלו באופן מעמיק. הניחו בנייתו שהחישוב מבוצע על מעבד יחיד. שאלות מכוונינות אליהם התייחסו בניתוח:

- האם בחלקים מסוימים ניתן לראות מגמה כזאת או אחרת? עליה/ירידה/קו שטוח? ממה הדבר נובע לדעתכם?
- מהו מספר החוטים האידיאלי לכל עומס? הסבירו סיבות לכך.
- האם בהכרח זמן החישוב היה משתפר אם היינו מוסיפים מעבדים נוספים?
- השוו בין הגרפים של העומס הקטן, גדול ובינוני. במידה ויש שוני, ממה נובע השוני בין הגרפים של העומסים השונים?
- האם הגרף מתנהג כמו אחד הגרפים שהתקבלו בסעיפים א', ב' באופן גס? אם כן, כמה מקבילי אתם מעריכים שהקוד שלכם?

**שימו לב:** הגרפים שיתקבלו בסעיף זה יכולים להיות שונים ומגוונים. לא בהכרח שהגרפים יסתדרו עם הציפיות שלכם. במידה ומתקבלים גרפים המתארים התנהגות לא "מקבילית" – בדקו את מימושכם עד שהשתכנעתם שהוא סביר. בכל מצב, הצדיקו את הגרפים שהתקבלו עם טיעונים איכותיים. הניקוד בחלק זה ינתן עבור הסברים משכנעים של התוצאות בגרף, המראים הבנה של החומר ושיקולי המערכת.