מטלת מנחה (ממ"ן) 11

הקורס: "מערכות הפעלה"

חומר הלימוד למטלה: ראו פירוט בסעיף "רקע"

מספר השאלות: 5	משקל המטלה: 10
סמסטר: 2025א	מועד אחרון להגשה: 30.11.2024

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

החלק המעשי (70%)

כללי

בתרגיל זה נכיר את מבנה של מערכת הפעלה בכלל ומערכת ההפעלה XV6 בפרט . מערכת ההפעלה XV6 היא מערכת ממשפחת **LINUX** שפותחה לצורכי לימוד ע"י **MIT** . היא הרבה יותר פשוטה והרבה פחות נוחה(תרגישו את זה מיד בשימוש בה אפילו ב **CLI** שלא מאפשר שימוש בחצים למשל), אבל מצד שני מאפשרת להבין את קוד מערכת ההפעלה ולשנות אותו בקלות יחסית. היא לא מושלמת ויש בה באגים!

מטרות:

- הכרת מערכת הפעלה xv6
- הכרת ההיבטים המעשיים של מימוש קריאות מערכת
- הכרת מבני נתונים שונים של מערכת הפעלה
- הוספת קריאת מערכת חדשה
- הוספת פקודת מערכת חדשה ps שמדפיסה את מצב תהליכים במערכת
- **התנסות בבניה והרצה של מערכת הפעלה בצורה הקרובה למציאות(כשלא כל המידע זמין וצריך להבין ולמצוא אותו לבד) !**

רקע

א) פרק Makefile מחוברת "Ubuntu 24.04 programming environment, making first steps" (הורידו את החוברת מאתר הקורס).

ב) "Running and debugging xv6.pdf"(באנגלית, כולל הוראות דיבוג משורת הפקודה) ו/או "XV6 Instalation and EclipseConfig.pdf" (בעברית, מאחד התלמידים, כולל דיבוג ב ECLIPSE) מתוך maman11.zip. התקינו את המכונה הוירטואלית מאתר הקורס, סיסמת המנהל ubuntu .

ג) פרק 0 (עד PIPES בע' 13), פרק 1 ופרק 3(עד X86 protection בע' 40) מתוך https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev11.pdf .

אין צורך להתייחס לענייני ניהול זיכרון ראשי.

ד) expect - שפת סקריפט אינטראקטיבית : https://likegeeks.com/expect-command -
שפת סקריפט למיכון (automation) אינטראקציה עם פקודות shell ותוכנות אשר מורצות משורת הפקודה.

ה) במידת הצורך סרטונים על שימוש ודיבוג ב XV6 מאתר הקורס(בחלק ממ"נים). מספרי הממ"נים והדוגמאות בהם לא זהים לתוכן המטלות העכשווי.

תיאור המשימה

הוספת פקודת מערכת חדשה ps שמדפיסה את מצב תהליכים במערכת.

בקובץ maman11.zip תמצאו ספרייה עם מערכת ההפעלה xv6 שאין בה פקודה ps ואין קריאת מערכת הדרושה לביצועה, המטרה היא להוסיף אותם.

הסבר מפורט

1. הפעילו את מערכת ההפעלה xv6 כמתואר בסעיף ב' של "חומר קרע". הריצו את תוכנת ps , תקבלו את הפלט הבא :

```
cpu1: starting 1
cpu0: starting 0
sb: size 2000 nblocks 1954 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap s
tart 45
init: starting sh
$ ps
exec ps failed
$
```

הסיבה לשגיאה היא שפקודה ps כלל לא קיימת במערכת.

אחרי הוספת הפקודה תוצאת ההרצה צריכה להיות :

```
cpu1: starting
cpu0: starting
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap st
art 58
init: starting sh
$ ps
name     pid     state          ppid
init     1       SLEEPING       0
 sh      2       SLEEPING       1
 ps      3       RUNNING        2
 $
```

אפשר להריץ את פתרון ביה"ס לפי ההוראות ולראות את ההדפסה בפועל.

פתרון ביה"ס

להריץ מתוך תיקיית הממ"ן את make clean ו אחריו make qemuss

אפשר להריץ בשורת הפקודה של XV6 פקודת sh ואחריה שוב ps ולראות שנוסף עוד תהליך.

כדאי לעקוב אחרי מספרי ה PID ו PPID בפתרון ביה"ס ולהבין מי בן של מי ולמה. כמו כן מה התפקיד של תהליכים שונים.

2. הוסיפו את קריאת המערכת הדרושה ואת פקודת המערכת ps(אחרי ההוספה היא תופיע בין פקודות המערכת). כדי ש ps תוכל לבצע את עבודתה, צריך להוסיף קריאת מערכת מתאימה, **ראו בהמשך.**

כדי שהמטלה לא תיראה קשה, כדאי להתחיל במדריך שעושה משהו דומה(אבל לא מטפל בהדפסת PPID) :

https://github.com/raj-maurya/xv6-public_modifiedOS

אפשר למצוא בתוך xv6-modified **רוב המטלה עשויה(כולל** Makefile מתאים**).**
צריך לבצע שינויים קטנים.

בהמשך הסברים יותר מפורטים.

הדפסה מתוך הגרעין נעשית בעזרת cprintf ולא printf הרגילה. זה גם ההיגיון שהשם של קריאת המערכת

במטלה הוא cps . "c" - פלט ישיר ל CONSOLE ולא ל STDOUT .

סרטון שמסביר את תהליך ההוספה(באנגלית) כולל שינוי ב Makefile :

https://www.youtube.com/watch?v=21SVYiKhcwM

כדאי להשתמש במדריך שלמעלה במידה נכונה ולא להפוך שת פתרון המטלה להעתק הדבק בלבד!

- **לקריאת המערכת צריך להיות שם cps1xx** , כש xx הן 2 הספרות האחרונות של ת"ז של הסטודנט. לדוגמא, אם ת"ז 3135678**92** אז שם קריאת המערכת צריך להיות **cps192 !!!**
- **מספר קריאת המערכת צריך להיות כמו(שווה) לספרות אחרי cps , 192 בדוגמא הנ"ל.**
- **לקובץ ps.c צריך להיות שם ללא תוספת ספרות(ps.c בלבד) !!!**

אופן ביצוע המטלה:

כדי לבצע את המטלה, צריך להכניס שינויים לקבצים:

proc.c , syscall.c , usys.S , sysproc.c , user.h , defs.h , ליצור קובץ חדש ps.c ולהוסיף אותו לתיקיה של XV6 .

- **בקובץ syscall.c מותר(וצריך) לשנות 2 שורות.**
- **בקבצים defs.h , user.h מותר(וצריך) לשנות רק שורה/הכרזה/הצהרה/פקודה אחת.**
- **בקבצים usys.S , sysproc.c , ps.c , proc.c מותר(וצריך) לשנות בהתאם לנדרש.**

אי עמידה בכללים תביא לפסילת החלק המעשי !!!

דרך הפתרון שונה במקצת ממדריכים, **שימו לב שאסור לשנות(ולהגיש) את הקובץ syscall.h .**
כדי להצליח במטלה ללא אפשרות לשנותו, צריך להבין את התפקיד ואת המשמעות של המבנה הנתונים הרלוונטי בקובץ syscall.c (שורות 112-134) ואיך "לעקוף" את המגבלה הנ"ל. בנוסף, צריך לבצע שינוי(הוספה) בקובץ **usys.S** , יש בו דוגמא לשינוי שנעשה כהוספה "ידנית" של קריאת מערכת FORK ,
תפעלו בצורה דומה בשביל להוסיף קריאת מערכת הדשה. בקובץ **syscall.h** יש הערה לגבי קריאת מערכת FORK שמדמה את המצב שבמטלה כשאין אפשרות לשנות את תוכנו של הקובץ.
חשוב לציין, **שהמגבלה** נועדה רק לגרום להבנה ולא מהווה דרך מקובלת להכניס שינויים לקוד המערכת.

בנוסף, שימו לב שהפעולה עצמה של קריאת המערכת(מה שהיא מציגה) צריכה להיות שונה ממה שיש במדריכים.

כדי להדפיס את שדה PPID (לא ממומש בקישור הנ"ל) צריך למצוא אותו ב PCB של התהליך- מבנה struct proc בקובץ proc.h , שם השדה שונה(לא PPID) , ניתן למצוא בקלות ע"פ ההערות.

בשביל אחידות הפלטים בבדיקה, הדפסת שורת הכותרת של הפלט צריך לבצע בעזרת:

שימו לב לרווח בין \t ל \t // cprintf("name \t pid \t state \t \t ppid \n");

והפלט עצמו צריך להיות כמו בתמונה ופתרון ביה"ס.

שימו לב שהשדה PPID המודפס של INIT **צריך להיות 0 למרות** שבשדה המחזיק את PPID ב PCB יש מספר אחר, אנחנו מניחים שלאבא של התהליך ראשון במרחב המשתמש יש PID =0 , צריך לממש את זה במטלה ולמצוא את הדרך לזהות את תהליך INIT . אצל השאר בשדה המחזיק את PPID מופיע המספר הנכון.

- צריך להדפיס את כל התהליכים הקיימים(שנוצרו במערכת, ללא שורות ריקות בטבלת התהליכים). בשביל פשטות ואחידות הבדיקה **כל תהליך שלא נמצא במצב RUNNING אמיתי מודפס כ SLEEPING (במשמעות NOT RUNNING)** .

שלבי הביצוע:

a. תקראו את ההסבר על תהליך הוספת קריאת מערכת ל XV6 ואת תפקידים של הקבצים הרלוונטיים:
https://viduniwickramarachchi.medium.com/add-a-new-system-call-in-xv6-5486c2437573 הסבר הכי מתאים לצורכי המטלה בין מה שראיתי.
שימו לב שבד"כ בקובץ **sysproc.c** יש רק את "השלד" של קריאת המערכת שקורא לפונקציה עצמה שעושה את העבודה ונמצאת ב **proc.c** . בקישור למעלה קריאת המערכת קצרה מאוד, כמו שלד עצמו ולכן מיקמו אותה ב **sysproc.c** . בפתרון המטלה את הקוד ביצוע ממשי של קריאת המערכת צריך לשים ב **proc.c** .
ומתוך: https://www.ics.uci.edu/~aburtsev/238P/hw/hw5-syscall/hw5-syscall.html

רק פתיח והקטע Considerations .

שימו לב שתהליך עשיית המטלה דומה, אבל שונה במקצת ממדריכים.
המטרה להבין את התהליך ולהכיר את התפקיד של קבצים שונים.

b. תשנו את ה Makefile בשביל שיתאים לשינויים. עדיף להכיר את השימוש הבסיסי ב Makefile ולבצע את השינויים הנדרשים. במידת הצורך ניתן למצוא את Makefile מתאים בתוך הקבצים של מערכת xv6-modified (קישור).

c. אחרי ביצוע השינויים תריצו את המערכת מחדש, תבדקו שהמערכת החדשה(במקצת) מתפקדת כמצופה. תריצו ps ותראו שהפלט תקין.

d. אחרי סיום המטלה צריכים להיות ברורים המושגים הבאים והבדלים ביניהם:
מצב גרעין, מצב משתמש, קריאת מערכת, למה בכלל במקרה שלנו יש צורך בקריאת מערכת ולא מספיק תוכנית המשתמש, מספר קריאת מערכת, ממשק לקריאת מערכת, אופן הפעלה מעשית של קריאת מערכת(על פי מספר בעזרת INT), למה קריאת מערכת מופעלת בעזרת

INT ולא סתם קריאה לפונקציה, פונקציה(קוד) המבצעת את קריאת מערכת, תוכנית המשתמש שמפעילה את קריאת המערכת.
חשוב לשים לב שב XV6 יש 2 אימג'ים(IMAGES) שמדמים 2 מערכות קבצים, אחת של הגרעין והשנייה של מערכת עצמה עם אפשרות לשמור בה את הקבצים של משתמש.

e. אופציונאלי, אבל חשוב מאוד: לדבג את עליית המערכת כמו שמוסבר בסרטון באתר ולעקוב אחרי השלבים של עליית הגרעין ומעבר למרחב המשתמש.

f. אופציונאלי, אבל חשוב: להכיר את פעולת ה SHELL של המערכת(קובץ sh.c) ובפרט, זיהוי הפקודה, הבדל באופן ביצוע בין פקודות SHELL (הפקודות הפנימיות) לבין פקודות המערכת (יצירת התהליך המבצע בעזרת FORK והחלפת תוכנו לפקודת המערכת בעזרת EXEC).

בדיקה סופית

1. לאחר תיקון הבאג הריצו make clean; make qemu .
ודאו בפעם נוספת שאתם מסוגלים להריץ את ps ושפלט של הפקודה תקין
2. כעת המשיכו לבדיקות regression שמטרתן לוודא כי כל הבדיקות (tests) עוברים בהצלחה. לשם כך כבו את QEMU.
3. הריצו משורת הפקודה של מערכת ubuntu 16.04 **מתוך התיקייה של xv6** פקודה הבאה:

```
./runtests.exp my.log
```

אם צריך, תתנו הרשאות הרצה לקובץ runtests.exp .

4. ודאו כי תוכנת סקריפט יצאה עם סטאטוס 0 מיד לאחר סיומה(ערך הסיום נכתב גם לקובץ).

```
$ echo $?
0
```

5. להכרות כללית עם expect מומלץ(לא חובה) לקרוא את פרק ד' של "חומר רקע".

פתרון ביה"ס

להריץ מתוך תיקיית הממ"ן את make clean ו אחריו make qemuss

הגשה בזיף אחד ביחד עם החלק העיוני!

יש להגיש אך ורק את הקבצים שהיה צורך לשנות/להוסיף:

(ps.c , proc.c , syscall.c , usys.S , sysproc.c , user.h , defs.h ו Makefile) בלבד.

אין להגיש קבצים נוספים ו/או מקומפלים. ראה הוראות הגשה כלליות בחוברת הקורס.

את הקובץ/הקבצים המוגש/ים יש לשים בקובץ ארכיון בשם exYZ.zip (כאשר YZ הנו מספר המטלה). עדיף להכין את הארכיון בפורמט זיפ ZIP ב WINDOWS . אם אין אפשרות, ע"י הרצת הפקודה הבאה משורת הפקודה של Ubuntu: zip exYZ.zip <exYZ files>

<u>הערה חשובה: בתוך כל קובץ קוד שאתם מגישים יש לכלול כותרת(בהערה) הכוללת תיאור הקובץ, שם הסטודנט ומספר ת.ז.</u>

בדיקה לאחר הגשה

לאחר ההגשה יש להוריד את המטלה (חלק מעשי/עיוני) משרת האו"פ למחשב האישי' לבדוק תקינות של הקבצים המוגשים (לדוגמא, שניתן 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 רלוונטיות: וידוא תקינות הריצה של החלק המעשי

החלק העיוני (30%)

שאלה 2 (5%)

א) מהי פעולת ה TRAP? תארו מתי ובשביל מה היא מתבצעת ומה קורה בעת ביצועה.

ב) הסבירו מה קורה בעת הקריאה לפונקצית write של ה C library . בפרט הסבירו כיצד עוברים הפרמטרים של ה write למערכת הפעלה Linux וכיצד המערכת מטפלת ב write.

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

שאלה 3 (15%)

במערכת הפעלה LINUX קיים מנגנון איתותים(סיגנלים) SIGNALS חשוב ושימושי **שהשימוש בו בתוך אפליקציה** יכול להיות סינכרוני ו/או א-סינכרוני.

כדי להכיר את המנגנון תקראו את המאמר: https://cs341.cs.illinois.edu/coursebook/Signals

תבינו היטב את ההבדל בין שימוש סינכרוני ל א-סינכרוני.

בין קבצי הממ"ן יש קבצי קוד שמדגימים את השימוש הבסיסי בשתי צורות הסיגנלים.

הקבצים באים רק לעזור, אין חובה להריץ אותם ולבצע את מה שבהערות, אך זה עוזר להבנת העניין.

אם הכל ידוע או מובן בלי דוגמא, אין צורך אפילו להסתכל על הקודים האלה.

א) ממשו את "**סמפור" בינארי יחיד(ללא שם) על בסיס שימוש סינכרוני בסיגנל וללא שימוש במנגנוני סנכרון נוספים**. ה"סמפור" מיועד לשימוש ע"י **מספר תהליכונים THREADS של אותו תהליך(לא תהליכים שונים).**

המירכאות בגלל שה"סמפור" המיועד **אינו חייב לכלול את מבני הנתונים** הפנימיים הלא חיוניים לתיפקודו.

ממשו את הפונקציות הבאות **בשפת C:**

- void sem_init(int status) לאתחול הסמפור למצב מסומן(פתוח, = 1) או לא מסומן(סגור, = 0).
- void sem_down() להורדה(סימון כתפוס, המתנה) של סמפור.
- void sem_up() לשחרור(סימון כדלוק/פינוי) של סמפור.
- הכרזה ואתחול של משתנים שחייבים להיות גלובאליים עם ציון בצורה חופשית שהם גלובליים.

שימו לב שבצורת השימוש המדוברת, הסיגנל צריך להיות חסום לפני הפעלת sigwait .

כמו כן, **שימו לב** שפונקציה **kill שולחת סיגנל** לתהליך המיועד **ולא הורגת**(חוץ מסיגנל ספציפי)!

הסיגנל שנשלח לתהליך "מגיע" לכל ה THREADS שלו וה THREAD ויקבל(יתפוס) אותו ראשון, ינקה(ימחק) אותו מרשימת הסיגנלים הממתינים.

הדרך **האוניברסאלית** לשליחת הסיגנל ללא משמעות קבוע במערכת לתהליך עצמו היא:

kill(getpid() , SIGUSR1); . אפשר להשתמש בכל אפשרות תקינה אחרת.

לצורך התרגיל הבסיסי ובשביל הפשטות אין צורך לחסום בנפרד לכל תהליכון THREAD , אלא לחסום בתוך התהליך שיחול על כל ה THREADS שייוצרו בתוכו(לחסום בתוך הפונקציה sem_init(int status)).

אפשר(אך לא חובה) להיעזר בדוגמא:

https://www.ibm.com/docs/en/zos/2.4.0?topic=functions-sigwait-wait-asynchronous-signal

אין חובה לכתוב תוכנית שלמה הכוללת יצירת THREADS והסנכרון ביניהם בעזרת הפונקציות שמימשתם, אך זה מוסיף להבנה וניסיון. בנוסף, כתיבתה והרצתה תאפשר לבדוק את נכונות הפתרון.

ב) האם בדרך דומה(ולא מסובכת משמעותית יותר) ניתן לממש את הסמפור המיועד לסנכרון בין מספר **תהליכים? תנו נימוק מילולי, אין צורך בכתיבת קוד.**

שאלה 4 (5%)

תקראו את מאמר שמסכם את הבדלים בין user threads ו kernel threads ואת מאמר שמסביר מודלים שונים של מימוש מנגנון threads . תענו לשאלות הבאות:

א. האם **M:1 model** מאפשר לנצל מספר ליבות במעבד CPU cores ? נמקו.

ב. האם ב **M:1 model** חסימת אחד מ user threads תגרום לחסימת כל התהליך? נמקו.

ג. מה המשמעות של מושגים user thread ו kernel thread ב **1:1 model** ?

שאלה 5 (5%)

א) הוכיחו כי בפתרון של Peterson ל 2 תהליכים(ע' 52 במדריך הלמידה), תהליכים אינם ממתינים זמן אינסופי על מנת להיכנס לקטע קריטי. בפרט הוכיחו כי תהליך שרוצה להיכנס לקטע קריטי לא ממתין יותר ממה שלקח לתהליך אחר להיכנס ולעזוב את הקטע הקריטי.

ב) האם פתרון יישאר תקין אם יחליפו את סדר ביצוע 2 שורות הקוד(הראשונה תתבצע אחרי השניה):

```
interested [# ] = TRUE;
turn = #;
```

הסבירו למה כן או הפריכו ע"י דוגמא נגדית.

דבר כזה אכן יכול לקרוא בעקבות אופטימיזציה(Instruction reordering).

הגשת החלק העיוני

החלק העיוני יוגש כקובץ Word או pdf עם **שם exYZ.docx או exYZ.pdf** (כאשר YZ הנו מספר המטלה)

בתוך אותו zip עם החלק המעשי. אין להגיש יותר מזיפ אחד בסה"כ!