Operating Systems – 234123

**Homework Exercise 1 – Dry**

**Winter 2023**

Teaching assistant in charge:

**Ori Ben-Zur**

Assignment Subjects & Relevant Course material

**Processes and inter-process communications**

**Recitations 1-3 & Lectures 1-3**

# Submission Format

1. Only **typed** submissions in **PDF** format will be accepted. Scanned handwritten submissions will not be graded.
2. The dry part submission must contain a single PDF file named with your student IDs –

**DHW1\_123456789\_300200100.pdf**

1. The submission should contain 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.
2. Submission is done electronically via the course website, in the **HW1 – Dry** submission box.

# Grading

1. **All** question answers must be supplied with a **full explanation**. Most of the weight of your grade sits on your **explanation** and **evident effort**, and not on the absolute correctness of your answer.
2. Remember – your goal is to communicate. Full credit will be given only to correct solutions which are **clearly** described. Convoluted and obtuse descriptions will receive low marks.

# Questions & Answers

* The Q&A for the exercise will take place at a public forum Piazza **only**. Please **DO NOT** send questions to the private email addresses of the TAs.
* 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 .

# Late Days

* Please **DO NOT** send postponement requests to the TA responsible for this assignment. Only the **TA in charge** can authorize postponements. In case you need a postponement, please fill out the attached form:

https://forms.office.com/r/X8SqhiTNwe

# Question 1 – Abstraction & OSs (15 points)

1. הסבירו: מהי אבסטרקציה במערכות מחשבים?  
   אבסטרקציה(=הפשטה) הינו ממשק הנותן למשתמש את הפריבלגיה להתעלם מפרטי מימוש, ולהשתמש בפונקציות/אובייקטים של האבסטרקציה על מנת להשיג את מטרותיו.  
   למשל, אם ארצה לממש מערך – אני יכול לעשות זאת בעצמי, או להשתמש ב-std::vector שמישהו אחר כתב, וסיפק לי את הממשק.
2. מדוע אנו משתמשים באבסטרקציות במערכות הפעלה?  
   על מנת לספק למשתמש ממשק נוח וקריא ברמה עלית, מאשר להצריך אותו לרדת לרמת האסמבלי והמימוש הספציפי בכל פעם שירצה לבצע פעולה ברמת החומרה (כמו כתיבה לדיסק).  
   בנוסף, מכיוון שהמימוש הוא ספציפי לחומרה עליה מערכת ההפעלה רצה – אבסטרקציות מאפשרות לנו להשתמש בממשק אחיד שעובד עבור כולם.  
   בנוסף, גם על מנת לשמור על המשתמש (מעצמו ומגורמים זדוניים), ולא לתת לו את היכולת לבצע פעולות שיכולות לסכן את המכשיר שלו (למשל לכתוב לכתובת זיכרון בלי לדעת לאיזה תהליך היא שייכת).
3. תנו דוגמה לאבסטרקציה מרכזית שמשתמשים בה במערכות הפעלה והסבירו מדוע משתמשים בה.  
   זיכרון וירטואלי. כפי שנאמר למעלה, משתמשים באבסטרקציה הזאת על מנת להגן על המשתמש. בכל פנייה לזיכרון, אנו פונים לכתובת שלא קיימת באמת (וירטואלית), ומערכת ההפעלה מפקיעה מאיתנו את השליטה ומחליטה האם לתת לנו גישה לכתובת הפיזית המקושרת לכתובת אליה פנינו. בכך אנו לא יכולים לדרוס את המשאבים של תהליך אחר.

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

נתונה התוכנית הבאה:

1 int main() {   
2 char s[] = "A";   
3 int fd = open("file", O\_RDWR|O\_CREAT);    
4 write(fd, "B", 1);   
5 close(fd);    
6 fd = open("file", O\_RDWR|O\_CREAT);    
7 close(STDIN\_FILENO);     
8 close(STDOUT\_FILENO);   
9 dup(fd);    
10 dup(fd);   
11 if (fork() == 0) {   
12 scanf("%s", s);    
13 printf("C");   
14 write(STDERR\_FILENO, s, 1);   
15 return 0;   
16 }   
17 s[0] = 'D';   
18 wait(NULL);   
19 printf("E");   
20 close(fd);   
21 return 0;   
22 }

הניחי שהקובץ file לא קיים לפני ריצת התכנית וכל קריאות המערכת מסתיימות בהצלחה. עבור כל סעיף מבין הסעיפים הבאים, צייני (i) מה יהיה פלט התכנית למסך (ii) ומה יהיה תוכן הקובץ "file". אם יש כמה אפשרויות לתוכן הקובץ או לפלט התכנית כתבי את כולן.

1. (3 נק') עבור התכנית כמו שהיא (ללא שינויים):

פלט התכנית (2 נק'): "B"                     תוכן הקובץ: (1 נק') "BCE"

נימוק: אנו מדפיסים B לקובץ בהתחלה, לפני סגירתו. לאחר מכן אנו פותחים שוב, והפוינטר לקריאה/כתיבה מאותחל לתחילת הקובץ.  
לאחר מכן אנו סוגרים את stdin ו-stdout, ושמים במקום את ה-fd של הקובץ שפתחנו.  
כעת, בתהליך הבן, אנו סורקים את הקובץ (כי scanf קורא מהמקום של stdin ב-FDT), ושמים ב-s את הקלט (התו B). שימו לב כי אנו מקדמים את פוינטר הקריאה/כתיבה.  
לאחר מכן אנו מדפיסים לקובץ (כי printf מדפיס למקום של stdout ב-FDT) את התו C. קידמנו את הפוינטר מקודם, לכן תו זה אינו דורס את התו הקודם.  
לבסוף, בתהליך הבן, אנו מדפיסים לפלט (stderr) את מה ש-s מחזיק בתהליך בן, כלומר את B.  
נשים לב כי אמנם האב משנה את s, אך זה נעשה לאחר ה-fork, ועל כן לא משפיע על הבן.  
אנו מדפיסים את התו E לקובץ בעזרת printf, ומסיימים.

1. (4 נק') במידה ונסיר את שורות 5+6.

פלט התכנית (2 נק'): "A"                 תוכן הקובץ (2 נק'): "BCE"

נימוק: כמעט זהה לסעיף 1, רק שכעת, מכיוון שלא נסגור ונפתח מחדש, הפוינטר לא יחזור לתחילת הקובץ. על כן ב-scanf נקרא מסוף הקובץ, ולא נצליח לקרוא שום דבר. אז s ימשיך להחזיק את A, וזה מה שיודפס ל-stderr.

1. (4 נק') במידה ונסיר את שורה 12. (הניחי שלא בוצע שום שינוי אחר ביחס לקוד המקורי.)

פלט התכנית (2 נק'): "A"                         תוכן הקובץ (2 נק'): "CE"

נימוק: תוכן הקובץ ישאר זהה למה שראינו בסעיף 1.  
מעבר לכך שאם נסיר את הסריקה, אז s ימשיך להחזיק את A, וזה מה שיודפס ל-stderr, יהיה גם שינוי בתוכן הקובץ. מכיוון שאנו לא מקדמים את הפוינטר של הקובץ בעזרת הסריקה, אז הוא ימשיך להצביע לתחילת הקובץ. משמע, כשנדפיס את התו C, זה ידרוס את התו B שהיה לפניו.

1. (4 נק') במידה ונסיר את שורה 15. (הניחי שלא בוצע שום שינוי אחר ביחס לקוד המקורי.)

פלט התכנית (2 נק'): "B"         תוכן הקובץ (2 נק'): "BCEE"

נימוק: פלט התוכנית ישאר זהה למה שראינו בסעיף 1.  
אך כעת תהליך הבן ימשיך עד לסוף של main, על כן גם הוא וגם האב ידפיסו E לקובץ, ולכן נקבל אקסטרה E בסוף הקובץ.

1. (5 נק') במידה ונסיר את שורה 18. (הניחי שלא בוצע שום שינוי אחר ביחס לקוד המקורי.)

פלט התכנית (2 נק'): "B" או "A" תוכן הקובץ (3 נק'): "BCE" או "EC"  
(תוכן הקובץ כתוב בהתאמה לפלט התכנית)

נימוק: מכיוון שהסרנו את ה-wait, הפלט ותוכן הקובץ תלויים בסדר הריצה של האב והבן.  
אם הבן רץ עד סופו, ורק אז האב רץ, אז נקבל מה שקיבלנו בסעיף 1.   
אחרת, האב ידרוס את ה-B שכתובה בקובץ, וישים במקומה E.  
שינוי ה-s לא ישתקף אצל הבן, לכן לא רלוונטי לנו.  
לאחר מכן, הבן ינסה לסרוק ולא יצליח, כי הוא בסוף הקובץ. אז s יחזיק את A.  
נדפיס C לקובץ, ולאחר מכן נדפיס ל-stderr את s.

1. (5 נק') במידה שנמחוק את שורות 16–11 ונוסיף ;()fork בין השורות 2 ו 3. (הניחי שלא בוצע שום שינוי אחר ביחס לקוד המקורי.)   
   פלט התכנית (3 נק'): אין               תוכן הקובץ (2 נק'): "E"

נימוק: אנו לא מדפיסים ל-stderr כאן, והרי stdout מצביע על הקובץ, לכן לא יהיה פלט.

באשר לתוכן הקובץ, נשים לב ראשית כי ה-CWD של שני התהליכים זהה, ונתנו להם אותו שם קובץ, לכן הם יצביעו על אותו קובץ, אם כי ה-seek ptr שלהם יהיו עצמאיים זה מזה.  
כעת, יש לנו 2 אופציות – הבן ירוץ לפני האב, או האב ירוץ לפני הבן:

האב ירוץ לפני הבן: האב ידפיס B לקובץ, אך סגרנו ופתחנו מחדש, לכן ה-seek ptr שלו יחזור ל-0. כעת הוא ימתין לבן. הבן ידרוס את B בכך שידפיס במקומו B (בכך לא שינינו כלום), וגם הוא סוגר ופותח מחדש, לכן ה-seek ptr שלו יחזור ל-0. כעת הוא ידפיס E לקובץ בעזרת printf, וידרוס את B. סיימנו עם הבן. כעת האב ימשיך לרוץ, אבל נזכור כי הוא מצביע לתחילת הקובץ, לכן הוא ידרוס את E בכך שידפיס במקומו E (בכך לא שינינו כלום), ויסיים.  
התוכן הסופי יהיה E.

הבן ירוץ לפני האב: דומה לעיל. הבן ידפיס B לקובץ, לא יחכה ב-wait, וידרוס את B עם E, שוב, כי סגרנו ופתחנו מחדש.  
כעת האב ידרוס את E עם B, לא יחכה ב-wait (כי הבן סיים וכעת ימחק), אבל ב-printf ידרוס את B עם E, שוב, כי סגרנו ופתחנו מחדש.  
התוכן הסופי יהיה E.

# 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 (זו התשובה הנכונה)
   3. 1
   4. 1 or 2
   5. 2

נימוק: בקוד יש לנו תהליך אב שיוצר 2 בנים – תהליך A ותהליך B.  
יש 2 אפשרויות:  
תהליך A רץ 'לפני' תהליך B: במקרה זה נדפיס פעם אחת את הפלט הרצוי, ואז ניכנס ללולאה אינסופית.  
אחרי זה B יהרוג את A.

תהליך B רץ לפני תהליך A: במקרה זה B קודם יהרוג את A, ועל כן תהליך A לא יקרא לפונקציה ProcessA(), ובכך לא נדפיס בכלל את הפלט הרצוי.

1. status: 1
   1. 0 (זו התשובה הנכונה)
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2

נימוק: לנמק

1. status: 137
   1. 0
   2. 0 or 1
   3. 1 (זו התשובה הנכונה)
   4. 1 or 2
   5. 2

נימוק: לנמק

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

נימוק: לנמק

1. The end
   1. 0
   2. 0 or 1
   3. 1
   4. 1 or 2
   5. 2 (זו התשובה הנכונה)

נימוק: כל התהליכים יגיעו לסוף של main, חוץ מ-A שמבצע exit.  
זה משאיר אותנו עם תהליך האב והתהליך B.

