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 שומרת את הערכים הרלוונטים ברגיסטרים(register file) לקראת הקריאה לפסיקה int 0x80. |
| פקודת int 0x80 | קוד גרעין/  **קוד משתמש**/  חומרה | פונקציית המעטפת יוזמת פסיקת מעבד(128), אשר גורמת להחלפת מחסניות וקפיצה לשגרת הטיפות בפסיקה. |
| מציאת פונקציית הטיפול בפסיקה | קוד גרעין/  קוד משתמש/  **חומרה** | גישה לכתובת הזיכרון שבה שמורה כתובת שגרת הטיפול בפסיקה. |
| שמירת הרגיסטרים ss,esp,eflags,cs,eip במחסנית החדשה לאחר המעבר ל-kernel mode | **קוד גרעין/**  קוד משתמש/  חומרה | המעבד בעקבות פסיקה יזומה(או פסיקת חומרה) שומר את רגיסטרי המצב כדי לאפשר דריסתם ע"י שגרות אחרות, וכדי לאפשר שיחזור של המצב הקודם. |
| שמירת orig\_eax במחסנית, אשר מייצג את מספר השירות המבוקש, וקריאה ל-SAVE\_ALL | **קוד גרעין**/  קוד משתמש/  חומרה | פונקציית הטיפול בפסיקה 128 מבצעת "שימור סביבה" לקראת קריאה לפונקציית השירות הרצויה |
| 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 | בעזרת הערך ברגיסטר eax שפונקציית המעטפת הכינה מראש קוד הגרעין יודע למצוא את מספר השירות הרלוונטי (או להחזיר שגיאה במקרה שמספר השירות חורג מהקבוע NR\_syscalles) |
| **שלב** | **מבוצע על-ידי (הקף בעיגול)** | **הסבר** |
| קריאה לפונקציית השירות | **קוד גרעין/**  קוד משתמש/  חומרה | מספר השירות שמצאנו מהווה את האינדקס של המצביע לפונקציית ביצוע השירות הרלוונטית בטבלה  sys\_call\_table |
| ביצוע שגרת השירות | **קוד גרעין/**  קוד משתמש/  חומרה | בתחילת ביצוע השגרה נבדקים הארגומנטים לשגרה, ולאחר מכן מתבצע קוד שגרת השירות. |
| בהנחה והתרחשה שגיאה בקריאת המערכת, כתיבת קוד השגיאה לerrno- | קוד גרעין/  **קוד משתמש**/  חומרה | במידה והתבצעה שגיאה פונקציית המעטפת מציבה את הערך המוחלט של הערך המוחזר לתוך errno ומחזירה 1-. |
| החזרת ערך קריאת המערכת למשתמש | **קוד גרעין/**  קוד משתמש/  חומרה | ערך החזרה של פונקציית השירות מוחזר ע"י קריאת המערכת(קוד גרעין) ברגיסטר eax. |

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.**בקבלת פסיקה נשמרים רגיסטרי המצב (ואיתם ebx), לאחר מכן מתבצעת שמירת הרגיסטרים ע"י המאקרו SAVE\_ALL (ושוב נשמר אותו הערך של ebx).בשלב השחזור נשחזר את הרגיסטרים(הרלוונטים) ואיתם את ebx ע"י RESTORE\_ALL , וללא ביצוע פעולה נוספת על ebx, כשיופעל iret, ישוחזר אותו ערך(שכבר נמצא בebx) לebx,ולכן אין פגיעה במימוש.

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:

**המימוש אינו תקין.**החלפת המחסניות הכרחית ע"מ להגן על המערכת מקוד משתמש זדוני (הגרעין עובד ב"עולם משלו", והמשתמש רק מעביר בקשה ל"איזה שירות הוא צריך"), אם הגרעין יעבוד על אותה מחסנית, משתמש חכם יוכל לגרום למערכת לקפוץ לקוד משתמש שיבוצע בהרשאות מלאות ובכך לפגוע במערכת ובהתקנים חיצוניים.

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?

**המימוש אינו תקין.**במימוש המוצע, לא נוכל לשחזר את דגלי המצב איתם יזמנו את פסיקה 128.  
בנוסף בחזרה משגרת הטיפול בפסיקה נשאר עם דגלי המצב של שגרת הטיפול בפסיקה, ובפרט ברמת הרשאה גבוהה, מה שיאפשר לקוד המשתמש לפגוע במערכת (כי כעת יש לו הרשאה לעשות הכל).

**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.  
   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
3. (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:

int main(){

int forkId=fork();

if(forkId==0){//son

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

}else{//father

**Wait();**

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

}

return 0;

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:**

**גישה למתאר התהליך מתבצעת בלינוקס באמצעות חישוב כתובות(פעולות אריתמטיות על מיקומים בזיכרון).**

**בפועל כשרוצים בלינוקס ל"שלוף" את כתובת מתאר התהליך כאשר אנו נמצאים בקוד גרעין, מספיק לאפס את 13 הביטים התחתונים של esp כדי לקבלו.  
השינויים שנינוקס עשו גרמו לתהליכים לאבד את "מפת הדרך" שלהם, בעזרתה יכלו לגשת למתאר התהליך שלהם, כך שבפועל מערכת ההפעלה לא מצליחה למצוא את מתאר התהליך, כאשר מתבקשת לכך**, **ואפילו יכולה לשנות את הזיכרון בצורה שלא התכוונה אליו, ובכך לדרוס זיכרון של תהליכים אחרים.**

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:**

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**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:**

**Valgrind:**

**כשזיכרון מוקצה (באמצעות new או malloc) בעזרת רשימה זו ניתן לעקוב(ולשמור) אחר סדר הקריאות לפונקציות כך שבמידה וזיכרון זה לא ישוחרר, valgrind ייתן למתכנת מידע איזה פוקנציה הקצתה זיכרון זה, ומהי מחסנית הקריאות בעת הקצאת הזיכרון.  
בקצרה וכנראה לא במדויק: בעת הקצאת זיכרון, valgrind שומרת מידע שהתבצעה הקצאת זיכרון, וכמו כן את מצב מחסנית הקריאות(איזה פונקציה קראה לאיזה לאיזה פונקציה שקראה לפונקציות אחריה).  
כאשר זיכרון משוחרר בתוכנית, valgrind מעדכנת זאת ברשומות שהיא מחזיקה (כמה זיכרון הוקצה, כמה שוחרר וכדו').  
בסוף ריצת התוכנית, בעזרת הרשומות על זיכרון שאיננו נמחק, valgrind מספקת למתכנת את המידע(כמה זיכרון הוקצה בעת ההקצאה, מהי מחסנית הקריאות וכד') לטובת דיבוג יעיל.**

