

# מערכות הפעלה

234123

סמסטר

אביב

תשע"ח

## מגישים:

|             |           |                      |
|-------------|-----------|----------------------|
| עידו יחזקאל | 204397368 | idoye                |
| שם מלא      | מספר ת.ז. | דואר אלקטרוני טכניון |
| אוהד זוהר   | 305285694 | ohadzo               |
| שם מלא      | מספר ת.ז. | דואר אלקטרוני טכניון |

# Homework 1 Dry

**Due Date: 24/04/2018 23:30**

Teaching assistant in charge:

- Shalev Kuba

**Important:** the Q&A for the exercise will take place at a public forum Piazza only. 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 hw1 , put them in the hw1 folder

Only the TA in charge, can authorize postponements. In case you need a postponement, contact him directly at [234123cs@gmail.com](mailto:234123cs@gmail.com) .

Dry part submission instructions:

1. Please submit the dry part to the electronic submission of the dry part on the course website.
2. The dry part submission must contain a single dry.pdf file containing 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.
3. Only typed submissions will be accepted. Scanned handwritten submissions will not be accepted.
4. Only PDF format will be accepted.
5. You do not need to submit anything in the course cell.
6. When you submit, **retain your confirmation code and a copy of the PDF**, in case of technical failure. It is **the only valid proof** of your submission.

יש לנמק כל תשובה, תשובות ללא נימוק לא יתקבלו.

## שאלה 1 (50 נק')

1. (30 נק') שאלה זו מתייחסת לשלבי תהליך הקריאה לשירות מערכת ההפעלה מתהליך המשתמש. עבור כל שלב בתהליך המתואר ציין אם הוא מבוצע על-ידי קוד משתמש (כידוע קוד עם  $CPL=3$ ), קוד בגרעין ( $CPL=0$ ) או על-ידי החומרה (מעבד) - הקף בעיגול את המתאים. עבור כל השלבים, הוסף הסבר מה השלב מבצע. הניקוד של כל שורה הינו 3 נק'.

| שלב   | מבוצע על-ידי<br>(הקף בעיגול)      | הסבר  |
|---|-----------------------------------|---|
| קריאה לפונקציית מעטפת עם פרמטרים במחסנית  | קוד גרעין/<br>קוד משתמש/<br>חומרה | פונקציית המעטפת אחראית לשליחת הפרמטרים לשירות והפעלת קריאת המערכת עצמה  |
| העברת הפרמטרים לרגיסטרים  | קוד משתמש                         | פונקציית המעטפת היא שאחראית להעביר את הארגומנטים לשגרת הטיפול בפסיקה ברגיסטרים מכיוון שהמחסנית משתנה. לכן הפעולה מתבצעת בקוד משתמש. |
| פקודת <code>int 0x80</code>   | קוד משתמש                         | פונקציה המעטפת יוזמת את הקריאה לפסיקה וכן נמצאת בקוד משתמש.   |
| מציאת פונקציית הטיפול בפסיקה  | חומרה                             | החומרה ניגשת ל-IDT וניגשת לשגרת הטיפול בפסיקה הנמצאת ב- <code>0x80</code> .   |
| שמירת הרגיסטרים <code>ss, esp, eflags, cs, eip</code> במחסנית החדשה לאחר המעבר ל- <code>kernel mode</code>  | חומרה                             | עקב השינוי במחסניות ומעבר למחסנית הגרעין המעבד שומר את המצב של המשתמש לפני הפסיקה.  |
| שמירת <code>orig_eax</code> במחסנית, אשר מייצג את מספר השירות המבוקש, וקריאה ל- <code>SAVE_ALL</code>   | קוד גרעין                         | שגרת הטיפול בפסיקה אחראית לשמירת מספר קריאת המערכת ואז מבצעת קריאה למאקרו <code>SAVE_ALL</code> אשר נמצאת בקוד הגרעין.              |
| איתור כתובת פונקציית השירות ב <code>syscall_table</code> ובדיקה שמספר השירות המבוקש (מספר קריאת המערכת) נמצא בטווח החוקי של מספרי השירות האפשריים | קוד גרעין                         | שגרת הטיפול בפסיקה אחראית לאתר את פונקציית השירות בעזרת מספר קריאת המערכת. שקיבלה מפונקציית המעטפת.                                 |
| קריאה לפונקציית השירות  | קוד גרעין                         | לאחר איתור פונקציית השירות, שגרת הטיפול בפסיקה אחראית לקריאה של פונקציית השירות ע"פ קונבנציית הקריאה לפונקציית C.                   |
| ביצוע שגרת השירות   | קוד גרעין                         | שגרת השירות מתבצעת ע"י הגרעין (זוהי תכלית הגרעין).  |

|   |           |   |
|---|-----------|---|
| קוד השגיאה מוחזר ברגיסטר<br>eax מהגרעין אך פונקציית<br>המעטפת היא זו ששומרת את<br>ערכו המוחלט במשתנה errno. | קוד משתמש | בהנחה והתרחשה שגיאה בקריאת<br>המערכה, כתיבת קוד השגיאה<br>errno-ל |
| מבוצע ע"י פונקציית המעטפת<br>אשר נמצאת בקוד המשתמש.   | קוד משתמש | החזרת ערך קריאת המערכת<br>למשתמש                                  |

בסעיפים הבאים (2,3,4) הוצעו שינויים במנגנון הטיפול בפסיקות. בכל הסעיפים הבאים מערכת  
ההפעלה והמעבד נותרים ללא שינוי, מלבד השינוי המוצע.  
2. (6 נק') השינוי המוצע: בקבלת פסיקה ישמר גם רגיסטר ebx, בנוסף לרגיסטרים שנשמרו  
במימוש המקורי על המחסנית הגרעין. להלן שרטוט הממחיש את המימוש החדש:

| סדר השמירה החדש | סדר השמירה המקורי | בסיס המחסנית<br> <br> <br> <br>V<br>ראש המחסנית |
|-----------------|-------------------|---|
| ss              | ss                |   |
| esp             | esp               |   |
| eflags          | eflags            |   |
| cs              | cs                |   |
| eip             | eip               |   |
| ebx             |                   |   |

בנוסף, כדי להשלים את המימוש, הוצע שפקודת iret תשלוף את רגיסטר ebx ולאחר מכן  
ישלפו שאר הרגיסטרים כפי שהיה במימוש המקורי.  
האם המימוש תקין? אם לא, איזו בעיה עלולה להיווצר במימוש?

**תשובה:**

**המימוש תקין. זאת מכיוון ששמירת הרגיסטר הנוסף בעת הקריאה לפסיקה ושליפתו בעת  
החזרה לא תשנה דבר מפני ששינוי זה אינו מפריע לקוד הגרעין להשתמש במחסנית.  
(נשים לב ששמירה זו אמנם תקינה אך מיותרת כי רגיסטר זה שמר תחת המאקרו SAVE\_ALL).**

3. (7 נק') השינוי המוצע: עם קבלת פסיקה לא מחליפים מחסניות, דוחפים את eflags, cs, eip  
בלבד בראש המחסנית הנוכחית, ועוברים לבצע את שגרת הטיפול בפסיקה.  
בהתאם, בחזרה מפסיקה שולפים את שלושת הערכים שנדחפו ונשארים במחסנית הנוכחית.  
האם המימוש תקין? אם לא, איזו בעיה עלולה להיווצר במימוש?

**תשובה:**

**המימוש אינו תקין. זאת כי ללא החלפת המחסניות קוד המשתמש יכול לקבוע איפה הגרעין  
יאתחל את המחסנית שלו ע"י שינוי ערך הרגיסטר esp וכך לגרום לגרעין לאתחל מחסנית  
בכתובות לא חוקיות (קוד זדוני).**

4. (7 נק') השינוי המוצע: בקבלת פסיקה 128 כדי לחסוך בשמירת רגיסטרים, הוחלט לא לשמור ולשחזר את רגיסטר eflags. להלן שרטוט הממחיש את המימוש החדש:

| סדר השמירה החדש | סדר השמירה המקורי | <div style="text-align: center;"> בסיס המחסנית<br/>  <br/>  <br/>  <br/> V<br/> ראש המחסנית </div> |
|-----------------|-------------------|--|
| ss              | ss                |  |
| esp             | esp               |  |
| cs              | eflags            |  |
| eip             | cs                |  |
|                 | eip               |  |

בנוסף, כדי להשלים את המימוש, פקודת iret תשלוף את הרגיסטרים לפי הסדר כך שלא משחזרים את ערך eflags כמקודם, כלומר גם בחלק של שליפת הרגיסטרים נבצע את התיקון הדרוש.

האם המימוש תקין? אם לא, איזו בעיה עלולה להיווצר במימוש?

**תשובה:**

המימוש אינו תקין. אי שמירת הרגיסטר eflags גורמת לכך שבעת חזרה לקוד משתמש ייתכן כי אבד מידע חשוב עבור קוד המשתמש אשר חושב לפני הפסיקה.  
לדוגמא: פקודת cmlt שומרת את תוצאת ההשוואה בדגלים (sign flag, zero flage...) נשים לב כי אם מיד לאחר פקודת cmlt מתבצעת פסיקה אשר גורמת לשינוי ברגיסטר eflags, בעקבות המימוש החדש נאבד את תוצאת ההשוואה, הקפיצה בקוד המשתמש לא תבצע על סמך ההשוואה של המשתמש.

## שאלה 2 (50 נק')

1. (18 נק')

א. (10 נק') ניר, ששונא לחכות, וגם מאמין במשפט "מה ששנוא עליך אל תעשה לתהליךך", החליט שבתוכניות מחשב שהוא כותב, הוא לעולם לא ישתמש בקריאת המערכת wait(). ליאור העיר לניר שאם לא ישתמש בקריאת המערכת הנ"ל ייאגר לו מידע בזיכרון על תהליכיו אשר סיימו להתבצע אך לא בוצע להם wait ("זומבים"), האם ליאור צודק? הסבר את טענתך.  
הערה: ניתן להניח כי ניר לא כותב תכניות בהן קיים תהליך שרץ זמן רב.

**תשובה:**

ליאור צודק. זאת מכיוון שעבור תהליך כלשהו אשר הסתיים ולא נעשה עבורו wait, מערכת ההפעלה תשמור פרטים אודות תהליך זה. פרטים אלו כוללים למשל את ה-pid של התהליך והסטטוס שלו. זאת כדי שבעת הקריאה ל-wait() עבור תהליך זה, מערכת ההפעלה תוכל להחזיר נתונים אלו. בנוסף, מערכת ההפעלה לא תמחזר את ה-pid של תהליך zombie עד אשר הוחזר מקריאת wait() ולכן נשמר pid של תהליך זה.

ב. (8 נק') שקד, שלמד על קריאת המערכת `fork()`, רצה להתנסות בבית בשימוש בה, ולכן כתב את קטע הקוד הבא:

```
int main(){
    int forkId=fork();
    if(forkId==0){//son
        printf("hey father, I am your son\n");
    }else{//father
        printf("hey son, I am your father\n");
    }
    return 0;
}
```

למרבה הצער, על המסך הודפס הפלט הבא (בהרצה מסוימת):

hey son, I am your father

hey father, I am your son

שקד התבאס מאוד שכן רצה שקודם הבן ידפיס למסך את ההודעה ורק לאחר מכן האב ידפיס את ההודעה שלו. עזרו לשקד, ע"י הוספת שורת קוד אחת בלבד, לגרום לתוכנית להדפיס **בכל הרצה** את הפלט:

hey father, I am your son

hey son, I am your father

**תשובה:**

**מכיוון שתהליך האב וגם תהליך הבן ניגשים בצורה לא מתואמת למשאב משותף, המסך, יש להוסיף קריאה ל `wait()` מתהליך האב על מנת לוודא שתהליך האב יגש למסך רק לאחר שתהליך הבן יסתיים.**  
**שורת הקוד שיש להוסיף:**

```
int main(){
    int forkId=fork();
    if(forkId==0){//son
        printf("hey father, I am your son\n");
    }else{//father
        wait(NULL);
        printf("hey son, I am your father\n");
    }
    return 0;
}
```

2. (15 נק') כזכור, מתאר התהליך מאוחסן ביחד עם מחסנית הגרעין שלו בקטע זיכרון בגודל 8KB המתחיל בכתובת מיושרת.  
חברת נינוקס, החליטה לפתח מערכת הפעלה מודרנית יותר מהמערכת הנלמדת בתרגולים. בפרט, החברה טענה שלא יתכן שגודל כל מתאר תהליך יהיה מוגבל בגודלו, ולכן הפרידה את מתאר התהליך ממחסנית הגרעין (מתאר התהליך ומחסנית הגרעין כבר אינם צמודים כפי שנלמד בתרגולים) כך שגודל מתאר התהליך אינו מוגבל במערכת החדשה. נינוקס, שהעתיקה מלינוקס את קוד הקרנל הנלמד בתרגולים בלי לשנות דבר מלבד הפרדת מתאר התהליך ממחסנית הגרעין, הופתעה לגלות, יום לפני ההפצה של המערכת, שכאשר מבצעים קריאת מערכת שדורשת גישה למתאר התהליך, המערכת קורסת. עזרו לנינוקס להבין היכן הטעות שלה. על תשובתכם להיות מפורטת.

### תשובה:

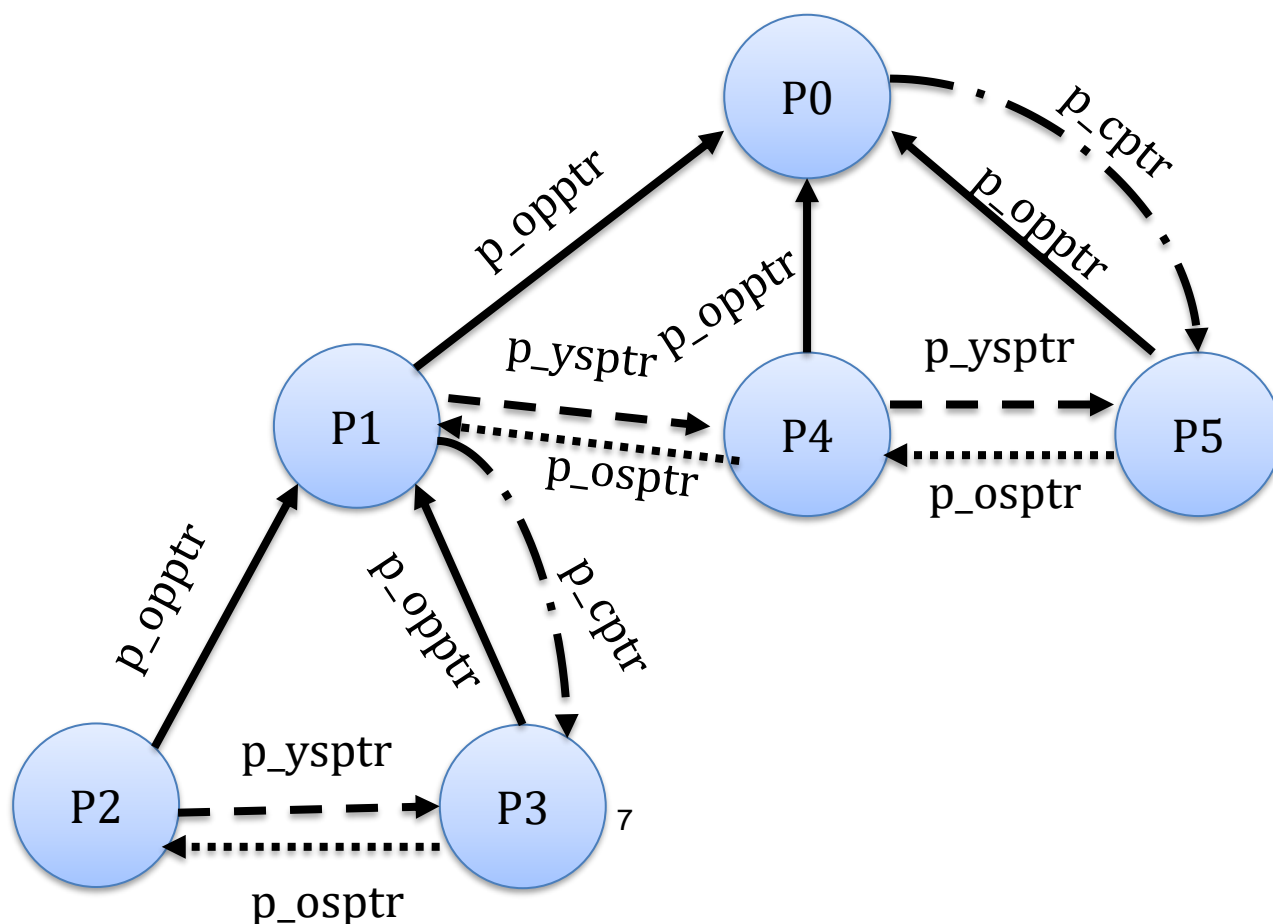
צורת האחסון של מתאר התהליך יחד עם מחסנית הגרעין שלו מאפשרת דרך פשוטה לשלוף את הכתובת של מתאר התהליך הנוכחי מתוך הרגיסטר esp כאשר המעבד בקוד הגרעין. בהתאם למה שראינו בתרגול, איפוס 13 הביטים הנמוכים של esp יניב את כתובת מתאר התהליך (כאשר קטע הזיכרון בגודל קבוע של 8KB). בנוסף, המאקרו current משתמש בשיטה זו על מנת לאחסן את כתובת מתאר התהליך בערך המוחזר. מכיוון שחברת נינוקס לא שינתה את קוד הגרעין (וכן את מאקרו current) הרי שהמאקרו מוביל לקריסת המערכת כי אינו מחזיר את הכתובת הנכונה.

3. (17 נק')

בשאלה זו נדון בקשרי המשפחה כפי שבאים לידי ביטוי בשדות התהליך ( $p\_opptr$ ,  $p\_ysptr$ , ...).  
ונלמדו בתרגולים:

א. (10 נק') עבור קטע הקוד הבא, ציירו את הגרף המתאר את קשרי המשפחה, כנלמד בתרגולים, כפי שנראה במערכת רגע לפני שתהליך כלשהו מסתיים (ניתן להניח כי כל התהליכים בתכנית נוצרים לפני שתהליך כלשהו נגמר). הקפידו לרשום על כל חץ את שם השדה ובתוך הצומת רשמו את המחרוזת שאותה התהליך מדפיס:

```
int main(){//father
    printf("P0");
    int forkId=fork();
    if(forkId==0){
        printf("P1");
        forkId=fork();
        if(forkId==0){
            printf("P2");
            return 0;
        }
        forkId=fork();
        if(forkId==0){
            printf("P3");
        }
        return 0;
    }
    //more code on the right
    //(continue)
    .
    .
    for(int i=4;i<6;i++){
        forkId=fork();
        if(forkId==0){
            printf("P%d",i);
            return 0;
        }
    }
    return 0;
}
```





ב. (7 נק') תנו דוגמה לקריאת מערכת שנלמדה בתרגול, שבה משתמשים בקשרי המשפחה כדי לבצעה? הסבירו איך בא לידי ביטוי השימוש בקשרי המשפחה בה:

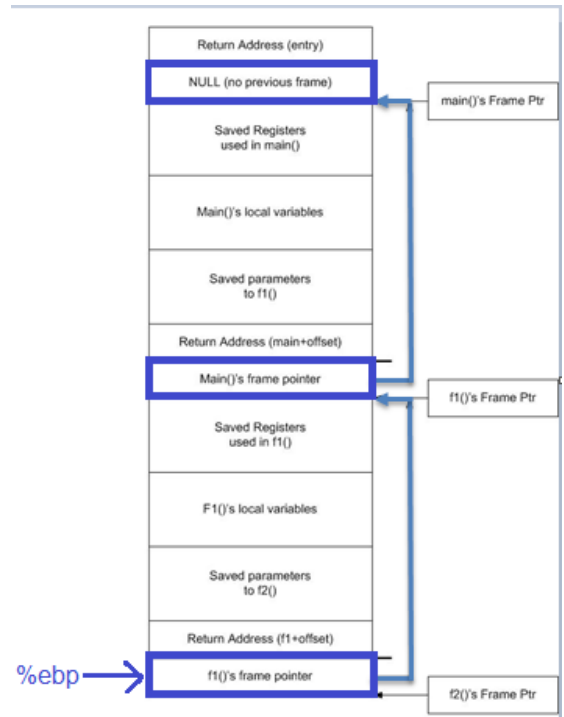
**תשובה:**

**קריאת המערכת getpid() משתמשת בקשרי משפחה. הפקודה מחזירה את pid של תהליך האב וזאת ע"י שימוש המצביע p\_optr.**

## שאלת בונוס (5 נק')

כידוע, ניתן להסתכל על ערכי הרגיסטר `ebp` ששמורים בבסיסי מסגרות הפונקציות, כרשימה מקושרת, כך שהאיבר הראשון של הרשימה נמצא ברגיסטר `ebp`. מצורף איור, שבו הודגשו בסיסי הפונקציות בכחול, יחד עם המצביעים (שהם למעשה הערכים אשר שמורים במחסנית), זאת כדי שתוכלו לראות בצורה נוחה יותר את הרשימה שנוצרת.

איזה כלי (שכולכם מכירים ממת"מ) מבצע שימוש נפוץ ברשימה זו. בתשובתכם הסבירו כיצד כלי זה משתמש ברשימה.



### תשובה:

הכלי המדובר הוא כמובן debugger שמאפשר הצגת מסגרות מחסניות הפונקציות לפי סדר הקריאה שלהן. זאת ע"י שימוש בערכי `ebp` השמורים בבסיסי המסגרות כרשימה מקושרת. לכל איבר ברשימה (מחסנית הפונקציה) ה debugger מחלץ את הפרטים הרלוונטים וממשיך לאיבר הבא (למחסנית הפונקציה הגבוהה יותר בהיררכיה).