# מטלת מנחה (ממ"ן) 11

**הקורס: "מערכות הפעלה"**

**חומר הלימוד למטלה:** ראו פירוט בסעיף "רקע"

**מספר השאלות: 6** **משקל המטלה: 12**
**סמסטר: 2019ב** **מועד אחרון להגשה:** 28.3.2019

**הגשת המטלה: שליחה באמצעות מערכת המטלות המקוונת באתר הבית של הקורס.**
**הסבר מפורט ב"נוהל הגשת מטלות המנחה".**

**החלק המעשי (70%)**

**כללי**

בתרגיל זה עליכם לממש ספריית תהליכונים פשוטה ברמת המשתמש אשר מבצעת החלפת תוכן בין תהליכונים.

**מטרה**

- הכרת xv6 ושפת שף
- הכרת ההיבטים המעשיים של מימוש תהליכונים ברמת המשתמש
- שימוש ב-non-local branching

**רקע**

א) פרקים 2.3.5, 2.5.1, 2.2.1, 2.2.2, 2.2.3 בספר של Tanenbaum, Modern operating systems.

ב) פרק"Mafkefile" מחוברת " Ubuntu 16.04 programming environment, making first steps"

ג) קובץ "Running and debugging xv6.pdf"

ד) פרקים 0,1,2 מתוך https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf - xv6 book

## תיאור המשימה

1. בקובץ maman11.zip תמצאו ספרייה עם מערכת ההפעלה xv6 המכילה את הקבצים uthread.c ו uthread_switch.S.
2. עיינו ב Makefile על מנת ולוודא כי משמעות השורות הבאות ברורה לכם:

```
_uthread: uthread.o uthread_switch.o
	$(LD) $(LDFLAGS) -N -e main -Ttext 0 -o _uthread uthread.o uthread_switch.o $(ULIB)
	$(OBJDUMP) -S _uthread > uthread.asm
```

שימוב לב שלפי כללי הכתיבת ה Makefile לפני שורת הפקודה מופיע TAB (ולא רווחים).

3. חפשו ב Makefile במקום שבו רשומות כל תוכנות ה userspace של xv6 את השורה המכילה את _uthread וודאו שאתם מבינים את משמעות המשתנה UPROGS.
4. קראו כיצד מריצים את המערכת vx6 מתוך הקובץ "Running and debugging xv6.pdf" המופיע בחומר רקע של הממ"ן. הריצו את xv6 באמצעות אחת הפקודה הבאה:

```
make CPUS=1 qemu
```

שימו לב, שורת הפקודה איתה הרצנו את xv6 מכילה CPUS=1. עליית המערכת תיראה כך:

```
$ make CPUS=1 qemu
dd if=/dev/zero of=xv6.img count=10000
10000+0 records in
10000+0 records out
5120000 bytes transferred in 0.037167 secs (137756344 bytes/sec)
dd if=bootblock of=xv6.img conv=notrunc
1+0 records in
1+0 records out
512 bytes transferred in 0.000026 secs (19701685 bytes/sec)
dd if=kernel of=xv6.img seek=1 conv=notrunc
307+1 records in
307+1 records out
157319 bytes transferred in 0.003590 secs (43820143 bytes/sec)
qemu -nographic -hdb fs.img xv6.img -smp 1 -m 512
Could not open option rom 'sgabios.bin': No such file or directory
xv6...
cpu0: starting
init: starting sh
$
```

5. כשמערכת xv6 תעלה, הריצו את הפקודה uthread מתוך שורת הפקודה של המערכת. הרצת התוכנית uthread תגרום לשגיאה:

```
$ uthread
pid 4 uthread: trap 14 err 5 on cpu 1 eip 0xffffffff addr 0xffffffff--kill proc
```

6. המשימה שלכם היא להשלים את uthread_switch.S כך שהפלט של ה uthread שלכם תהיה זהה (עד כדי הכתובות) לפלט הבא:

```
$ uthread
my thread running
my thread 0x2A30
my thread running
my thread 0x4A40
my thread 0x2A30
```

```
my thread 0x4A40
my thread 0x2A30
my thread 0x4A40
....
```

**הסבר מפורט**

1. כפי שניתן לראות uthread מייצרת 2 תהליכונים ומחליפה בינהם בצורת round-robin. כל תהליכון מדפיס "my thread ..." ולאחר מכן מוותר על ה CPU לטובת תהליכון אחר.
2. לפני שתגשו למימוש של uthread_switch.S, הבינו כיצד unthread.c משתמשת ב uthread_switch.S. שימו לב ש uthread.c מגדירה 2 משתנים גלובליים: current_thread ו next_thread. כל אחד מהם הוא בעצם מצביע ל thread structure. מבנה של thread (או thread structrure) מכיל מחסנית (stack) ומצביע של המיקום בתוך המחסנית (מצביע ה sp). תפקידו של uthread_switch.S הוא לשמור את מצב התהליכון הנוכחי בתוך מבנה של thread אשר אליו מצביע המצביע current_thread, לאחר מכן לשחזר את התוכן של next_thread ולבסוף לגרום ך current_thread להצביע למבנה אליו הצביע ה next_thread.
3. עליכם להבין את thread_create אשר מבצעת אתחול מחסנית לתהליכון חדש. הבנת thread_create תספק לכם רמזים על מה ש thread_switch אמורה לבצע. הכוונה היא ש thread_switch תשתמש בפקודות שפת שף popal ו pushal כל מנת לשחזר ולשמור את שמונת האוגרים של x86. שימו לב, thread_create מסמלצת מצב (עושה סימולציה של מצב) שבו שמונת האוגרים נשמרו במחסנית.
4. על מנת לכתוב את thread_switch, עליכם להבין כיצד מהדר (compiler) מאחסן את struct thread בזיכון:

```
    --------------------
    | 4 bytes for state|
    --------------------
    | stack size bytes |
    | for stack        |
    --------------------
    | 4 bytes for sp   |
    --------------------  <--- current_thread
         ......

         ......
    --------------------
    | 4 bytes for state|
    --------------------
    | stack size bytes |
    | for stack        |
    --------------------
    | 4 bytes for sp   |
    --------------------  <--- next_thread
```

5. על מנת לכתוב לשדה sp של struct thread אשר אליו מצביע current_thread, תעזרו בקוד הבא:

```
movl current_thread, %eax
movl %esp, (%eax)
```

הוא שומר את esp% ב current_thread->sp וזה עובד כי sp "יושב" בהסטר 0 בתוך ה struct thread.

6. אתם יכולים ללמוד את שפת הסף אשר נוצרה מ unthread.c ע"י הסתכלות על הקובץ uthread.asm.

7. לאחר שהשלמתם את 10 שורות הקוד החסרות ב utrhead_switch.S, בדקו את הקוד שלכם. בנוסף להרצה של תוכנית ה utrhead, תוכלו לבצע מעבר step-by-step על הוראות של thread_switch באמצעות ה gdb. קראו כיצד מפעילים את ה gdb כדי "לדבג" תוכניות משתמש במערכת vx6 מתוך הקובץ "Running and debugging xv6.pdf" המופיע בחומר רקע של הממ"ן. הרוצו ב gdb את הפקודות הבאות:

```
(gdb) target remote localhost:26000
Remote debugging using localhost:26000
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x0000fff0 in ?? ()
(gdb) symbol-file _uthread
Reading symbols from _uthread...done.
(gdb) b thread_switch
Breakpoint 1 at 0x21f: file uthread_switch.S, line 9.
(gdb)
```

שימו לב, ש breakpoint יכול להיות מופעל אף לפני שהרצתם את thread_switch. וודאו שאתם מבינים כיצד הדבר יכול לקראות (מתוך ההסברים המופיעים ב "Running and debugging xv6.pdf".

8. כש xv6 עולה, הריצו משורת הפקודה את התוכנית uthread. מנפה שגיאות gdb יגיעה ל breakpoint ב thread_switch ותוכלו לבצע פקודות לבחינת מצב ה uthread. לגודמא:

```
(gdb) p /x next_thread->sp
$4 = 0x4ae8
(gdb) x /9x next_thread->sp
0x4ae8 <all_thread+24560>:    0x00000000    0x00000000    0x00000000    0x00000000
0x4af8  <all_thread+24576>:   0x00000000    0x00000000    0x00000000    0x00000000
0x4b08 <all_thread+24592>:    0x000000d8
(gdb) p next_thread->state
$5 = 1
(gdb) p current_thread->state
$6 = 2
```

**הגשה**

יש להגיש את הקובץ uthread_switch.S בלבד. אין להגיש קבצים מקומפלים. ראה הוראות הגשה כלליות בחוברת הקורס.

את הקובץ/הקבצים המוגש/ים יש לשים בקובץ ארכיון בשם exYZ.zip (כאשר YZ הנו מספר המטלה). הכנת קובץ ארכיון מתבצעת ע"י הרצת הפקודה הבאה משורת הפקודה של Ubuntu:

<zip exYZ.zip <ExYZ files

*<u>הערה חשובה: בכל קובץ קוד שאתם מגישים יש לכלול כותרת הכוללת תיאור הקובץ, שם הסטודנט ומספר ת.ז.</u>*

**בדיקה לאחר ההגשה**

לאחר ההגשה יש להוריד את המטלה (חלק מעשי/עיוני) משרת האו"פ למחשב האישי ולבדוק שהקבצים אכן הוגשו באופן תקין ושניתן לקרוא אותם. בנוסף, הבדיקה של החלק המעשי תכלול את הצעדים הבאים:

- פתיחת ארכיון *exXY.zip* בספרייה חדשה (*new folder*).
- יצירת ספריה חדשה עם הקוד המקורי של *xv6*
- העתקת הקובץ המוגש לספרייה עם הקוד המקורי של *xv6*
- הרצת *make qemu* ווידוא שכל ה *target* נוצר ללא שגיאות וללא *warnings*
- הרצת בדיקות רלונטיות לוידוא תקינות הריצה של החלק המעשי

***החלק העיוני (30%)***

שאלה 2 (10%)

א) מהי פעולת ה TRAP? תארו מתי היא מתבצעת ומה קורא בעת ביצועה.

ב) הסבירו מה קורה בעת הקריאה לפונקציית write של ה C library. בפרט הסבירו כיצד עוברים הפרמטרים של ה write למערכת הפעלה Linux וכיצד המערכת מטפלת ב write. יש התייחס הן למקרה של legacy system calls והן ל fast system calls.

ג) מה ההבדל בין write ל printf? תוכלו להעזר בקבצי מקור של C library מ www.gnu.org/software/libc

שאלה 3 (5%)

הסבר את מדוע פתרון התור (strict alternation), איננו מהווה פתרון סביר. איזה תנאי/ים הוא מפר.

שאלה 4 (10%)

תקראו פרק 3 של המאמר שדן בנושא הוספת תהליכונים כספריה לשפה שלא תמכה בהם מלכתחילה והסברו מדוע תקן של Pthreads אינו מתאר באופן פורמאלי את מודל הזיכרון ואת הסמנטיקה של המקביליות הממומשות ב Pthreads. כיצד מפתחי התקן מסבירים מהו מודל הזיכרון בכל זאת?

שאלה 6 (5%)

הוכיחו כי בפתרון של Peterson תהליכים אינם ממתינים זמן אינסופי על מנת להיכנס לקטע קריטי. בפרט הוכיחו כי תהליך שרוצה להיכנס לקטע קריטי לא ממתין יותר ממה שלוקח מתהליך אחר להיכנס ולעזוב את הקטע הקריטי.

**הגשת החלק העיוני**

החלק העיוני יוגש כקובץ Word או כקובץ pdf. שם הקובץ צריך להיות exYZ.pdf או exYZ.doc (כאשר YZ הנו מספר המטלה).