TECHNION
Israel Institute of Technology

Computer Science Department

Operating Systems – 234123

Homework Exercise 3 – Dry

Teaching Assistant in charge:

Ido Imanuel

Assignment Subjects & Relevant Course material

Threads, Synchronization, pthread Library

Recitations 5-6, Lectures 4-5

Submission Format

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 –

   DHW3_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. Convoluted 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,** put them in the **hw3** 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://goo.gl/forms/eW76r9cRNPTw9vAW2

הנחיות בנוגע לתרגיל הבית הנוכחי:

א. 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.

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 יינתן 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 שחוויית 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 **115**. 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 (או תכונות מנעולים, ביחס להבטחתם על 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 **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**.

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.

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 קימת 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. ממשים מנעול חדש הכולל:
   - Mutex סטנדרטי
   - מונה count
   - סף איטרציות MAX_ITER
   - קבוע שלם T

תיאור מנגנון הנעילה: בזמן ניסיון נעילה (דהיינו, קריאה לפונקציה ()lock), המנעול תומך ב-Timeout אותו ממשים על ידי מונה בצורה הבאה: במידה והחוטים 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 החוטים. תיאור בפסודו קוד של מימוש הפונקציה ()lock נתון בקטע הקוד הבא:

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

הניחו מערכת עם **מעבד יחיד** ואפשרות לנפילת חוטים 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 ? התשובה צריכה להיות מורכבת מערך מקסימלי אפשרי וערך מינימלי אפשרי, עם תרחיש אפשרי לכל אחד. ניתן להניח שפעולות store ו-load 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. בהנחה שהקוד מורץ על **<u>מעבד יחיד</u>**, 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.

```
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.
הניחו ש-sum הינו משתנה גלובלי.

```
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 בשמם 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. **<u>סעיף בונוס</u>** (**5 נקודות**): ממש משתנה תנאי בעזרת singlephore ו-mutex (שכבר מימשתם). מלא את תבניות הקוד הבאות: (שימו לב, הסעיף הבא **<u>אינו</u>** 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. 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: ניתוח של החלק הרטוב וחוק אמדל

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.

שאלה 1: ניתוח החלק הרטוב

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

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

השימוש במנגנון היה כדלקמן:

Producer:

1. Init Barricade b
2. Init PCQueue p
3. Init fields $curr, next$
4. for t=0 $\rightarrow$ t=n_generations

    $for\ i = 0 \rightarrow i = N$

        p.push(job);

        b.increase();

    b.wait();

    swap($curr, next$);

Consumer (One of N)

while(1)

    $job\ \ j = \text{p.pop()} \,//\, blocked\ here\ \ if\ \ queue\ is\ empty$

    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.

א. הסבירו את כוונותיו של פיראס – איך היה אמור המנגנון לעבוד? מה ההבדל העיקרי בין מנגנון זה לבין ה-semaphore עליו למדתם בביתה?

ב. במימוש זה מספר בעיות **Correctness**.

1. מצאו בעיה אחת של **Race Condition** בפתרון. הסבירו. בכלליות – RC מתרחש כאשר 2 או יותר חוטים בעלי משאב משותף רצים במקביל, ושקיים סדר תזמונים (scheduling) ביניהם **המשבש את לוגיקת הקוד** ביחס למשאב המשותף.
2. תארו תרחיש שבו מופר ה-**Mutual Exclusion**. דהיינו, חישוב הלוח curr טרם הסתיים, וה-Producer מבצע למרות זאת את ה-swap של הלוחות.
3. תארו שני תרחישים שונים בהם יתכן **Deadlock** בפתרון.

ג. תקנו את `class Barricade` ואת הפסודו קוד של היצרן-צרכן כך שכל בעיות ה-Correctness יפתרו. אין לשנות את מתווה הפתרון של פיראס באופן מהותי.

1. הראו פתרון אחד עם הוספה של mutex יחיד, וקריאות מתאימות שלו בתוך ה-Class והפסודו קוד.
2. מנגנון הסנכרון שפיראס ניסה ליצור, ואותו אתם השלמתם בסעיף 1' נקרא "מונה משותף", ואינו מוצלח במיוחד ממבט של בביצועים - Performance. הסבירו מדוע. התייחסו לחסרון המנגנון כאשר N, מספר החוטים האפקטיבי גבוה מאוד.
3. הראו פתרון נוסף בעזרת שימוש בפעולות אטומיות. הפעולות האטומיות שעומדות לרשותכם:
   - `atomicAdd(int * ptr,int val)` אשר מוסיפה **<u>באופן אטומי</u>** val לערך התא ptr
   - `atomicCAS`, עם תיאור הפעולה בקוד:

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

   - הניחו שפקודות אלו ממומשת **<u>בחומרה</u>**, ולא בתוכנה, כיאה לפעולות אטומיות.
   - **<u>אין</u>** להשתמש במנגנוני סנכרון נוספים, למשל mutex או semaphore, אך ניתן לנצל busy wait.
4. הסבירו למה הפתרון ב-(3) עדיף על פתרון ב-(1).

ד. **בונוס (10 נקודות):** הציגו מנגנון סנכרון המנצל **<u>שני</u>** PCQueues כקופסא שחורה לטובת הסנכרון ההדדי בין Producer ל-Consumer. הסבירו בפירוט את פתרונכם.

- **<u>אין</u>** להשתמש בכל מנגנון סנכרון נוסף (בפרט mutex,semaphore, atomics, ו-busy wait).
- מומלץ להחליף קטעים רלוונטיים בפסודו קוד לשם הצגת פתרונכם.
- פתרון זה אינו "מינימלי" – דהיינו, הוא מבצע פעולות מיותרות בצד ה-Producer אשר אינן דרושות לנכונות המנגנון. בסביבה בה ביצועי המנגנון הינו דבר קריטי, דבר זה מהווה בזבוז מהותי. הציגו מנגנון שקול **ומינימלי**, אשר אינו מבצע פעולות אלו. ניתן להשתמש ב-mutex יחיד, condvar יחיד ו-PCQueue יחיד.

שאלה 2: ביצועים וחוק אמדל

נניח אלגוריתם A המורכב ממספר רב של עבודות $j_1, j_2, \ldots, j_M$

לנוחיותכם, מושגים נפוצים לניתוח Performance של האלגוריתם:

i. **Latency(A)** – זמן החישוב הכולל של האלגוריתם A
ii. **Latency(j)** – זמן החישוב הכולל של עבודה j
iii. **Turnaround Time(j)** – זמן החישוב + זמן ההמתנה בתור של עבודה j
iv. **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. ניתן להשתמש בכל תוכנה ליצירת גרפים שתרצו, למשל Desmos, Excel, Python או Matlab.

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.

- סעיף זה נועד לספק לכם מדד איכותי לגרפים התיאורטיים שיתקבלו בתנאי "מעבדה", בהתאם לכמה האלגוריתם סריאלי/מקבילי.
- ניתן לקבע את זמן הביצוע הכולל אם A היה סריאלי לחלוטין כרצונכם (למשל – שניה).

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.
iii. גרף scatter plot של Tile Index אל מול Thread Id, כאשר Index Tile הינו מספור של כל ה-Tiles השונים לפי סדר ההופעה שלהם להיסטוגרמה `m_tile_hist`, ו-Thread Id הוא המספר הסידורי של החוט שביצע את העבודה על ה-Tile. שימו לב – עליכם לאפשר הדפסה של וקטור זה ע"י שינוי `main.cpp`.

- 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 קונפיגורציה ל-Generations 100, עם מספר חוטים משתנה: $N = 1, 2, 3, \ldots, \min(100, N_{effective})$, 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:**

שאלות מכוונות אליהם התייחסו בניתוח:

1. האם בחלקים מסוימים ניתן לראות מגמה כזאת או אחרת? עליה/ירידה/קו שטוח ? ממה הדבר נובע לדעתכם?
2. מהו מספר החוטים האידיאלי לכל עומס ? הסבירו סיבות לכך.
3. האם בהכרח זמן החישוב היה משתפר אם היינו מוסיפים **מעבדים** נוספים?
4. כיצד ניתן להסיק מהגרף השלישי **(tid vs tile id)** את מספר הליבות? מה גרף זה מספר לנו על יעילות הקוד שכתבתם? נסו לספק שערוך מספרי למספר הליבות הקיימות בסביבת ההרצה (למשל, שרתי CSL3 או LUX. אם הרצתם על מחשבכם האישי – מספר הליבות הניתן להסקה **מהגרף**). נסו לספק שערוך לגודל ה-time slice של כל חוט מתצורת הגרף.
5. השוו בין הגרפים של העומס הקטן, גדול ובינוני. במידה ויש שוני, ממה נובע השוני בין הגרפים של העומסים השונים?
6. 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.

במידה והייתם רוצים לספק גרפים על מטריצות אחרות או הגדרות אחרות – ניתן ורצוי, במידה והם מסייעים בהסבר שלכם לשאלה מעלה.