**תרגיל בית 3 – יבש – מערכות הפעלה**

יניב הולדר

yaniv.holder@campus.technion.ac.il

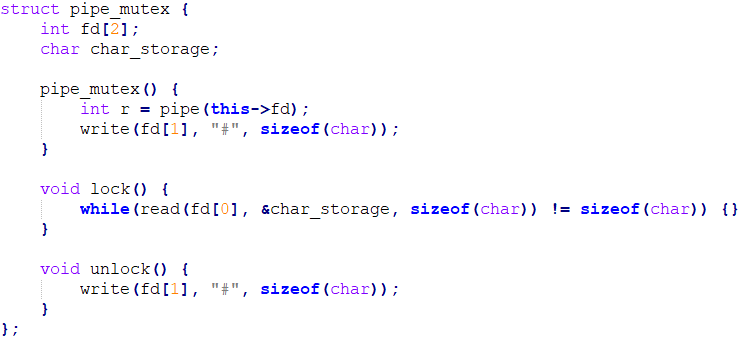
207025297

אילון קורנבוים

eilon.ko@campus.technion.ac.il

315677880

**חלק ראשון**



1. מדוע הגישה הזאת פועלת רק בין חוטים של אותו תהליך? ואיך ה-pipe שראינו בתרגול מתגברות על בעיה זאת?

יש במימוש זה הנחה שכל החוטים יכולים לגשת לשדות של אותו הmutex\_pipe . זאת בגלל שיש שימוש בחוצץ משותף על הheap, ולחוטים שונים של אותו תהליך יש heap משותף. לעומת זאת תהליכים שונים אינם חולקים heap ולכן הפתרון בתרגיל לא יתאים (שינוי ב mutex\_pipeשל תהליך אחד לא ישנה את ה mutex\_pipe של תהליך אחר)

בגישה בראינו בתרגול, כדי לשתף מידע בין תהליכם שונים, השתמשנו בFDT שלהם, והפננו ערוצים אל קובץ שיהיה משותף. כל הקבצים הפתוחים של כל התהליכים במערכת נשמרים בטבלה גלובאלית המנוהלת ע"י הגרעין – ה-Global FDT או GFDT. וביצענו הפניה של ערוצים בFDT של התהליכים אל הקובץ.   
החוצץ כפי שנראה בתרגול - אליו נכתב בפועל הpipe, נמצא בהקצאה של הגרעין, ומיוצג ע"י הקובץ, ולכן תהליכים שונים שלא חולקים heap יכולים לגשת אליו

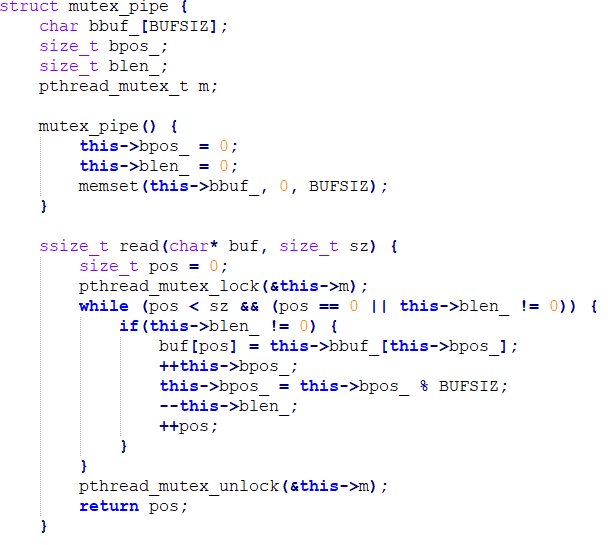
1. איזה מהשינויים הבאים (אם נעשה באופן נפרד) גורם להתנהגות לא מוגדרת בשימוש mutex\_pipe?
2. מחיקת שורה 4– **נקבל ערך זבל התחלתי עבור המרחק בין כותב וקורא ולכן לא ידוע מראש האם קריאה לפני כתיבה תוכל לקרות או לא (תלוי ב blen\_)**
3. מחיקת שורה 5 – **אכן שורה מיותרת ומחיקתה לא תשפיע על ההתנהגות**
4. מחיקת " this->blen\_ < BUFSIZ ||" משורה 18. – **בעקבות שינוי זה בפעולת הכתיבה רק התו הראשון ייכתב. התנהגות לא נכונה אך מוגדרת.**
5. מחיקת שורה 21 – **לפעמים תתרחש חריגה ממערך, לפעמים לא (תלוי אם בזמן הקריאה ל write מתקיים: bpos\_ + blen\_ + sz < BUFSIZ). אם תתרחש חריגה מהמערך, נקרא תוכן לא נכון בפעולת ה read.**
6. מחיקת שורה 23 ו-24 – **באג אבל התנהגות מוגדרת – לולאה אינסופית בפעולה ה write.**

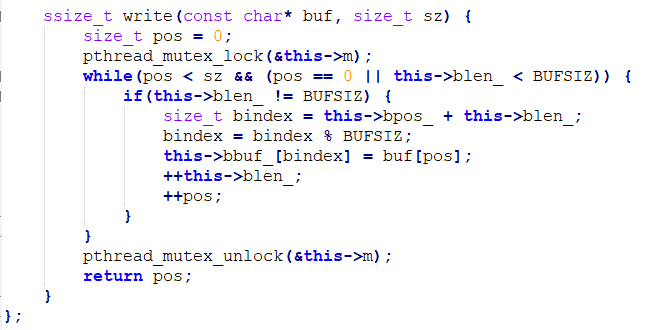


1. מחיקת שורה 4 – **למשל, קריאה ללא כתיבה לפניה, אם ב blen\_ היה ערך זבל 5 אז ייקראו 5 תווים של 0.**
2. מחיקת שורה 5 – **אכן שורה מיותרת ומחיקתה לא תשפיע על ההתנהגות**
3. מחיקת " this->blen\_ < BUFSIZ ||" משורה 18 – **אם בקריאה ל write העברנו "abc" ב buf בפועל הלולאה תיפסק לאחר איטרציה אחת וייכתב רק “a” ולכן לא נוכל לקרוא אחכ “abc” ואם תיקרא read היא תקרא רק “a”** .
4. מחיקת שורה 21 – **לפעמים תתרחש חריגה ממערך, לפעמים לא (תלוי אם בזמן הקריאה ל write מתקיים: bpos\_ + blen\_ + sz < BUFSIZ). אם תתרחש חריגה מהמערך, נקרא תוכן לא נכון בפעולת ה read.**
5. מחיקת שורה 23 ו-24 – **אם ננסה לכתוב "abc" פונקציית write תיכנס ללולאה אינסופית, לא נוכל לקרוא שום דבר (ובפרט לא משהו לא תקין) עם פונקציית write מכיוון ש blen\_ = 0 תמיד.**

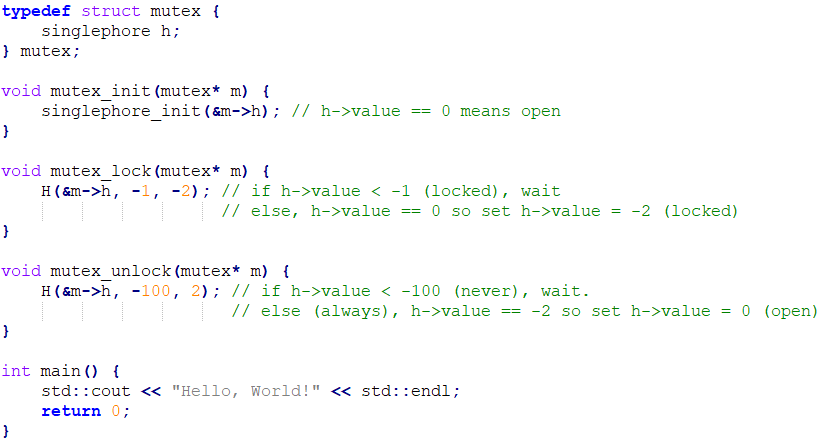


1. מחיקת שורה 4 – **למשל, קריאה ללא כתיבה לפניה, אם ב blen\_ היה ערך זבל ששווה ל BUFSIZ אז לולאת ה while תתחיל (pos == 0) אך ה if לא יקרה ולכן תתקבל לולאה אינסופית.**
2. מחיקת שורה 5 - **שורה מיותרת ומחיקתה לא תשפיע על ההתנהגות**
3. מחיקת " this->blen\_ < BUFSIZ ||" משורה 18. - **בעקבות שינוי זה בפעולת הכתיבה רק התו הראשון ייכתב ובפרט הפעולה תסתיים.**
4. מחיקת שורה 21 – **לפעמים תתרחש חריגה ממערך, לפעמים לא (תלוי אם בזמן הקריאה ל write מתקיים: bpos\_ + blen\_ + sz < BUFSIZ). אם תתרחש חריגה מהמערך, יכול להיות שתתרחש חרגית segfault ולכן write לא תסתיים. (כלומר לא יתרחש ה return בסופה)**
5. מחיקת שורה 23 ו-24 – **לולאה אינסופית, כי pos לא מקודם ועל כן תנאי הלולאה ישאר נכון לעד**
6. השלימו את המימוש שמצורף למעלה, וודאו שאין deadlock במימוש, ואופן ההמתנה יעיל





**חלק שני**



typedef struct condvar {

mutex m;

singlephore h;

vector<pid\_t> vector;

} condvar;

void cond\_init(condvar\* c) {

mutex\_init(m)

}

void cond\_signal(condvar\* c) {

lock (vector protection)

if vector.empty:

return;

to\_be\_countinued = vector.pop()

unlock (vector protection)

lock (the regular lock)

thr\_continue(to\_be\_countinued)??? Sigcont?

return

}

void cond\_wait(condvar\* c, mutex\* m) {

lock (vector protection)

vector.push(get\_thrread\_pid())

unlock (vector protection)

thr\_suspend(self/ get\_thrread\_pid())??? Sigstop?

unlock (the regular lock)

return

}

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

H(&c->h, INT\_MIN, 1); יוצא תנאי שתמיד מתקיים (אף ערך של h.value לא יהיה קטן מ INT MIN)

לכן, יבוצע קידום ב1 של ערך h. value גם במקרה שאף אחד לא ממתין. ומקרה זה לא עומד   
בתנאי רמז 1: אם אין חוט שמחכה על משתנה התנאי c ,אז (signal\_cond(c לא יעשה דבר.

אם ייקרא wait\_cond(c) לאחר מכן, אז הוא לא יגרום להשהיה עד ל signal\_cond הבא (מכיוון שבעת קריאתו h.value == 1) מה שמפר תנאי נוסף.

כמו כן, לאחר סיום wait, המנעול הושאר נעול, בניגוד לגישה מהתרגול " הפונקציה cond\_wait() מקבלת שני פרמטרים – מנעול ומשתנה תנאי – ואז משחררת את המנעול ויוצאת להמתנה באופן אטומי"

typedef struct singlephore\_waiter {

-thread pid

-the condition (maybe a pointer to a boolian func that checks the condition)

} singlephore\_waiter;

typedef struct singlephore {

mutex m;

int value;

std::vector<singlephore\_waiter\*> waiters;

// YOUR CODE HERE (if necessary)

} singlephore;

void singlephore\_init(singlephore \*h) {

h->value = 0;

// YOUR CODE HERE (if necessary)

}

void H(singlephore \*h, int bound, int delta) {

singlephore\_waiter hw;

mutex\_lock(&h->m);

while (h->value < bound) {

cond\_wait(&h->cv, &h->m);

}

h->value += delta;

if (delta > 0) {

for itr in vector:

if itr.cond == true:

thread\_signal

}

mutex\_unlock(&h->m);}

**חלק שלישי**

1. ניתן להניח כי פיראס התכוון שהשימוש במחלקה Barrier תשמש ככלי סנכרון בלי החוטים השונים בקוד שלו. בכל generation היצרן מעלה ב1 את המונה שב Barrier ולבסוף מבצע את פעולת wait אשר מבצעת לולאה אינסופית עד שהמונה ב Barrier יגיע ל 0. כאשר אחד החוטים מסיים לבצע משימה הוא מוריד את המונה ב Barrier ב1. כך, כנראה התכוון פיראס לגרום לחוט הראשי (היצרן) לחכות שכל שאר החוטים (הצרכים) יסיימו לבצע את כל המשימות שב PCQ.
2. a.