מערכות הפעלה – 234123

**תרגיל בית 1 - יבש**

**אביב 2023**

**מגישים:**

**נועה פריאנטה 206200305**

[pariente@campus.technion.ac.il](mailto:pariente@campus.technion.ac.il)

**ששון שמואל למעי 325172351**

sason-shmuel@campus.technion.ac.il

# Question 1 – Abstraction & OSs (15 points)

1. הסבירו: מהי אבסטרקציה במערכות מחשבים?

אבסטרקציה במערכות מחשבים זו הפעולה של הסתרת פרטי מימוש של פונקציות או מודלים מסוימים, כך שבסופו של דבר יש לנו רק את המידע על מה הפונקציה אמורה לעשות אך לא איך היא עושה זאת.

1. מדוע אנו משתמשים באבסטרקציות במערכות הפעלה?

אנו משתמשים באבסטרקציות במערכות הפעלה ממספר סיבות, ראשית, בכך אנו יוצרים שכבת הגנה על הפונקציה, כי לא יודעים את המימוש שלה ובפרטלא את המשתנים, משאבים וכו' שהיא משתמשת בהם. שנית, זה מקל על פיתוח של אפליקציות בכך שהמשתמש לא צריך להיות מוסח ממימוש שלא אמור להשפיע עליו.

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

דוגמה אחת מרכזית לשימוש באבסטרקציה: וירטואליזציה של משאבים פיזיים (מעבד וזיכרון) עבור המשתמש. מערכת ההפעלה מציגה למשתמש גרסאות וירטואליות של המעבד והזיכרון שמספקים אבסטרקציה לחומרה הפיזית ובכך הופכים את השימוש לפשוט. מלבד זאת ככה אנו מגנים על המשאב הפיזי כי אף משתמש לא יידע היכן המידע באמת קיים.

# Question 2 – Inter-Process Communication (45 points)

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

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

עליכם להשתמש אך ורק בקריאות המערכת הבאות:

int **open**(const char \*path, …);

int **execv**(const char \*filename,

char \*const argv[]);

int **close**(int fd);

* אין עליכם חובה לבדוק שקריאות המערכת צלחו.
* פתחו קבצים באמצעות הקריאה open(path,…), כלומר בפרמטר השני יש לרשום "..." (גם לקבצים קיימים וגם לחדשים).
* אין לבצע קריאות מערכת מיותרות.
* תזכורת:

"2> err.txt" מציין redirection של fd=2 (STDERR) של התהליך לכתיבה לקובץ err.txt.

pid\_t pid **=** fork**();**

**if** **(**pid **==** 0**)** **{**

close(0) ;

close(1) ;

close(2) ;

open("in.txt", "...");

open("out.txt", "...");

open("err.txt", "...");

chr\* args[] = {"/bin/prog.out", NULL};

execv(args[0], args);

**}** **else** **{**

   wait**(NULL);**

**}**

1. (7 נקודות)

לפניכם קטע קוד המשתמש ב-pipes.

|  |  |
| --- | --- |
| **1** | #include <stdlib.h> |
| **2** | #include <unistd.h> |
| **3** | #include <stdio.h> |
| **4** |  |
| **5** | int main**()** **{** |
| **6** | int fd**[**2**];** |
| **7** | int count**;** |
| **8** | int id**;** |
| **9** | pipe**(**fd**);** |
| **10** | pid\_t p **=** fork**();** |
| **11** | count **=** get\_random\_between**(**1**,** 1000000**);//gets random integer in [1,1000000]** |
| **12** | **if** **(**p **==** 0**)** **{** |
| **13** | id **=** 234123**;** |
| **14** | **for(**int i **=** 0**;** i **<** count**;** i**++)** |
| **15** | **{** |
| **16** | write**(**fd**[**1**],** **(**void**\*)(&**id**),** **sizeof(**int**));** |
| **17** | **}** |
| **18** | **}** **else** **{** |
| **19** | close**(**fd**[**1**]);** |
| **20** | **for(**int i **=** 0**;** i **<** count**;** i**++)** |
| **21** | **{** |
| **22** | **if(**read**(**fd**[**0**],** **(**void**\*)(&**id**),** **sizeof(**int**))** **>** 0**)** |
| **23** | **{** |
| **24** | printf**(**"My favorite course, %d\n"**,** id**);** |
| **25** | **}** |
| **26** | **else** |
| **27** | **{** |
| **28** | **break;** |
| **29** | **}** |
| **30** | **}** |
| **31** | **}** |
| **32** | **return** 0**;** |
| **33** | **}** |

בשני הסעיפים הבאים, הקף את התשובה הנכונה: (1x2 = 2 נקודות)

אם היינו מחליפים את שורות 10 ו-11, ערכי המשתנה count בשני התהליכים -    
 הקף: בהכרח זהים \ בהכרח שונים \ ייתכן ששונים וייתכן שזהים

אם **לא** נחליף את שורות 10 ו-11, ערכי המשתנה count בשני התהליכים –   
הקף: בהכרח זהים \ בהכרח שונים \ ייתכן ששונים וייתכן שזהים

האם הקוד תקין? (5 נקודות)   
הקיפו: כן \ לא

אם בחרתם לא, תנו דוגמא לתרחיש הבעייתי ביותר האפשרי:   
תהליך הבן לא סגר את FD של הקריאה בpipe לכן, גם אם האבא שלו יסיים לקרוא הבן ימשיך לכתוב עד מגבלת count שהוא הגריל. מקרה הכי גרוע, נגריל את המספר מליון עבור הבן והמספר 1 עבור האב, הבן יכתוב ברצף מליון int'ים , דבר שגם משפיע על הביצועים שמרחשים לחינם וגם עלול לגרום לשגיאות אם אי אפשר כבר לכתוב כי אין מקום (במימוש של הpipe)

חלק 2: (10 נקודות)

לפניכם קטע קוד:

#include <stdio.h>

#include <signal.h>

#include <stdlib.h>

void fpe\_catcher **(**int signum**)** **{**

  printf**(**"Hello\n"**);**

  exit(0);

**}**

int main**()** **{**

  signal**(**SIGFPE**,** fpe\_catcher**);**

  int x = 234123 / (0);

  printf**(**"Hi\n"**);**

**while(**1**);**

  return 0**;**

**}**

תזכורת: SIGFPE הוא הסיגנל המתאים לשגיאות אריתמטיות, כמו חלוקה ב-0 (גם במקרה של מספרים שלמים).

1. בחרו באפשרות הנכונה בנוגע לריצת הקוד: (4 נקודות)
2. יודפס קודם "Hi" ואז “Hello”.
3. יודפס רק "Hello".
4. יודפס רק "Hi".
5. לא יודפס כלום.
6. תשובות i,ii אפשריות.

**נמקו:**

תיווצר החריגה בגלל החלוקה ב0, שגרת הטיפול תיקרא, היא תדפיס Hello ואז מבצעת קריאת המערכת exit() שעל פי הגדרתה מסיימת את ביצוע התהליך הקורא ומשחררת את כל המשאבים שברשותו.

1. כעת נניח שבזמן כלשהו של ריצת הקוד הנ"ל, תהליך אחר שולח את הסיגנל SIGFPE ל תהליך המריץ את הקוד.

עבור כל אחד מהתרחישים הבאים, הכריעו האם הוא **אפשרי** או לא, ונמקו בקצרה: (2x2=4 נקודות)

* יודפס “Hello” פעמיים.

הקיפו: כן \ לא

נימוק:

נתאר את התרחיש הבא: קטע הקוד הנתון רץ תחת תהליך שנקרא לו A. כפי שלמדנו בהרצאה במעבר בין מצב גרעין למשתמש נבדק האם לתהליך נשלחו סיגנלים. תהליך B שולח סיגנל SIGFPE לתהליך A אשר הוא במצב משתמש וקראה פסיקת שעון שגורמת לעבור למצב גרעין בפרק זמן המתאים בו קריאת exit() עוד לא נקראה אבל כן יודפס hello במסך, כך המעבד מפסיק לבצע את הפקודה הנוכחית (כלומר לא קורא ל-exit()) ועובר לבצע את שגרת הטיפול בפסיקה שתדפיס שוב hello ואז תקרא לפונקציה exit()

לא יודפס “Hello” בכלל.

הקיפו: כן \ לא

נימוק:

שגרת הטיפול בהכרח תתרחש, אם מהחלוקה באפס או מהסינגל מהתהליך. ובמהלך שגרת הטיפול מודפס hello.

1. תארו את ריצת התכנית במידה והיינו מסירים את פקודת ה-exit(0): (2 נקודות)

כפי שלמדנו בהרצאה נקודה החזרה של ההנדלר היא לאותה פקודה שהורצה ברגע תחילת הטיפול בסיגנל. לכן אם נסיר את exit() בסוף ההנדלר, התהליך יבצע את החלוקה באפס והסיגנל יקרא שוב ושוב ולכן יודפס אינסוף פעמים (על שהתהליך מת) hello\n"" כלומר hello בכל שורה.

# Question 3 – Process management (40 points)

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;

}

בשאלה זו עליכן להניח כי:

1. קריאות המערכת fork() וkill() אינן נכשלות.
2. כל שורה הנכתבת לפלט אינה נקטעת ע"י שורה אחרת.
3. כאשר תהליך מקבל סיגנל x הוא מסתיים וערך היציאה שלו הוא 128 + x.

עבור כל אחת משורות הפלט הבאות, סמנו כמה פעמים הן מופיעות בפלט כלשהו, נמקו את תשובתכן.

1. process A
   1. 0
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2

נימוק: יתכנו שני מקרים: פונקציה ProcessA() תקרא לפני לפני שבפונקציה ProcessB() קורא ל-killAll() ובכך יודפס על המסך "process A" או שיבוצע killAll() לפני שהקריאה לפונקציה ProcessA() מתרחשת ובקרה זה לא יודפס.

1. status: 1
   1. 0
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2

נימוק: נסתכל על שני המקרים הבאים: B מסתיים לפני A (כי פקודת killall רק שולחת סיגנל אבל אינה מחכה שהסיגנל יטופל) והערך החזרה של B הוא 1 ולכן יודפס status: 1. יתכן גם ש-B יסתיים לפני שנגיע ל-wait ולכן ערך ההחזרה יהיה -1 כי כבר סיים לרוץ ולכן לא יודפס status:1 למסך.

1. status: 137
   1. 0
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2

נימוק: סיגנל kill הינו בעל מספר 9. אם A הסתיים לפני B בגלל שליחת הסיגנל ב-B של kill יודפס status:137 שהרי 9+128=137 יתכן גם ש-B יסתיים לפני שנגיע ל-wait (מכאן שגם A) ולכן ערך ההחזרה יהיה -1 כי כבר סיים לרוץ ולכן לא יודפס status:137 למסך.

1. status: 143
   1. 0
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2

נימוק: killAll אינה שולחת סינגל לתהליך B מכיוון שהוא זה שקורא לפונקציה ולכן p2=0 ו-p1 שווה לpid של תהליך A ולכן לא יתכן שיודפס status: 143

1. The end
   1. 0
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2
2. נימוק: גם האבא הראשי וגם הבן של תהליך B יגיעו לשורת קוד המדפיסה The end ולכן יודפס פעמיים