Homework 1 Dry

**Due Date: 20/11/2018 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, please fill out the following form: <https://docs.google.com/forms/d/e/1FAIpQLSftN-3vAFuM3pmzHjoWb-QokiROxWZG4q75osCe6ItVnwXLig/viewform>

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. Only PDF format will be accepted.
4. You do not need to submit anything in the course cell.
5. 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')**

* **את שאלה זו יש לפתור לאחר שפתרתם את החלק הרטוב.**

1. (10 נק') מהו סדר הפונקציות/רוטינות בביצוע קריאת המערכת open?  
   1. sys\_open → system\_call → libc wrapper
   2. sys\_open →libc wrapper → system\_call
   3. system\_call → sys\_open → libc wrapper
   4. system\_call → libc wrapper→ sys\_open
   5. **libc wrapper→ system\_call → sys\_open**
   6. libc wrapper→ sys\_open → system\_call

פרטו את צורת התקשורת בכל קריאה לפונקציה/רוטינה (העברת פרמטרים והחזרת תוצאות חישוב):

*הקריאה מlibc wrapper לקריאת המערכת תתבצע באמצעות השמת הערכים אל תוך רגיסטרים (מפני שתתבצע החלפת מחסניות), החזרת הערכים תתבצע גם כן באמצעות רגיסטרים. הקריאה מקריאת מערכת ההפעלה לsys\_open תתבצע באמצעות דחיפת הערכים אל תוך המחסנית וערך ההחזרה מsys\_open יהיה באמצעות אוגר (eax).*

תזכורת: בחלק הרטוב הגדרנו רמות הרשאה לתהליכים וכן ספי הרשאה למספר קריאות מערכת "מאובטחות": fork, wait, waitpid, sched\_yield. במערכת של תרגיל הבית, תהליך בעל רמת הרשאה x יוכל לבצע קריאת מערכת בעלת סף הרשאה y אם ורק אם .

נרצה להרחיב את מנגנון האבטחה כך שניתן יהיה לעדכן את רשימת קריאות המערכת המאובטחות ואת ספי ההרשאה שלהן ללא צורך בהידור מחדש של הגרעין. לשם כך, ניצור קובץ קונפיגורציה במיקום קבוע וידוע למערכת ההפעלה אשר מכיל רשימה של קריאות מערכת וסף ההרשאה המתאים לכל אחת מהן. המשתמש יכול לערוך את קובץ הקונפיגורציה באופן דינמי וכך לעדכן את מנגנון האבטחה ללא צורך בקידוד והידור מחדש של הגרעין.  
שימו לב: מנגנון האבטחה נדרש להיות מעודכן לכל הפחות לזמן הפעלת המערכת האחרונה. כלומר, עריכת הקובץ בזמן ריצת המערכת לא בהכרח משפיעה מיד על מנגנון האבטחה.

לשם מטרה נעלה זו התגייסו כוכבי "בית הנייר" והציעו מימושים שונים להרחבה.

ההצעה של טוקיו: רמת ההרשאה של תהליך תשמר במשתנה גלובלי בגרעין מטיפוס integer. כמו כן, בתחילת כל פונקצית שירות (sys\_X) תמקם קריאה לפונקציית גרעין בשם checkSec עם מספר קריאת המערכת. checkSec תקרא את קובץ הקונפיגורציה ותבדוק האם התהליך בעל הרשאה לבצע את קריאת המערכת.

1. (8 נק') האם המימוש של טוקיו תקין? אם לא, מה הבעיה במימוש זה?

*משתנה גלובלי מעצם היותי גלובאלי, משותף לכולם. במידה שכל התהליכים ישתפו את אות המשתנה הגלובאלי, ברגע שניצור תהליכים חדשים, נשנה עימם את רמת הרשאתם, כלומר המשתנה הגלובלי ישתנה לכל שאר התהליכים אשר כבר נוצרו, חלקם עם רמת הרשאה נמוכה או גבוהה יותר, כלומר אופן הפעולה שלהם יפגע.*

ההצעה של ריו: רמת הרשאה של תהליך תשמר כשדה במתאר התהליך (PCB). כמו כן, ריו ימקם קריאה בתחילת system\_call לפונקציית גרעין בשם checkSec עם מספר קריאת המערכת. הפונקציה checkSec תקרא את קובץ הקונפיגורציה ותבדוק האם התהליך בעל הרשאה לבצע את קריאת המערכת.

1. (8 נק') האם המימוש של ריו תקין? אם לא, מה הבעיה במימוש זה?

*ההצעה של ריו תקינה כל עוד אנחנו לא פוגעים חיצונית בקובץ הקונפיגורציה, שכן כל תהליך מבודד מתהליך אחר ולכן רמות ההרשאות של תהליך מבודדות. מימשו רעיון זה באופן דומה בתרגיל בית 1.*

ההצעה של הפרופסור: רמת הרשאה של תהליך תשמר כשדה במתאר התהליך (PCB). כמו כן, הפרופסור יתחזק רשימה גלובלית בגרעין אשר מכילה מספרי קריאות מערכת וספי הרשאה המתאימים להן. את רשימה זו יאתחל רק בעלייה של המערכת על ידי קריאת קובץ הקונפיגורציה. בנוסף, ימקם קריאה בתחילת system\_call לפונקציית גרעין בשם checkSec עם מספר קריאת המערכת. הפונקציה checkSec תבדוק האם התהליך בעל הרשאה לבצע את קריאת המערכת על ידי מעבר על הרשימה הגלובלית.

1. (8 נק') האם המימוש של הפרופסור תקין? אם לא, מה הבעיה במימוש זה?

*המימוש של הפרופסור תקין שכן הוא לוקח את המימוש של ריו ואף משפר אותו.*

1. (8 נק') מה היתרון של ההצעה למימוש של הפרופסור על פני המימוש של ריו?

אם קובץ הקונפיגורציה נמחק, כל קריאות המערכת לא יוכלו להתבצע. מכיוון שבאתחול המערכת הרשימה מאותחלת בזיכרון עד לתום פעילות המערכת, לא ניתן לשנות דינאמית את הרשאות מערכת ההפעלה בזמן ריצתה וגם לא ניתן למחוק אותה בשגגה, אלא ע"י פעולה מכוונת של המתכנת. כמו כן, קריאה מזיכרון RAM מהירה יותר מקריאת מידע מקובץ (קריאה מדיסק).

1. (8 נק') מה היתרון של ההצעה למימוש של ריו על פני המימוש של הפרופסור?

*צורך פחות זיכרון RAM על המחשב ודינאמי, שכן אין צורך בהעלאת המערכת כדי לשנות את ההרשאות.*

**שאלה 2 (50 נק')**

1. (18 נק')
2. (10 נק') ניר, ששונא לחכות, וגם מאמין במשפט "מה ששנוא עליך אל תעשה לתהליכך", החליט שבכל תוכניות המחשב 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.

מצויין בהערה כי בתוכניות שכתב, התהליך לא רק זמן רב, לכן לפי ההנחה הוא סיים לרוץ טרם הסתיים לרוץ אחד הבנים עליהם לא נעשה Wait(), לכן Init יקח את התפקיד על הילד ולכן יבצע עליו Wait() מה שיגרום לשחרור הזכרון שלו, כלומר את הזיכרון של הpid שלו ואת סטטוס היציאה. כל שאר הזיכרון משוחרר כבר מהיותו של התהליך "זומבי".

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:

*נוסיף את הפקודה Wait(NULL) לאחר הFork על מנת לגרום לילד לרוץ החל מהשורה שאחרי הFork ולהדפיס ראשון את המחרוזת. לאחר סיום תהליך הבן, האבא יוכל לשוב לרוץ ולהדפיס את המחרוזת השנייה.*

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.  
*לא ברור מן השאלה אז נפריד לשני מקרים:  
יכול להיות שהPCB שלנו יימצא באיזור הuserspace וזה כמובן חריגה מהאופן שבו מערכת ההפעלה מהונדסת, בה אין גישה למשתמש באופן ישיר לכל התהליכים במערכת (למשל, אין לו הרשאות)  
מקרה שני: הPCB יושב מעל המחסנית הגרעין, או בOFFSET קבוע ממנה. אם נגדיל את גודל הPCB מבלי לעדכן את הOFFSETים במערכת, במהלך CONTEXT SWITCH לדוגמה, או יותר נכון במהלך הרצת המאקרו current, כאשר אנחנו ניגשים לPCB, ניגש לאזורים אחרים בזיכרון, אשר יכולים להיות ערכי זבל ולגרום בכך לקריסת התוכנית, שכן הזיכרון ימופה בצורה שונה מk8.*

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:

**///(continue)**

**.**

**.**

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

**forkId=fork();**

**if(forkId==0){**

**printf("P%d",i);**

**return 0;**

**}**

**}**

**return 0;**

**}**

**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){**

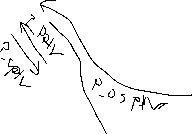
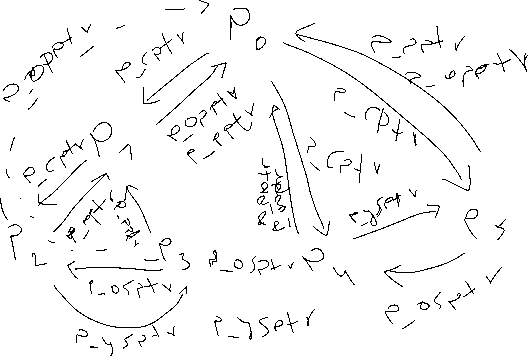
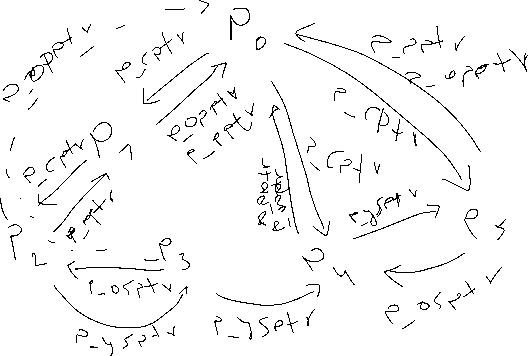
**printf("P3");**

**}**

**return 0;**

**}//more code on the right**

**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:

*קריאה לפונקציה getppid() תבצע לולאה שבה תשתמש בp\_opptr כדי לבדוק מי הוא האבא הקדמון של התהליך הנוכחי. היא תבצע זאת עד שתגיע לשורש העץ, כלומר לא יהיה יותר אבא שונה. Snippet של הפונקציה:*

*asmlinkage long sys\_getppid(void)*

*{*

*int pid;*

*struct task\_struct \* me = current;*

*struct task\_struct \* parent;*

*parent = me->p\_opptr;*

*for (;;) {*

*pid = parent->pid;*

*#if CONFIG\_SMP*

*{*

*struct task\_struct \*old = parent;*

*mb();*

*parent = me->p\_opptr;*

*if (old != parent)*

*continue;*

*}*

*#endif*

*break;*

*}*

*return pid;*

*}*