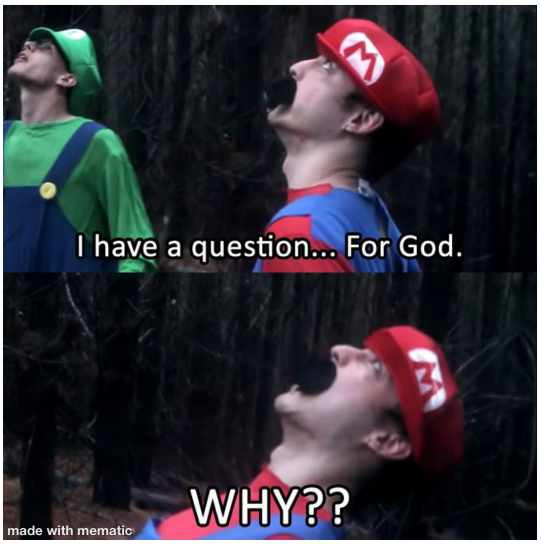
מערכות הפעלה - תרגיל בית יבש 1

Ido Tausi, *214008997,* ido.tausi@campus.technion.ac.il
Noam Bitton, *213745953*, bitton.noam@campus.technion.ac.il

כשדחית את הטיפול בג׳ובים בטיימאאוט,
ועכשיו זה עושה מלא באגים:

אני והשותפה אחרי מטלה 1 בהפעלה:



שאלה 1:

1. הסבירו: מהי אבסטרקציה במערכות מחשבים?

1. אבסטרקציה של מערכות הפעלה היא הפשטה והנגשה למשתמש של משאבי המחשב. האבסטרקציה מסתירה את המורכבות של מערכת ההפעלה וגם את החומרה והתוכנה ב-low level, מהמשתמש ומהתוכנות שרצות על המחשב. באמצעות שימוש באבסטרקציה, מערכת ההפעלה משמשת כמתווכת בין החומרה לבין האפליקציות שרצות על המחשב (ב-high level).

2. מדוע אנו משתמשים באבסטרקציות במערכות הפעלה?

2. אנו משתמשים באבסטרקציות במערכות הפעלה כדי לפשט למשתמשים את המערכת ולהקל עליהם בשימוש במערכת וביצירת אפליקציות שירוצו על מערכת ההפעלה. כלומר, המשתמש יוכל לנצל את יכולות המערכת ללא צורך בהבנת הפרטים הבסיסיים ביותר של אופן פעולתה. בצורה זו ניתן לפתח אפליקציות בצורה נוחה יותר ולהנגיש את הסביבה ליותר ויותר משתמשים.

3. תנו דוגמה לאבסטרקציה מרכזית שמשתמשים בה במערכות הפעלה והסבירו מדוע משתמשים בה.

3. דוגמה לאבסטרקציה מרכזית במערכות הפעלה היא וירטואליזציה של זיכרון. מערכת ההפעלה מעניקה לכל תהליך (process) את האשליה שיש לו מרחב זיכרון נפרד משלו, בו הוא יכול לרוץ ולשמור דברים ללא חשש שתהליכים אחרים ייגשו אליו וישנו אותם. בנוסף, בצורה זו תהליכים אחרים לא יכולים לקרוא מזיכרון של תהליך. בצורה זו, הוירטואליזציה של הזכרון מגנה על התהליך.
כמו כן, הודות לאבסטרקציה זו מערכת ההפעלה גם מקלה על התהליך בכך שהתהליך לא צריך לבדוק ו"לחשוב" לאיזה זכרון מותר לו לגשת שכן כל הזכרון הוירטואלי שלו. מערכת ההפעלה ממפה מאחורי הקלעים בין כתובות וירטואליות לכתובות הפיזיות שמוקצות לתהליך.

שאלה 2

חלק 1:

סעיף 1:

השלימו את קוד ה-C הנתון הממשמש shell כך שיבצע את פקודת ה-bash:

/bin/prog.out < in.txt 2> err.txt > out.txt

```
close(0)
int inputFd = open ("in.txt",...);
close(1);
int outputFd = open ("out.txt",...);
close(2);
int errorFd = open ("err.txt",...);

char* args[] = {"/bin/prog.out", NULL};
execv(args[0], args);
```

סעיף 2:

אם היינו מחליפים את שורות 10 ו-11, ערכי המשתנה count בשני התהליכים -

הקף: בהכרח זהים \ בהכרח שונים \ ייתכן ששונים וייתכן שזהים

1. אם היינו מחליפים את שורות 10 ו-11, ערכי המשתנה count בשני התהליכים - בהכרח זהים.

אם **לא** נחליף את שורות 10 ו-11, ערכי המשתנה count בשני התהליכים –

הקף: בהכרח זהים \ בהכרח שונים \ ייתכן ששונים וייתכן שזהים

2. אם לא נחליף את שורות 10 ו -11, ערכי המשתנה count בשני התהליכים - ייתכן ששונים וייתכן שזהים.

האם הקוד תקין? (5 נקודות)

הקיפו: כן \ לא

אם בחרתם לא, תנו דוגמא לתרחיש הבעייתי ביותר האפשרי:

3. הקוד לא תקין. התרחיש הבעייתי ביותר האפשרי:

במקרה שבו ה-*counter* של האבא קטן מזה של הבן, נצפה שהאבא והבן יפסיקו לקרוא ולכתוב כאשר לולאת ה- for של האבא יסתיים אך מאחר והבן לא סוגר את ערוץ הקלט ב-*pipe*, ניתקל בבעיה הבאה:

- במקרה הטוב, זה יבזבז זמן עד שגם לולאת ה-*for* עם ה-counter של הבן תסתיים.
- במקרה הרע, הבן ימלא את ה-*pipe* לפני שלולאת ה-*for* תיגמר ואז הוא יתקע.

חלק 2:

1. בחרו באפשרות הנכונה בנוגע לריצת הקוד: (4 נקודות)

1. יודפס קודם "Hi" ואז "Hello".
2. יודפס רק "Hello".
3. יודפס רק "Hi".
4. לא יודפס כלום.
5. תשובות i,ii אפשריות.

1. פקודת *signal* קובעת את ההנדלר של הסיגנל המתאים לשגיאות אריתמטיות להיות הדפסת המילה "*Hello*" ואז exit(0) כלומר התכנית מסתיימת בסוף ההנדלר.
בשורה $int\ x\ =\ 234123\ /\ (0);$ מתקבלת שגיאה אריתמטרית (חלוקה -0) ולכן יישלח הסינגל SIGFPE ותיקרא הפונקציה שהגדרנו שתדפיס את המילה "Hello" ולאחר מכן תצא מהתכנית ולא תגיע לשורות שלאחר מכן.
לפיכך, התכנית תדפיס רק את המילה "Hello".

- יודפס "Hello" פעמיים.

הקיפו: כן \ לא

2. 1) **לא ייתכן.**
אם תהליך מקבל שני סיגנלים, הוא לא יטפל בהם בו זמנית, אלא בכל אחד מהם בנפרד, ולכן לא ייתכן שההנדלר ייקרא פעמיים וידפיס "Hello" פעמיים, מפני שלאחר שהוא ייכנס להנדלר בפעם הראשונה הוא יגיע ל-exit(0) והתהליך ייפסק. לכן לא ייתכן שיודפס "Hello" פעמיים.

- לא יודפס "Hello" בכלל.

הקיפו: כן \ לא

2) **ייתכן.**
מצב זה ייתכן כאשר הסינגל SIGFPE יישלח מהתהליך האחר לפני שהשורה signal(SIGFPE, fpe_catcher) תתבצע. במצב כזה יישלח הסיגנל לפני שההנדלר החדש אשר מדפיס Hello הוגדר להיות ההנדלר של סיגנל זה, ולכן ההנדלר המתאים יהיה ההנדלר הדיפולטיבי אשר הורג את התהליך. כתוצאה מכך יודפס Hello אפס פעמים כי התהליך של ההנדלר רק יסיים את התכנית מבלי להדפיס כלום.

3. תארו את ריצת התכנית במידה והיינו מסירים את פקודת ה-exit(0):

3) במידה והיינו מסירים את פקודת exit(0) אז בכל פעם שהתכנית מקבלת את סיגנל SIGFPE, היא מפעילה את ההנדלר של סיגנל זה שהוגדר בהתחלה, ויודפס $Hello$ על המסך. בנוסף, מפני שאין קריאה ל-$exit(0)$ אז התוכנית לא תסיים לאחר השורה שבה יש חלוקה ב-0, ותחזור לשורה ממנה הסיגנל נשלח, ומאחר וההשגיאה לא תוקנה ועדיין ישנה חלוקה ב-0, אז שוב יישלח סיגנל שייקרא להנדלר וידפיס Hello ואז יחזור לשורה ממנה הסיגנל נשלח ושוב יישלח סיגנל וכן הלאה.
כלומר, מאחר וההנדלר לא מתקן את השגיאה וגם לא מסיים את התהליך, אז נקבל אינסוף חזרות של שליחה של הסיגנל והדפסת Hello בהנדלר ולכן, בריצת התוכנית יודפס $Hello$ אינסוף פעמים.

שאלה 3:

```
int X = 1, p1 = 0, p2 = 0;
int ProcessA()
{
   printf("process A\n");
   while(X);
   printf("process A finished\n");
   exit (1);
}

void killAll(){
 if(p2) kill(p2, 15);
 if(p1) kill(p1, 9);
}

int ProcessB()
{
  X = 0;
  printf("process B\n");
  killAll();
  printf("process B finished\n");
  return 1;
}

int main(){
 int status;
 if((p1 = fork()) != 0)
 {
   if((p2 = fork()) != 0)
   {
     wait(&status);
     printf("status: %d\n", status);
     wait(&status);
     printf("status: %d\n", status);
   }
   else
   {
     ProcessB();
   }
 }
 else
 {
```

```
    ProcessA();
  }
  printf("The end\n");
  return 3;
}
```

1. ייתכן 0 פעמים וייתכן פעם 1.

בכל שורת if, לתהליך הקיים נוצר תהליך בן, והוא נכנס לתוך בלוק ה-if, לכן נוצרים שני תהליכים בנים שהם גם אחים, כלומר בעלי אב משותף. התהליכים הבנים מבצעים את הקוד שבבלוק ה-else שמתאים ל-if שממנו נוצרו, ולכן כמות ההדפסות תלוייה באיזה תהליך יגיע ראשון לפונקציה שהוא מריץ (processA/processB) בבלוק ה-else שלו.

- אם processA ייקרא ראשון, תודפס המילה "process A". בנוסף, בתחילת התוכנית, הוגדר X=1 והוא יישאר עם ערך זה ולא ישתנה בתוך הפונקציה processB כי שתי הפונקציות האלו יקראו דרך שני תהליכים שונים ולכן יש להם מרחב זיכרון שונה ועל כן שינוי של X בתהליך אחד לא ישפיע עליו בפונקצייה השניה. לכן, לאחר הדפסת "process A" התהליך ייכנס ללולאה אינסופית ולא יגיע לשורה שמדפיסה את "process A finished". בנוסף, זוהי הקריאה היחידה לפונקצייה processA ולכן השורה הנ"ל תודפס רק פעם אחת.
- אם processB ייקרא ראשון, תיתבצע הפונקצייה killAll לפני שתתבצע הפונקצייה processA ובפרט יישלח ל-p1 הסיגנל SIGKILL שתסיים את התהליך p1 ואז הפונקצייה processA לא תקרא באף שלב ולכן השורה "process A" לא תודפס כפלט בכלל.

2. שורה זו תודפס 0 פעמים.

כפי שהסברנו בסעיף הקודם, לכל אורך ריצת התכנית יתקיים בפונקצייה processA כי X=1 ולכן התוכנית תיכנס ללולאה אינסופית, ולכן לעולם לא תגיע לשורה exit(1) ותסיים, ופעולת ה-wait לא תחזיר סטטוס 1 מתהליך זה, אלא סטטוס שמעיד על סיום התהליך בעקבות SIGKILL, כלומר 137. בנוסף, בפונקצייה processB אנו נשלח SIGKILL לתהליך האח שלו, ואז נחזור לביצוע המשך הקוד ב-main ונסיים עם השורה האחרונה שמחזירה 3, ולכן נקבל באחת מקריאות ה-wait, את הסטטוס 3 בחזרה, שמסמל את הסיום המוצלח של תהליך p2. לכן לא יודפס "status: 1" בשום שלב.

3. שורה זו תודפס רק פעם אחת.

כאשר processB ייקרא מתוך תהליך הבן השני (כאשר p2=0), יישלח הסיגנל SIGKILL לתהליך הבן הראשון (כאשר P1=0), כפי שפירטנו בסעיפים קודמים, ולכן נקבל בסטטוס כערך יציאה את המספר 137 שמשמעותו היא שהתהליך הסתיים עם סיגנל שמספרו 9, כלומר SIGKILL, ולכן שורה זו תודפס פעם אחת בדיוק, מפני שהתהליך p2 מסתיים עם ערך חזרה 3.

4. שורה זו תודפס 0 פעמים.

כאשר הפונקציה killAll נקראת מתוך הפונקציה processB תמיד מתקיים ש- p1!=0 וגם p2=0.
במצב כזה, מאחר ו- p2=0 אז תנאי ה-if לא מתקיים, והשורה kill(p2, 15) לא תתבצע ולכן לא ייתכן שהתכנית תסתיים עם הסיגנל 15 ולכן בהכרח גם לא תסתיים עם ערך היציאה 143. נשים לב כי ערך הסיום (שלא כתוצאה מסיגנל) האפשרי היחיד בתוכנית זו הוא 3, ולכן שורה זו מודפסת בהכרח 0 פעמים.

5. שורה זו תודפס פעמיים.

כפי שתיארנו בסעיפים קודמים, תהליך הבן השני מגיע לסוף ה-main ומסיים, ולכן מדפיס "The end". בנוסף אליו, גם תהליך האבא מגיע לסוף ה-main ומדפיס "The end". תהליך הבן הראשון מסתיים כתוצאה מ-SIGKILL, בשלב שבו הוא תקוע בלולאה אינסופית ב-processA. לכן, השורה תודפס פעמיים בלבד.