TECHNION Israel Institute of Technology

Computer Science Department

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
3. The submission should contain the following:
   a. The first page should contain the details about the submitters - Name, ID number, and email address.
   b. Your answers to the dry part questions.
4. 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 היא מונח שבו מבצעים הכללה או הפשטה של מנגנון, תכונה של מערכת מחשב או פעולה על מנת לאפשר למשתמש יכולת שימוש פשוטה ללא הבנה פנימית של פרטי המימוש והקטנה או הסתרה של מספר הפרטים הספציפיים כלפי אותו רעיון עד כמה שניתן.

2. מדוע אנו משתמשים באבסטרקציות במערכות הפעלה?
למעשה אחד הרעיונות החשובים במדעי המחשב הוא לפרק בעיות ולטפל בכל בעיה בנפרד (divide and conquer). תחת רעיון האבסטרקציה אנחנו למעשה יוצרים API או הגדרה כללית לפעולה כלשהי ומאחורי הקלעים מבצעים את האלגוריתם הספציפי באופן מרוחק מהעין. בצורה זו אותו ממשק יכול לפעול מעל כל סט ספציפי של תנאי סביבה שבו הוא לא שולט ובצורה שכל משתמש יוכל להבין בפשטות מה מטרת העל של הממשק ולא איך הוא מבצע את הדברים בצורה פרטנית.

3. תנו דוגמה לאבסטרקציה מרכזית שמשתמשים בה במערכות הפעלה והסבירו מדוע משתמשים בה.
בAPI של POSIX המשותף למערכות הפעלה מבוססות UNIX, כאשר מנסים לכתוב לקובץ קוראים לopen עם נתיב לקובץ ופרמטרים נוספים, מבצעים את הכתיבה באמצעות write וה file descriptor שניתן לנו ולסיום סוגרים את הfd על ידי close. אותו קוד מסוגל לרוץ ולכתוב לקובץ לא משנה אם אותו קובץ שמור בHard Disk, Solid State Drive, קובץ זמני בRAM או קובץ שממוקם בכונן רשת כמו smb או nfs. וגם סוג הfilesystem בכלל לא מחייב (ntfs,zfs,ext4,xfs….). אותו קוד יצליח לבצע את המשימה בהצלחה ולכתוב את המידע לקובץ באמצעות הממשק האבסטרקטי שלנו. אם לא היינו עושים זאת היינו צריכים לכתוב אלפי תוכניות שונות בעלות מטרה זהה או תוכנית כבדה מאוד שמכילה כל פונקציה אפשרית. ובכל מקרה כמות האינפורמציה שהייתה דרושה כדי להבין איך לעבוד עם כל סוג של טכנולוגיה הייתה הופכת את תהליך יצירת התוכנה קשה עד לא סבירה עבור מתכנתים בעולם.

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: יצרנו בהתחלה קובץ חדש בשם file וכתבנו לו "B" וסגרנו אותו. אחר כך פתחנו אותו שוב, סגרנו את הstdout והstdin והפננו את הfd של הקובץ לאותם אינדקסים עם dup. כעת אנחנו יוצרים בן ומחכים באב לסיומו. קוראים לscanf כאשר הקריאה מפונת לקובץ מה שיכתוב b לs (הfile offset גדל באחד לאחר מכן) וכתובים לקובץ "C" באמצעות printf כאשר הדפסות מופנות לקובץ. וכותבים את המידע בs שמכיל B למסך. לאחר סיום הבן באב כותבים "E" ועושים סגירה אחת (לא מספיק באופן כללי כי עשינו dup) וחוזרים מהmain. בסך הכל סדר הכתיבה לקובץ היא לפי התוכן קובץ שרשמנו משמאל לימין.

2. (4 נק') במידה ונסיר את שורות 5+6.

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

נימוק: ההבדל הוא שבשלב זה אנחנו לא סוגרים את הקובץ ופותחים אותו מחדש ככה שה file offset יחזור ל0 אלא אנחנו נמצאים בEOF. לכן בבן לא נכתוב שום דבר לs באמצעות scanf. שאר הדברים מתקיימים כפי שתמיד היו.

3. (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: בגלל שאנחנו לא כותבים לs אז הוא נשמר כפי שהוא. בנוסף בגלל שבשום מקום לא מתבצעת קריאה מהfile, הfile offset הוא עדיין 0 ולכן בכתיבה הראשונה אנחנו נדרוס את התוכן שהיה בfile. במקרה שלנו ההדפסה של הבן "C" לתוך הfile תבצע דריסה של "B".

4. (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: שוב כמו בפעם הראשונה ללא שינויים אנחנו דואגים לקדם את הfile offset של הבן ולא לדרוס את ה"B" שנכתב גם לs ומודפס לstderr של המסף. במקום לסיים בנקודה הזאת הבן ממשיך לעשות wait שכמובן מסתיים ישר כי לבן אין בנים ומדפיס לקובץ "E" ואחריו האב שחיכה לסיום הבן וגם הוא בתורו מדפיס "E" לקובץ.

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

פלט התכנית (2 נק'): A או B תוכן הקובץ (3 נק'): EC או BCE או BEC

נימוק: בגלל שהאב לא מחכה לבן אז יש לנו שלוש אופציות:

1.האב כותב E לפני הscanf של הבן ובכך משכתב את הקובץ ומזיז את הoffset לסוף הקובץ. בבן הscanf לא יכתוב כלום לs ולכן יודפס למסך A ולקובץ C.

2. הבן כותב C לפני שהאב כותב. במצב זה הoffset מוזז לאחר הדפסה של B למסף. C מודפס לקובץ ואז E של האב.

3. האב כותב E אחרי הscanf של הבן ולפני כתיבת C של הבן. שוב אין דריסה של הקובץ בעקבות שימוש בscanf להזזת הoffset וכתיבת הB לs להדפסה למסף. אחרי כן E נכתב ולבסוף C לקובץ.

6. (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: בגלל שאנחנו מבצעים את הfork לפני קריאה לopen אז כל הfile objects שייווצרו לאחר open לא יהיו משותפים לשני התהליכים ולכן יהיו בעלי offsetים נפרדים. אנחנו כותבים רק אחרי תמיד את B לפני שאנחנו סוגרים וכותבים שוב את E באותו offset ולכן תמיד להאם יש אופציה שתהליך ישכתב כתיבה של E עם B מובטח שלאחר מכן תהיה שוב פעם כתיבה של E.

הסיבה שאין פלט היא שמחקנו את הפעולה היחידה שנמצאת במצב שהיא כותבת עוד לfd שכותב למסף והיא write(STDERR_FILENO, s, 1). printf("E") כותב כבק לקובץ לאחר הdupים.

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 x + 128.

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

1. process A
   a. 0
   b. 0 or 1
   c. 1
   d. 1 or 2
   e. 2

נימוק: בפעם הראשונה שנקרא הfork, הבן מריץ את הפונקציה processA עם העתק X=1 ולכן ידפיס את המחרוזת ומיד יכנס ללולאה אינסופית. B לעומת זאת כבר מקבל העתק של pid1 עם הpid של A ושולח לו KILLSIG. יכול להיות שA יקבל אותו כבר אחרי הדפסה ויצא ויכול להיות שהוא יקבל לפני הדפסה ויצא מבלי להדפיס.

2. status: 1
   a. 0
   b. 0 or 1
   c. 1
   d. 1 or 2
   e. 2

נימוק: הexit code של הבן שקורא לprocessA (נקרא לו A) הוא 137 ושל הבן השני שקורא לprocessB (נקרא לו B) הוא 3 כי לא נשלח לו סיגנל בגלל שאצלו pid2=0 ולכן לו תהיה שום הדפסה כזאת אצל האבא.

3. status: 137
   a. 0
   b. 0 or 1
   c. 1
   d. 1 or 2
   e. 2

נימוק: לאחר שA נוצר גם B נוצר עם הערכים x=1,p1=A_pid,p2=0 ולכן בהמשך בתוך killall רק A מקבל signal וערכו הוא 9. A במקרה שלנו רק מדפיס ונכנס ללולאה אינסופית ויוצא בסיום עם 9+128

4. status: 143

a. 0
b. 0 or 1
c. 1
d. 1 or 2
e. 2

נימוק: כל התהליכים מסתיימים עם ערך יציאה שהוא 137 או 3 אם לא קיבלו סיגנל.

5. The end

a. 0
b. 0 or 1
c. 1
d. 1 or 2
e. 2

נימוק: יש לנו שני תהליכים שלא מקבלים סיגנל ומסיימים את הריצה שלהם בחזרה מהmain חוץ מA שמקבל סיגנל שהורג אותו. ולכן יהיו לנו שתי הדפסות.