ה א ו נ י ב ר ס י ט ה ה פ ת ו ח ה

20594

מערכות הפעלה

חוברת הקורס – סתיו 2020א

כתב: דוד שריאל

נובמבר 2019 – סמסטר סתיו- תש״פ

פנימי – לא להפצה.

תוכן העניינים

אל הסטודנט,

אנו מקדמים את פניך בברכה עם הצטרפותך אל הלומדים בקורס " מערכות הפעלה".

בחוברת זו תמצא את לוח הזמנים, תנאים לקבלת נקודות זכות ומטלות.

לקורס קיים אתר באינטרנט בו תמצאו חומרי למידה נוספים, אותם מפרסם/מת מרכז/ת ההוראה. בנוסף, האתר מהווה עבורכם ערוץ תקשורת עם צוות ההוראה ועם סטודנטים אחרים בקורס. פרטים על למידה מתוקשבת ואתר הקורס, תמצאו באתר שה"ם בכתובת:
http://telem.openu.ac.il
מידע על שירותי ספרייה ומקורות מידע שהאוניברסיטה מעמידה לרשותכם, תמצאו באתר הספריה באינטרנט www.openu.ac.il/Library.

אפשר לפנות אלי בדואר אלקטרוני davidsa@openu.ac.il או בשעות הנחיה הטלפונית המפורסמות באתר הקורס. הפרטים הללו מצויים גם באתר המחלקה למדעי המחשב telem.openu.ac.il/cs. פגישות יש לתאם מראש.

חשוב להדגיש כי התקשוב בקורס ישמש ערוץ רשמי בין צוות ההוראה של הקורס לבין הסטודנט, כלומר חובה על כל סטודנט להתעדכן באופן שוטף על הנעשה בקורס דרך אתר הבית. כל ההודעות - הן בנושאים אקדמיים והן בנושאים מנהליים - יועברו דרך אתר הבית בלבד, ולא יישלחו הודעות בדואר רגיל. סטודנטים אשר אין להם גישה לרשת האינטרנט יוכלו לגשת למרכז הלימוד הקרוב לביתם ולהשתמש במעבדת המחשבים שם. לפרטים מלאים על מרכזי הלימוד ושעות הפתיחה, ניתן להתקשר למוקד הפניות בטלפון: 09-7782222.

לתשומת לב הסטודנטים הלומדים בחו"ל:

למרות הריחוק הפיסי הגדול, נשתדל לשמור אתכם על קשרים הדוקים ולעמוד לרשותכם ככל האפשר.

הפרטים החיוניים על הקורס נכללים בחוברת הקורס וכן באתר הקורס.

מומלץ מאד להשתמש באתר הקורס ובכל אמצעי העזר שבו וכמובן לפנות אלינו במידת הצורך.

אל אתר הבית של הקורס ניתן לגשת מדף הבית של החטיבה למדעי המחשב:

http://telem.openu.ac.il/cs

בברכת לימוד פורה ומהנה,

דוד שריאל

מרכז ההוראה בקורס

1. לוח זמנים ופעילויות (20594 /א2020)

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
1	8.11.2019-3.11.2019	ראו חלוקה שבועית באתר הקורס		
2	15.11.2019-10.11.2019	ראו חלוקה שבועית באתר הקורס		
3	22.11.2019-17.11.2019	ראו חלוקה שבועית באתר הקורס		
4	29.11.2019-24.11.2019	ראו חלוקה שבועית באתר הקורס		
5	6.12.2019-1.12.2019	ראו חלוקה שבועית באתר הקורס		ממ"ן 11 5.12.2019
6	13.12.2019-8.12.2019	ראו חלוקה שבועית באתר הקורס		
7	20.12.2019-15.12.2019	ראו חלוקה שבועית באתר הקורס		
8	27.12.2019-22.12.2019 (ב-ו חנוכה)	ראו חלוקה שבועית באתר הקורס		

** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".*

לוח זמנים ופעילויות - המשך

שבוע הלימוד	תאריכי שבוע הלימוד	יחידת הלימוד המומלצת	מפגשי ההנחיה*	תאריך אחרון למשלוח הממ"ן (למנחה)
9	3.1.2020-29.12.2019 (א-ב חנוכה)	ראו חלוקה שבועית באתר הקורס		ממ"ן 12 2.1.2020
10	10.1.2020-5.1.2020	ראו חלוקה שבועית באתר הקורס		
11	17.1.2020-12.1.2020	ראו חלוקה שבועית באתר הקורס		
12	24.1.2020-19.1.2020	ראו חלוקה שבועית באתר הקורס		
13	31.1.2020-26.1.2020	ראו חלוקה שבועית באתר הקורס		ממ"ן 13 30.1.2020
14	7.2.2020-2.2.2020	ראו חלוקה שבועית באתר הקורס		

מועדי בחינות הגמר יפורסמו בנפרד

** התאריכים המדויקים של המפגשים הקבוצתיים מופיעים ב"לוח מפגשים ומנחים".*

2. תיאור המטלות

קרא היטב עמודים אלו לפי שתתחיל לענות על השאלות

חוברת זו מכילה מידע על המטלות ואת המטלות עצמן.
פתרון המטלות הוא חלק בלתי נפרד מלימוד הקורס - הבנה מעמיקה של חומר הלימוד דורשת תרגול רב. המטלות יבדקו על-ידי המנחה ויוחזרו לך בצירוף הערות המתייחסות לתשובות.

לכל מטלה נקבע משקל. יש לצבור 36 נקודות. חובה להגיש את כל המטלות.

ללא צבירת 36 נקודות בהגשת מטלות לא ניתן יהיה לגשת לבחינת הגמר

לתשומת לבכם!
ציון סופי מחושב רק לסטודנטים שעברו את בחינת הגמר בציון 60 ומעלה והגישו את כל המטלות בציון 60 לפחות.

כל סטודנט יכין את הממ"נים לבדו. אין להגיש את הממ"נים בזוגות (או קבוצות) !

3. התנאים לקבלת נקודות זכות

א. הגשת מטלות במשקל כולל של **36** נקודות לפחות עם ציון מינימלי של **60** נקודות בכל אחת מהמטלות שהוגשו.
ב. ציון של לפחות **60** נקודות בבחינת הגמר.

4. הדרכה לפתרון תרגילי התכנות

תרגילי התכנות בקורס זה דורשים מאמץ ניכר. התרגילים לכשעצמם אינם קשים באופן מיוחד אולם הם דורשים הכרה והבנה טובה של החומר המוצע כחומר רקע (ראו סעיף "חומר קרע" בגוף כל ממ"ן)

למרות שהקוד הנדרש בסופו של דבר בתרגילי התכנות איננו ארוך, סביר להניח כי תקדישו לתרגילים שעות רבות. תכנות מערכת הפעלה, דורש ניסיון, ולמרבה העצב רכישת הניסיון כרוכה לרוב גם בהקדשת זמן. עם זאת, התרגילים תוכננו כך שיעסקו מעט ככל האפשר בנושאים שמטבעם הם טכניים בלבד.

בפתרון התרגילים אנו מציעים את השלבים הבאים:

א. קראו היטב את דרישות התרגיל והבהירו לעצמכם מה הבעיות שעלולות להתעורר בעת יישומו.

ב. קראו את החומר המוצע כחומר רקע (ראו סעיף "חומר קרע" בגוף כל ממ"ן). לצורך זה מצויים
בידכם ארבעה מקורות, עיינו בהם על פי הסדר הבא:

1. ספר הקורס , Modern Operating Systems, המספק את הרקע התיאורטי.
2. המדריך למתכנת המערכת, The GNU C library reference manual, מתאר את פעולת קריאות המערכת ברוב מערכות UNIX הקיימות
3. הפקודה "man *command-name*" ב-UNIX מאפשרת לקבל מידע על פקודות, פונקציות ספריה, וקריאות מערכת, כפי שהן ממומשות במערכת שבידך.
4. מידע נוסף שמכיל דוגמאות קוד והסברים אפשר למצוא באינטרנט, בפרט באתרים שכתובותיהם מצויים בקטגוריה "אתרים ברשת" (ראו את הדף הראשי של אתר הקורס).

ג. בעת כתיבת הקוד, הקפידו על הכללים המקובלים, בהנדסת תוכנה. רוב הדרישות המפורטות כאן מוכרות לכם בודאי מקורסים קודמים אומנם ישנן דרישות ייחודיות לקורס במערכות הפעלה. לקיום הדרישות הללו קיימת השפעה על ציון הממ"ן:

1. מתן שמות משמעותיים למשתנים.
2. הימנעות משימוש במספרים שרירותיים.
3. כתיבת פונקציות קצרות.

4. תיעוד סביר. הכוונה לתיעוד מתומצת של פעולות התוכנית, של פונקציות ושל משתנים. כמו כן, יש לרשום בתחילת כל קובץ קוד שמוגש את הפרטים האישיים (שם מלא ומספר סטודנט) ותיאור קצר של תוכן הקובץ.
5. יש להקפיד על שימוש בשמות המוגדרים במטלה.
6. אין להשתמש ב goto. ליציאה מלולאות ניתן להשתמש במידת הצורך ב continue או break.
7. מבנה מדורג. מודולים ופונקציות קצרות וללא אפקטים משניים.
8. Indentation.
9. משפטי תנאי קצרים.
10. כל יציאה בגלל שגיאה חייבת להיות מתועדת. למשל, באמצעות הפונקציה perror().
11. בכל מקרה יש לבדוק את הערך המוחזר על ידי קריאות מערכת.
12. בכל מקרה יש לבדוק את נכונות הקלט.
13. התוכנית לא תיפול עקב שגיאה/תקלה כלשהי. במידה וקורה אירוע בלתי צפוי, על התוכנית להודיע על כך ולסיים את עבודתה.
14. אין להשתמש בפונקציה system().
15. יש לשחרר את כל המשאבים שאינם בשימוש.
16. הוראות קומפילציה יש לכתוב בשפת ההוראות של תוכנית השירות make ולהגישם בקובץ בשם makefile.
17. חובה להשתמש בדגל (flag) "Wall-" בזמן קומפילצית התוכניות

בונוס

במקרים יוצאי דופן, כאשר מוגשת תוכנית טובה במיוחד או כזו שעושה למעלה ממה שנדרש, תישקל האפשרות להוסיף עד 5 נקודות בונוס. בכל מקרה שהנכם מתכוונים להגיש תוכנית מעין זו, שימו לב כי:

1. כל הדרישות מהתוכנית המקורית יתקיימו.
2. כל תוספת תהיה מתועדת היטב.
3. תוספות המכילות שגיאות עלולות להוריד מהניקוד הסופי גם אם התוספות לא נדרשו במטלה. כוונות טובות אינן מובילות בהכרח לתוצאה הרצויה.

n

מטלת מנחה (ממ"ן) 11

הקורס: "**מערכות הפעלה**"

חומר הלימוד למטלה: ראו פירוט בסעיף "רקע"

מספר השאלות: 6 **משקל המטלה: 12**

סמסטר: 2020 א **מועד אחרון להגשה: 5.12.2019**

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 xv6 - book https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf

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 בMakefile במקום שבו רשומות כל תוכנות ה userspace של xv6 את השורה המכילה את _uthread ווודאו 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:

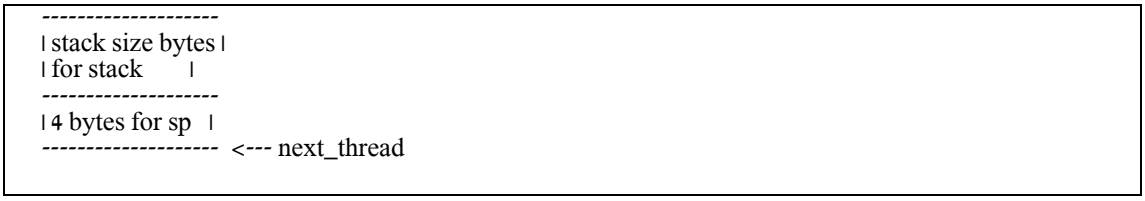
```
$ uthread
my thread running
my thread 0x2A30
my thread running
my thread 0x4A40
my thread 0x2A30
my thread 0x4A40
my thread 0x2A30
my thread 0x4A40
....
```

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 ל 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|
```



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%)

הראו כיצד ניתן לממש סמפרורים באמצעות העברת הודעות. יש לכתוב פסאודו-קוד המממש פעולות up ו down באמצעות send ו receive.

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 Wordאו כקובץ pdf. שם הקובץ צריך להיות exYZ.pdfאו exYZ.doc (כאשר YZהנו מספר המטלה).

מטלת מנחה (ממ"ן) 12

הקורס: "מערכות הפעלה"

חומר הלימוד למטלה: ראו פירוט בסעיף "רקע"

מספר השאלות: 5 **משקל המטלה: 12**

סמסטר: 2020 א **מועד אחרון להגשה: 2.1.2020**

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

החלק המעשי (80%)

בחלק זה של המטלה נממש מנגנון של job control ב command interpreter עם פונקציונליות מוגבלת.

מטרת התרגיל: תהליכים, תקשורת בין התהליכים, job control.

רקע

1. סיפקנו את הקובץ shell.c אותו אתם אמורים לשנות ולהרחיב ובפרט להוסיף את המנגנון של job control. קמפלו והריצו את התוכנית. עיינו בקובץ shell.pdf להסברים. כל השינוי מסתכם במספר מועט של שורות קוד אך כדי לבצע אותו עליכם להבין מספר נושאים להלן.
2. הסבר http://www.gnu.org/software/libc/manual/html_node/Executing-a-File.html עם פונקציות ממשפחת exec (אפשר להשתמש בפונקציה לבחירתכם). כמו כן סיפקנו את הקובץ exec.c שאפשר לקמפל ולהריץ.
3. פרקים 24.7.2, 24.7.3 מ http://www.gnu.org/software/libc/manual/html_mono/libc.html עם הסבר על פונקציות signal, sigemptyset, sigfillset, sigaddset, sigprocmask, sigsuspend. סיפקנו קובץ suspend.c. תקמפלו תריצו והבינו.
4. פרק http://www.gnu.org/software/libc/manual/html_mono/libc.html#Pipes-and-FIFOs עם הסבר על פונקציית pipe לתקשורת בין התהליכים. ספקנו קובץ pipe.c. תקמפלו תריצו והבינו.

5. פרק 13.12 מ https://www.gnu.org/software/libc/manual/html_mono/libc.html#Duplicating-Descriptors עם הסבר על פונקצית dup. תקמפלו ותריצו את dup.c שספקנו כדי לראות שימוש ב dup. dup.c מהווה וריאציה של pipe.c מהסעיף הקודם ומדגימה כיצד ניתן ליצור ערוץ תקשורת בין שני תהליכים בצורה שהיא שקופה לתהליכים עצמם.
6. פרק 14.1 מ http://www.gnu.org/software/libc/manual/html_mono/libc.html עם הסבר על הפונקציה chdir.
7. פרקים 27.6.2, 27.7.3 ו 27.7.2 מ http://www.gnu.org/software/libc/manual/html_mono/libc.html עם הפונקציות getpgrp, setpgrp, tcgetpgrp.
8. כמו כן יש להיזכר בפונקציות wait, fork.

כמו ניתן לקבל מידע על הפונקציות הנ"ל מה manשל LINUX.

תיאור המשימה

סיפקנו את הקובץ shell.c אותו אתם אמורים לשנות ולהרחיב. בפרט עליכם לממש מנגנון של job control ב command interpreter בשם smash (small shell). אחרי ההרחבה smash יהיה מסוגל:

1) לאפשר שרשור של לפחות 2 פקודות.
2) לתמוך בפקודות פנימיות exit ו cd.
3) להריץ תוכניות ברקע ובזמן אמת (background ו foreground).
4) לתמוך בפקודות bg, fg, jobs ולהגיב לסיגנלים של job control.

קיבלתם קובץ shell.c המממש פונקציונליות 1, 2, 3. כתבו עבורו Makefile שמייצר קובץ הרצה smash והריצו אותו משורת הפקודה:

```
maman12$ ./smash
```

כמו כן קיבלתם את קובץ smash_SSol המממש גם את 4. הריצו אותו משורת הפקודה:

```
maman12$ ./smash_SSol
```

במטלה הזאת עליכם לכתוב כ 40שורות קוד למימוש בפקודות של job control (סעיף ד). שאר הפונקציונליות כבר ממומשת (סעיפים א,ב,ג). אבל למימוש ה job controlעליכם להבין כיצד פועלים שאר הדברים. המיקום של השורות אותן תצטרכו לממש מופיע בקובץ shell.pdfיחד עם ה והפסאודו-קוד. מי שמכיר מהו שרשור הפקודות ב shellואת הפקודות jobs, fg, bgיכול לעבור ישירות לקריאת ההסברים הקובץ shell.pdf. אחרת, מומלץ קודם לקרוא הסברים מטה.

<u>הרצת תוכניות ברקע ובזמן אמת</u>

הרצת תוכנית (תוכניות) בזמן אמת (foreground) גורמת ל command interpreter להמתין עד סיום התוכנית (תוכניות). למשל

```
# ls
# ps | wc –l
```

הן דוגמאות להרצת תוכניות בזמן אמת.

הרצת תוכנית(תוכניות) ברקע (background) לא גורמת ל command interpreter להמתין עד לסיום התוכנית (התוכניות) שרצות ברקע. הרצת תוכניות ברקע תתבצע ע"י הוספת "&" בסוף שורת הפקודה. למשל:

```
# find /home -name Makefile –print &
# chown –R root: root /tmp&
```

שרשורש לפקודות

smash מאפשר שרשור של לפחות שני פקודות בשורת פקודה אחת. השרשור מתבצע ע"י סימן "|" (pipeline) בין הפקודות. משמעות השרשור היא שפלט של הפקודה הראשונה מהווה קלט לפקודה השנייה. כך למשל הרצת

```
# cat /etc/passwd | wc –l
```

גורמת לספירת כמות השורות בקובץ /etc/passwd. הפקודה "cat /etc/passwd" מדפיסה את תוכן הקובץ /etc/passwd ל stdout. באמצעות ה "|" אפשר "לומר" ל smash להפנות את הפלט של cat לתוכנית wc . והתוצאה שהיא כמות השורות בקובץ תודפס על הצג.

תמיכה בפקודותשל job control

smash יתמוך בפקודות הבאות:

1) jobs – הפקודה תגרום להדפסה של כל התהליכים **<u>המושהים</u>** ושל כל התהליכים **<u>שרצים ברקע</u>** אשר הורצו בעבר מתוך smash. תהליך מושהה הוא תהליך שהיה רץ בזמן אמת ואשר הושהה (למשל באמצעות Ctrl-Z). אם תהליך כלשהו רץ בזמן אמת, הצירוף Ctrl-Z משהה את ריצתו ומחזיר את שורת ה prompt של smash. מכאן שאם רוצים להריץ פקודת jobs ייתכן ויהיה צורך להשהות קודם תהליך שרץ בזמן אמת. לדוגמא:

```
# find /home -name Makefile –print
<Ctrl-Z>
[1] Stopped          find /home -name Makefile –print
# jobs
[1] Stopped          find /home -name Makefile –print
#
```

פקודת jobs נותנת לתהליכים מספר סידורי פנימי (ששונה בד"כ מ pid של תהליך) לפיו ניתן לזהות באופן יחיד כל תהליך שעדיין לא הסתיים ואשר הורץ מתוך smash.

2) fg %N – הפקודה תגרום להרצת תהליך [N] בזמן אמת. כך בדוגמא הקודמת הרצת

```
# fg %1
```

תעביר את find לרוץ בזמן אמת ולא תחזיר את ה prompt של ה smash עד לסיום ה find או עד להשהיתו הבאה.

3) bg %N – הפקודה תעביר את התהליך [N] ממצב מושהה למצב רץ ברקע.

תמיכה בפקודות פנימיות

smash יתמוך בשתי פקודות פנימיות:

א) exit – בעקבות הקשת הפקודה יסיים smashאת פעולתו.

ב) cd - בעקבות הקריאה לפקודה זו ישנה smash את ספרית העבודה הנוכחית שלו.

טיפול בשגיאות

smash צריכה לתת הודעות שגיאה על כשלון של קריאות מערכת או פונקציות שמכילות קירות מערכת. במקרה של שגיאהפטאלית יש לצאת עם סטטוס 1 (ע"י exit(1))

הגשה

יש להגיש **<u>כל</u>** קבצי הקוד ו Makefile המייצר קובץ הרצה smash. אין להגיש קבצים מקומפלים. את הקבצים המוגשים יש לשים בקובץ ארכיון בשם exYZ.zip (כאשר YZ הנו מספר המטלה). הכנת קובץ ארכיון מתבצעת ע"י הרצת הפקודה הבאה משורת הפקודה של Linux:

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 האישיולבדוק שהקבצים אכן הוגשו באופן תקין ושניתן לקרוא אותם. בנוסף, הבדיקה של החלק המעשי תכלול את הצעדים הבאים:

- פתיחת ארכיון exXY.zipבספרייה חדשה (new folder).
- וידוא שכל הקבצים הדרושים נוצרו בספריה בה פתחתם את הארכיון.
- הרצת makeווידוא שכל ה targetsנוצרו ללא שגיאות וללא warnings
- הרצת בדיקות רלונטיותלוידוא תקינות הריצה של החלק המעשי

<u>פתרון ביה"ס</u>

קיבלתם את קובץ smash_SSol כפי שמומש על ידינו.

החלק עיוני (20%)

שאלה 1 – (5%)

מדוע לא ניתן להשתמש באלגוריתם LRU בצורתו הטהורה לפינוי דפים (page eviction)?

שאלה 2 – (5%)

האם דף יכול להיות בו זמנית בשתי קבוצות עבודה (working sets)? נמקו.

שאלה 3 – (5%)

תארו מצב שבו התמיכה בזיכרון וירטואלי מסתברת כרעיון לא מוצלח. מה אפשר לשפר במצב שתיארתם אם אין תמיכה בזיכרון הוירטואלי?

שאלה 4 – (5%)

בעזרת תוכניות העזר size ו file גלו מהו גודלן הממוצע והחציון של קבצי הרצה במערכת הפעלה שסיפקנו לכם (Ubuntu 16.04). הסתכלו רק על קבצי ההרצה (לא על קבצי scripts) בספריות:

- /bin
- /usr/bin

הדגימו את החישוב של הגודל האופטימאלי של דפים במערכת בהתבסס על גודל ה text segment של קבצי ההרצה שמצאתם? הניחו שגודל של entry בטבלת הדפים הוא 4 בתים. הניחו שלכל קבצי ההרצה הסתברות זהה לרוץ ושהם מקבלים יחס זהה מבחינת מערכת ההפעלה. קחו בחשבון את הריסוק הפנימי.

הגשת החלק העיוני

החלק העיוני יוגש כקובץ Wordאו כקובץ pdf. שם הקובץ צריך להיות exYZ.pdfאו exYZ.doc (כאשר YZהנו מספר המטלה).

מטלת מנחה (ממ"ן) 13

הקורס: "עקרונות מערכות הפעלה"

חומר הלימוד למטלה: ראו פירוט בסעיף "רקע"

משקל המטלה: 12 **מספר השאלות: 5**

מועד אחרון להגשה: 30.1.2020 **סמסטר: 2020 א**

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

החלק המעשי (80%)

כללי

במטלה עליכם לשנות את מערכת הקבצים של xv6 ולהוסיף לה תמיכה בקבצים גדולים.

מטרה

- הכרת מבנה מערכת קבצים של xv6
- כתיבת regression test למערכת הקבצים של xv6
- היכרות בסיסית עם עבודה בפקויקט open source

רקע

א) פרק 13.2 ב Glibc manual המתייחס לפונקציות lseek, open, close, read, write.
ב) פרק 6 מתוך https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf
ג) קובץ "Running and debugging xv6.pdf"

תיאור המשימה

1. עליכם להבין כיצד בנוייה מערכת הקבצים של xv6. קראו פרק 6, "מערות קבצים", מתוך ה vx6 book.
2. כפי שניתן לראות, ה inode במערכת הקבצים של xv6 תומכת ב 12 בלוקים ישירים וקיימת תמיכה ב single indirection בלבד. עליכם להבין את הקוד של פונקציית bmap במערכת הקבצים ולהרחיב את התמכה ל double indirection.
3. פרק 6 של xv6 book נותן הסבר מעולה על מבנה מערכת הקבצים של xv6. מפרק זה ניתן ללמוד לא רק על המבנה של מערת הקבצים של xv6, אלה גם לדעת באילו קבצים יושב הקוד של מערכת הקבצים ומשם הדרך להשלמת המשימה של מימוש ה double indirection קצרה מאוד. אבל בפרויקטים של open source לא תמיד מקבלים ספר נלווה כמו במקרה של xv6. לכן, מי שמעוניין להתנסות בביצוע המשימה ללא תמיכה,

יכול לעשות זאת ולמצוא את רשימת הקבצים שבהם מוזכרת מילה indirection ע"י חיפוש ב repository של הפרויקט:

```
https://github.com/mit-pdos/xv6-public/search?q=INDIRECT&unscoped_q=INDIRECT
```

ולהבין ישירות מהקוד מה עליו לשנות על מנת לממש את ה double indirection. שימו לב, רשימת הקבצים שהחיפוש מוצא רחבה מדי. לדוגמא, לקובץ entry.S אין נגיעה למערת הקבצים ואין צורך לשנותו.

4. בכל מקרה סיפקנו לכם את הקוד של xv6 (בספריה xv6-public) עם ציון כל המקומות בהן נדרשת השלמה. הריצו מתוך הספריה xv6-public את הפקודה:

```
make clean; grep -rn "Add code" *
```

ותקבלו רשימה של קבצים ושורות בהם יש להוסיף קוד התומך ב double indirection. שימו לב להערות (במידה וישנן) באותם המקומות בהם הנכם מתבקשים להוסיף קוד.

5. לאחד השלמת המישוש של double indirection בקוד של הגרעין, עליכם לבדוק נכונות המימוש ע"י הרצת regression tests שיושבים בקובץ usertests.c. סיפקנו לכם פונקציה bigfile המייצרת קובץ בגודל מירבי, כותבת בו תוכן, קרואת ממנו את התוכן הנכתב ומבצעת בדיקה מדגמית עם התוכן המקורי. הרצת התוכנית usertests מתבצעת משורת הפקודה של xv6. שימו לב שבמערכת הקבצים המקורית הגודל המירבי של הקובץ הוא 140 בלוקים ואילו לאחר המימוש של double indirection הרצת הפונקציה bigfile אמורה ליצור קובץ בגודל 16523 בלוקים.

6. שימו לב לסדר הפעולות המתבצע בעת עליית המערת xv6. הרצת הפקודה make qemu

<u>הגשה</u>

יש להגיש אך ורק קבצי קוד ששיניתם. אין להגיש קבצים מקומפלים. את הקבצים המוגשים יש לשים בקובץ ארכיון בשם exYZ.zip (כאשר YZ הנו מספר המטלה). הכנת קובץ ארכיון מתבצעת ע"י הרצת הפקודה הבאה משורת הפקודה של Linux:

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>

<u>בדיקה לאחר ההגשה</u>

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 הקבצים מהספריה החדשה עם המטלה שלכם לספרייה עם הקוד של xv6

- הרצת make qemu ווידוא שכל ה targets נוצרו ללא שגיאות וללא warnings
- הרצת בדיקות רלונטיות לוידוא תקינות הריצה של החלק המעשי

החלק העיוני (20%)

שאלה 1 (5%)

לפי מדיניות חדשה של תזמון זרוע הדיסק, הבקשות מוחזקות בתור לפי סדר הגעתן והראשונה שמטופלת היא הבקשה שהגיע אחרונה. מדיניות זו נקראת LIFO (last in first out).

א) מהו היתרון של המדיניות הזאת?

ב) מהו החיסרון של המדיניות הזאת?

שאלה 2 (5%)

מערכי דיסקים RAID level 2 ו RAID level 3מסוגלים להמשיך לעבוד כאשר אחד מהדיסקים במערך מתקלקל. יחד עם זאת, Level 2 דורש מספר רב יותר של דיסקים עודפים. אז מדוע יש בכלל עניין כלשהו בשיטה הזאת?

תזכורת - קוד המינג:

בהנתן מילה בת 4 סיביות:

סיבית b1	סיבית b2	סיבית b3	סיבית b4

קוד המינג שלה הוא:

P1	P2	B1	P3	B2	B3	B4

כאשר

P1 = Even Parity of b1, b2, b4

P2 = Even Parity of b1, b3, b4

P3 = Even Parity of b2, b3, b4

לדוגמא: המינג קוד של מילה בת 4 סיביות 1101 יהיה 1100110 (משמאל לימין)

שאלה 3 (5%)

גודלו של קובץ כלשהו יכול להיות בין 4Kb ל 4Mb בכל רגע נתון בחייו. איזו מבין 3 מדיניות הייתם בוחרים:

- הקצאה רציפה
- FAT
- I-Node

הניחו הנחות סבירות נוספות שדרושות. הדגימו את החישובים עליהם תבססו את ההחלטה.

שאלה 4 (5%)

תארו שיטות להגנה על ה capabilities.

הגשת החלק העיוני

החלק העיוני יוגש כקובץ Wordאו כקובץ pdf. שם הקובץ צריך להיות exYZ.pdf או exYZ.doc (כאשר YZהנו מספר המטלה).