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:
   1. The first page should contain the details about the submitters - Name, ID number and email address.
   2. 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 (הקף בעיגול)** | **הסבר** |
| קריאה לפונקציית מעטפת עם פרמטרים במחסנית | קוד גרעין/  **קוד משתמש**/  חומרה | **פונקציית המעטפת אחראית לשליחת הפרמטרים לשירות והפעלת קריאת המערכת עצמה** |
| העברת הפרמטרים לרגיסטרים | קוד גרעין/  קוד משתמש/  חומרה | פונקציית המעטפת לקוד המשתמש אחראית לשמור את הפרמטרים הרלוונטים לרגיסטרים לפני sys\_call כפי שנלמד שהעברת הפרמטרים לשיגרה היא באמצעות רגיסטרים בלבד-ולא על המחסנית כי במעבר בין קוד משתמש לקוד גרעין מחליפים מחסניות  (לדוגמה מעביר את מספר השירות המבוקש, ברגיסטר eax)  במידה ויש יותר מ6 פרמטרים היא מעבירה כתובת למבנה נתונים שמכיל את כל הפרמטרים. |
| פקודת int 0x80 | קוד גרעין/  קוד משתמש/  חומרה | הפקודה int0x80 נקראת על ידי פונקציית המעטפת וזו פסיקת תוכנה שמבצעת מעבר לאיזור הגרעין ומפעילה את שגרת הטיפול עבור קריאת המערכת. |
| מציאת פונקציית הטיפול בפסיקה | קוד גרעין/  קוד משתמש/  חומרה | תפקיד ה system\_call הוא לנתב את הבקשה לפונקציה המתאימה לפי מספר השירות שהועבר לה בעזרת הרגיסטר eax. מחפשת את המספר המבוקש בטבלה sys\_call\_table. |
| שמירת הרגיסטרים ss,esp,eflags,cs,eip במחסנית החדשה לאחר המעבר ל-kernel mode | קוד גרעין/  קוד משתמש/  חומרה | בזמן הקפיצה לשגרת הטיפול בפסיקה - החומרה שומרת את הרגיסטרים במחסנית. |
| שמירת orig\_eax במחסנית, אשר מייצג את מספר השירות המבוקש, וקריאה ל-SAVE\_ALL | קוד גרעין/  קוד משתמש/  חומרה | נשמר על ידי שגרת הטיפול בפסיקות - system\_call . |
| איתור כתובת פונקציית השירות ב syscall\_table ובדיקה שמספר השירות המבוקש (מספר קריאת המערכת) נמצא בטווח החוקי של מספרי השירות האפשריים | קוד גרעין/  קוד משתמש/  חומרה | תפקיד שגרת הטיפול בפסיקות - system\_call .  מבצע בדיקה האם מספר קריאת השירות חוקי – בין 0 ל256 – אם לא מחזירה ENOSYS ל esp+24 כדי לדרוס את eax עבור ערך ההחזרה. במידה ולא קיימת קריאת שירות עם מספר מתאים מתבצעת קריאה לפונ' שמחזירה באותו אופן ENOSYS.  במידה ולא הייתה בעיה קופצים לתוית no\_bad\_sys ונגשים לsyscall\_table לתא המתאים ומקבלים את כתובת של השירות. |
| **שלב** | **מבוצע על-ידי (הקף בעיגול)** | **הסבר** |
| קריאה לפונקציית השירות | קוד גרעין/  קוד משתמש/  חומרה | תפקיד שגרת הטיפול בפסיקות - system\_call .  בדיקה אם מספר השירות תקין – ואם כן הפעלת פונקצית ביצוע השירות על ידי  call \*sys\_call\_table(0,%eax,4) |
| ביצוע שגרת השירות | קוד גרעין/  קוד משתמש/  חומרה | מתבצע על ידי הפונקציה המתאימה שכתובתה נמצאה קודם לכן בsys\_call\_table |
| בהנחה והתרחשה שגיאה בקריאת המערכה, כתיבת קוד השגיאה לerrno- | קוד גרעין/  קוד משתמש/  חומרה | לקוד הגרעין אין גישה ל errno – מעטפת קוד המשתמש כותבת את הערך המוחלט של ערך השגיאה המתאים (נמצא בerrno.h) – שחוזר כמספר שלילי ברגיסטר eax והחזרתו לקוד המשתמש. |
| החזרת ערך קריאת המערכת למשתמש | קוד גרעין/  קוד משתמש/  חומרה | רקע: פונק' ביצוע השירות מחזירה ערך ל system\_call שמעבירה את הערך למחסנית במקום שמונח הערך של eax. בעזרת restore\_all הערך מועבר לרגיסטר eax במהלך החזרה לפונ' המעטפת.  תפקידה של פונ' המעטפת הוא לבדוק את ערך החזרה בeax, אם קרתה תקלה המעטפת תחזיר -1 וסוג התקלה יימצא בernno. אחרת, תחזיר את ערך ההחזרה. |

בסעיפים הבאים (2,3,4) הוצעו שינויים במנגנון הטיפול בפסיקות. בכל הסעיפים הבאים **מערכת ההפעלה והמעבד נותרים ללא שינוי, מלבד השינוי המוצע**.

1. (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 המקורי** | **סדר השמירה החדש** |
| בסיס המחסנית | ss | ss |
| | | esp | esp |
| | | eflags | eflags |
| | | cs | cs |
| V | eip | eip |
| ראש המחסנית |  | **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 , בכל מקרה הערך של ebx נשמר במחסנית על ידי המאקרו save\_all ומשוחזר בסיום ריצת הפונקציה על ידי restore\_all, כאשר נשמור אותו שוב על המחסנית ונשחזר את ערכו על ידי iret לא ישתנה דבר – הערך הקודם של רגיסטר זה יטען גם ככה חזרה .

1. (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:**

כפי שנלמד בתרגול – החלפת המחסניות מונע מקוד זדוני להשתמש במחסנית הגרעין ובכך להשתלט על המערכת. דבר זה יכול להגרם על ידי

1. (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 המקורי** | **סדר השמירה החדש** |
| בסיס המחסנית | ss | ss |
| | | esp | esp |
| | | eflags | cs |
| | | cs | eip |
| V | eip |  |
| ראש המחסנית |  |  |

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 מכיוון שהקוד שהתבצע עד לקבלת הפסיקה עלול היה לשנות את הערכים הנמצאים בדגלים הנמצאים בeflag כתוצאה מחישובים שבוצעו. אם לא נשמור את eflags לפני המעבר לביצוע הפסיקה (ונשחזר אותם לאחר מכן) ייתכן שערכי הדגלים ישתנו במהלך הפסיקה והקוד המקורי יתנהג באופן לא צפוי. דבר זה עלול להגרם מכך שקוד המקור יתבסס על ערכים שונים מאלו שהיו לפני קבלת הפסיקה.

**שאלה 2 (50 נק')**

1. (18 נק')
2. (10 נק') ניר, ששונא לחכות, וגם מאמין במשפט "מה ששנוא עליך אל תעשה לתהליכך", 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 טועה , אמנם אם ניר לא יכתוב בעצמו wait ויחכה לסיום תהליכיו אכן ייאגר בזיכרון מידע שלא ישוחרר **על ידו ,** אך כחלק ממערכת ההפעלה , התהליך init תפקידו לחכות לסיום כל בניו (בסופו של דבר כל תהליכי המערכת – כי כאשר קיים תהליך שאביו השתחרר , הוא ישירות הופך להיות יתום ומצביע ישירות על init ) . לכן גם אם ניר לא ידאג בעצמו לשחרור המידע – התהליך init יבצע בדיקה אילו תהליכים שהם בניו סיימו ומבצע עליהם wait וכך נמנע זיהום של המערכת.

1. (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**:

}else{//father

Waitpid(forkId,NULL,0){ ///wait(NULL)

printf("hey son, I am your father\n");

}

}

הוספת השורה גורמת לתהליך לחכות עד לסיום תהליך הבן דבר אשר יבטיח כי קוד הבן ירוץ ויסתיים ורק לאחר מכן קוד האב ימשיך.

1. (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**   
במערכת ההפעלה לינוקס כאשר מבצעים קריאת מערכת שדורשת גישה למתאר התהליך משתמשים במאקרו current – אשר מחזיר מצביע למתאר התהליך הנוכחי באופן הבא: בעת הקצאת תהליך חדש מ"ה מקצה עבורו בלוק זכרון בגרעין בגודל 8KB – המוקצה תמיד בכתובת שהיא כפולה של 8KB כאשר בקצה הבלוק נמצא מתאר התהליך. כדי לגשת במהירות למתאר התהליך (ללא שימוש בתרגום pid בעזרת פונ' hash למתאר התהליך – שלוקח הרבה זמן) המאקרו מבצע עיגול של esp כלפי מטה לכפולה של 8KB ואז מגיע לתחילת מתאר התהליך (מאפס את 13 הביטים התחתונים). לכן כאשר מבצעים קריאת מערכת שדורשת גישה למתאר התהליך המערכת החדשה מתבצעת גישה לזכרון שמתאר התהליך לא נמצא בו – דבר העלול לגרום לקריסת המערכת.

1. (17 נק')

בשאלה זו נדון בקשרי המשפחה כפי שבאים לידי ביטוי בשדות התהליך (p\_(o)pptr, p\_ysptr,…) ונלמדו בתרגולים:

1. (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");

///(continue)

.

.

for(int i=4;i<6;i++){

forkId=fork();

if(forkId==0){

printf("P%d",i);

return 0;

}

}

return 0;

}

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

**ציירו כאן את תשובתכם:**

P0

P5

P4

P1

P3

P2

p\_osptr

p\_pptr

p\_ysptr

p\_cptr

1. (7 נק') תנו דוגמה לקריאת מערכת שנלמדה בתרגול, שבה משתמשים בקשרי המשפחה כדי לבצעה? הסבירו איך בא לידי ביטוי השימוש בקשרי המשפחה בה:

**תשובה**

בקריאת המערכת wait – משתמשים בקשרי המשפחה , מכיוון שרוצים לבדוק ולהחזיר איזה מהבנים של תהליך האב הסתיים , לכן נשתמש בקשרי המשפחה כדי לעבור על כל תהליכי הבן לפי סדר יצירתם ( כדי לא לפספס אף בן ) עד שנגיע לבן הראשון שפעולתו הסתיימה – ונחזיר את pid שלו כערך חזרה של הפונקציה .

**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 הוא debugger. כלי זה מאפשר לבצע מעבר בין מסגרות פונקציות. בגישה לפונקציה מסוימת בעזרת הדיבגר, מתאפשרת לנו גישה למסגרת שלה בעזרת הרגיסטר ebp דבר המאפשר גישה לרגיסטרים השמורים, הפרמטרים שנשלחו אל הפונקציה.

