TECHNION
Israel Institute of Technology

Computer Science Department

Operating Systems – 234123

Homework Exercise 1 – Dry

Teaching Assistant in charge:

Reda Agbarya

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://drive.google.com/open?id=1RiJCQ0mFee3cQH2uB0KBICWEYktZGgnjDM7YMr3QvWY

Part 1 – system calls(50 points)

חברת MaKore, הכורה (mining) מטבעות דיגיטליים, מריצה בכל רגע מספר גדול של תהליכים על-מנת להאיץ את פעולת הכרייה. התהליכים שומרים את תוצאות החישובים שלהם לקבצים בדיסק כדי למנוע אובדן מידע במקרה שהתהליך קורס לפתע. בכל שניה התהליך קורא לפונקציה הבאה כדי ליצור את שם הקובץ שבו ייכתב הפלט שלו:

```
#include <string>
using namespace std;

string create_file_name(time_t timestamp) {
    pid_t pid = getpid();
    string s = "results-" + to_string(pid) + to_string(timestamp);
    return s;
}
```

כפי שניתן לראות, כל תהליך מוסיף את ה-PID שלו לשם הקובץ כדי למנוע התנגשות בין קבצים של תהליכים שונים. לצורך הפשטות, לאורך כל השאלה הניחו כי החברה אינה משתמשת בחוטים כלל.

1. היכן הגרעין שומר את ה-PID של התהליך?

a. בספריה libc.
b. במחסנית המשתמש.
c. במחסנית הגרעין.
d. בערימה.
e. במתאר התהליך (ה-PCB).
f. בתור הריצה (runqueue).

נימוק:
כפי שלמדנו הגרעין שומר את כל המידע בנוגע לתהליך מסוים כולל PID שלו בתוך מתאר תהליך PCB.

שרה, בוגרת הקורס ומהנדסת צעירה בחברה, הבחינה כי הפונקציה הנ"ל נקראת פעמים רבות במהלך הריצה של כל תהליך. לכן שרה הציעה את השיפור הבא לקוד המקורי:

```
pid_t pid = getpid();

string create_file_name(time_t timestamp) {
    string s = "results-" + to_string(pid) + to_string(timestamp);
    return s;
}
```

2. מדוע הפתרון של שרה עדיף על המימוש המקורי?

נימוק:
המימוש של שרה מאפשר לחסוך במספר קריאות מערכת. כל תהליך מבצע קריאת מערכת יחידה getpid לפני כל קריאה לפונקציה במקום לבצע קריאה כל פעם שנכנסים לפונקציה. זה חוסך בזמן ריצה כי כל קריאת מערכת דורשת מעבר בין קוד משתמש לקוד גרעין.

דנה, מהנדסת בכירה בחברה, התלהבה מהרעיון של שרה והחליטה לקחת אותו צעד אחד קדימה. דנה עדכנה את פונקצית המעטפת (wrapper function) של קריאת המערכת getpid() כפי שמופיעה בספרית libc באופן הבא:

```
+   pid_t cached_pid = -1; // global variable

    pid_t getpid() {
        unsigned int res;
+       if (cached_pid != -1) {
+           return cached_pid;
+       }
        __asm__ volatile(
            "int 0x80;"
                :"=a"(res) :"a"(__NR_getpid) :"memory"
          );
+         cached_pid = res;
          return res;
      }
```

- שורות מסומנות ב-"+" הן שורות שדנה הוסיפה לקוד המקורי. אלו השורות היחידות שהשתנו בספריה.
- תזכורת: שורת האסמבלי שומרת את הערך "__NR_getpid" ברגיסטר eax לפני ביצוע הפקודה, ומציבה את ערך eax לאחר ביצוע הפקודה במשתנה res.

שרה השתמשה בספריה החדשה (של דנה), אך תוכניות מסוימות שעבדו לפני השינוי הפסיקו לעבוד כנדרש עם הספריה החדשה.

3. מהי התקלה שנוצרה בעקבות השינוי?

a. אם שני תהליכים קוראים ל-getpid() בו-זמנית עלול להיווצר race condition.
b. fork() עלולה לחזור עם אותו ערך בתהליך האב ובתהליך הבן.
c. fork() עלולה להיכשל במקרים בהם המימוש המקורי היה מצליח.
d. getpid() עלולה להחזיר pid של תהליך אחר.
e. getpid() עלולה להחזיר "-1".
f. execv() עלולה להיכשל במקרים בהם המימוש המקורי היה מצליח.

נימוק:
קריאת מערכת fork יוצרת עותק של תהליך הבא לכן לבן יהיה משתנה גלובלי cached_pid עם אותו ערך של הבא שהוא בפרט לא -1, ולכן כשנקרא ל- getpid עבור תהליך הבן אנחנו נחזיר את pid של הבא.

כדי לתקן את התקלה שנוצרה, דנה מציעה **בנוסף** את התיקון הבא של פונקציית המעטפת של fork:

```
1.    +    // the same global variable from above
2.    +    extern pid_t cached_pid;
3.
4.         pid_t fork() {
5.             unsigned int res;
6.             __asm__ volatile(
7.                 "int 0x80;"
8.                 :"=a"(res) :"a"(__NR_fork) :"memory"
9.             );
10.        +        ???
11.                return res;
12.          }
```

4. השלימו את התיקון הנדרש בשורה 10:

a. `if (res == 0) cached_pid = -1;`
b. `if (res == 0) cached_pid = getpid();`
c. `if (res == 0) return cached_pid;`
d. `if (res > 0) cached_pid = -1;`
e. `if (res > 0) cached_pid = getpid();`
f. `if (res > 0) return cached_pid;`

נימוק:
במידע ו-fork מחזיר 0 זה אומר שאנחנו נמצאים בתהליך של בן, בגלל העתקת מידע ה-cached_pid שלו מחזיק את הערך של הבא, כדי לגרום לקריאת מערכת getpid לעבוד בצורה נכונה, נשים במשתנה הזה ערך -1 ואז לפי התיקון של דנה ב-getpid שורת האסמבלי תחזיר את pid הנכון שייכנס לcached_pid .

סאטושי, מנהל החברה, הבחין כי למרות התיקון לעיל, הספריה החדשה עדיין בעייתית כאשר הקוד משתמש בסיגנלים. סאטושי הדגים את הבעיה באמצעות הקוד הבא:

```
1.    void my_signal_handler(int signum) {
2.        cout << getpid() << endl;
3.    }
4.
5.    int main() {
6.        // set a new signal handler
7.        signal(SIGUSR1, my_signal_handler);
8.        pid_t pid = fork();
9.        if (pid > 0) { // parent
10.             kill(pid, SIGUSR1); // send a signal to the
   child
11.             wait(NULL);
12.         }
13.     }
```

.5 (5 נק') מהי התקלה בקוד לעיל ומתי היא תתרחש?

a. הסיגנל לא יטופל אם האב שלח את הסיגנל לפני שהבן התחיל לרוץ.
b. הסיגנל לא יטופל אם האב שלח את הסיגנל אחרי שהבן סיים את שורה 8.
c. הסיגנל לא יטופל ללא תלות בסדר הזימון של התהליכים.
d. שורה 2 תדפיס ערך שגוי אם האב שלח את הסיגנל לפני שהבן התחיל לרוץ.
e. שורה 2 תדפיס ערך שגוי אם האב שלח את הסיגנל אחרי שהבן סיים את שורה 8.
f. שורה 2 תדפיס ערך שגוי ללא תלות בסדר הזימון של התהליכים.

נימוק:

במידע ותהליך אב שלח את הסיגנל לפני שהבן התחיל לרוץ אז במשתנה גלובלי של הבן cached_pid נמצא עדיין ערך של אבא ולא -1 ולכן קריאה getpid שנמצאת בשגרת טיפול של סיגנל הנשלח תחזיר מזהה של אבא ולא של הבן.

Part 2 - Pipes & I/O (50 points):

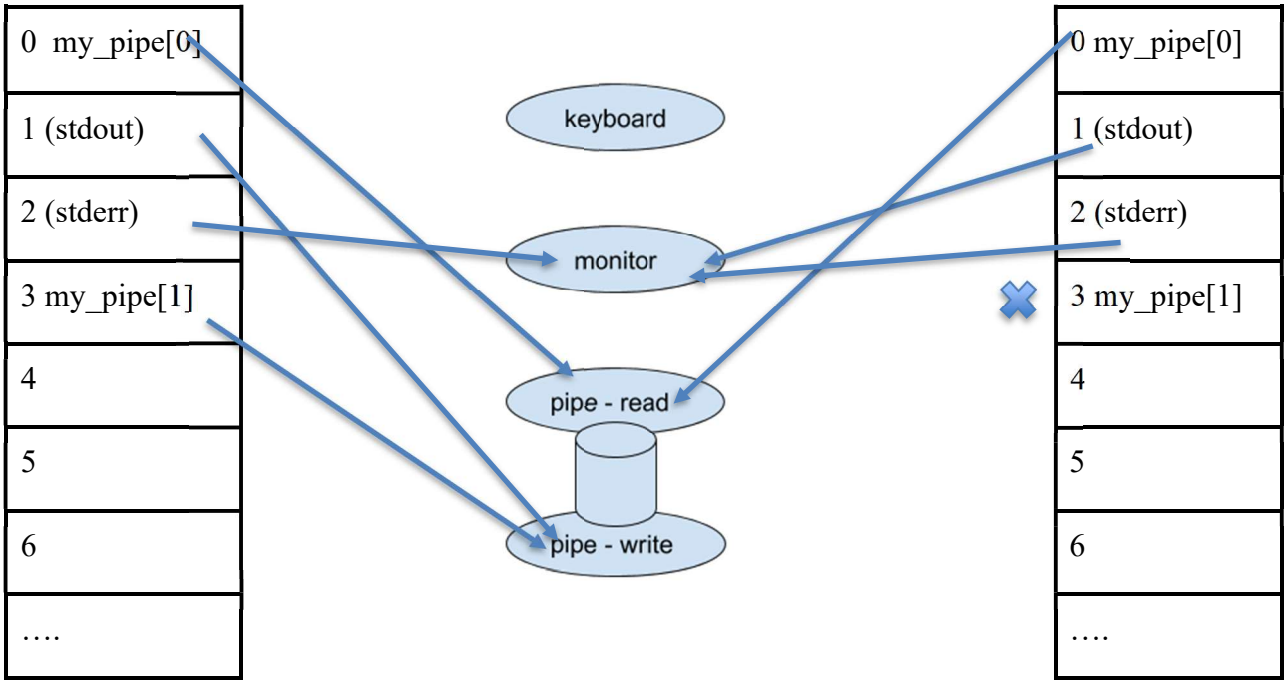
נתון קטע הקוד הבא:

```
1. void transfer() {// transfer chars from STDIN to STDOUT
2.   char c;
3.   ssize_t ret = 1;
4.   while ((read(0, &c, 1) > 0) && ret > 0)
5.        ret = write(1, &c, 1);
6.   exit(0);
7. }
8.
9. int main() {
10.  int my_pipe[2];
11.  close(0);
12.  printf("Hi");
13.  pipe(my_pipe);
14.  if (fork() == 0) { // son process
15.       close(my_pipe[1]);
16.       transfer();
17.  }
18.  close(1);
19.  dup(my_pipe[1]);
20.  printf("Bye");
21.  return 0;
}
```

1. השלימו באמצעות חצים את כל ההצבעות החסרות באיור הבא (למשל חץ מ- stdin ל- keyboard), בהינתן שתהליך האב סיים לבצע את שורה 19 ותהליך הבן סיים לבצע את שורה 15:

אבא

בן



2. מה יודפס למסך בסיום ריצת שני התהליכים? (הניחו שקריאות המערכת אינן נכשלות):

a. Hi
b. Bye
c. HiBye
d. לא יודפס כלום
e. התהליך לא יסתיים לעולם
f. לא ניתן לדעת, תלוי בתזמון של התהליכים

נימוק:

בשורה 12 ההדפסה תוציא את ההודעה למסך כי stdout עדיין פתוח.

תהליך הבן יכתוב למסך Bye כי הוא קורא מערוץ 0 שלו שמחובר לpipe-read ומחכה עד שיקבל קלט, לכן הוא יחכה עד שהאבא יכתוב לpipe-write בשורה 20.

בסעיפים הבאים נתבונן בקטע קוד חדש, המשתמש בפונקציה transfer מהסעיף הקודם:

```
1. int my_pipe[2][2];
2. void plumber(int fd) {
3.   close(fd);
4.   dup(my_pipe[1][fd]);
5.   close(my_pipe[1][0]);
6.   close(my_pipe[1][1]);
7.   transfer();
8. }
9.
10.int main() {
11.  close(0);
12.  printf("Hi");
13.  close(1);
14.  pipe(my_pipe[0]);
15.  pipe(my_pipe[1]);
16.
17.  if (fork() == 0) { // son 1
18.       plumber(1);
19.  }
20.  if (fork() == 0) { // son 2
21.  plumber(0);
22.  }
23.  printf("Bye");
24.  return 0;
}
```

3. מה יודפס למסך כאשר תהליך האב יסיים לרוץ? (הניחו שקריאות המערכת אינן נכשלות) רמז: שרטטו דיאגרמה של טבלאות הקבצים כפי שראיתם בסעיף 1.

a. Hi

b. Bye

c. HiBye

d. ByeHi

e. לא יודפס כלום

f. לא ניתן לדעת, תלוי בתזמון של התהליכים...

נימוק:

בשורה 12 ערוף 1 מצביע למסך ולכן יכתב למסך HI. לאחר מכן , כתוצאה מפעולות ה- plumber של שני הבנים, בן 1 יעביר את כל מה שיוכנס ל pipe[0] לתוך הכתיבה של Pipe[1], בן 2 יעביר את כל מה שיוכנס ל pipe[1] לתוך הכתיבה של Pipe[0]. כתוצאה מכך, כאשר האב יכתוב Bye זה יועבר ל-pipe[0] , ובן 1 יעביר את זה ל-pipe[1], ובן 2 יעביר את זה ל-pipe[0], בן 1 יעביר את זה ל pipe[1] וחוזר חלילה, ולכן "Bye" לעולם לא יודפס למסך, והתהליכים ירוצו לעד.

סנטה קלאוס שמע שסטודנטים רבים בקורס עבדו במהלך הכריסמס על תרגיל הבית, ואפילו נהנו ממנו יותר מאשר במסיבת הסילבסטר של הטכניון. בתגובה נזעמת, סנטה התחבר לשרת הפקולטה והריץ את התוכנית הנ"ל N פעמים באופן סדרתי (דוגמה ב-bash, כאשר out.a הוא קובץ ההרצה של התכנית הנ"ל):

>> for i in {1..N}; do ./a.out;

4. אחרי שהלולאה הסתיימה, נשארו במערכת 0 או יותר תהליכים חדשים.
מה המספר המינימלי של סיגנלים שצריך לשלוח באמצעות kill על מנת להרוג את כל התהליכים החדשים שסנטה יצר?

a. 0
b. 1
c. N
d. N/2
e. 2N
f. לא ניתן לדעת, תלוי בתזמון של התהליכים…

נימוק:

כפי שהסברנו בשאלה הקודמת, הרצה יחידה של a.out יוצרת שני תהליכים שרצים לעד (תהליך האבא מת), לכן הרצה של N a.out פעמים תיצור 2N תהליכים שיש להרוג אותם. בהנחה ששליחת סיגנל kill הורגת תהליך יחיד, נצטרך לשלוח 2N סיגנלים באמצעות kill כדי להרוג את כל התהליכים.

5. מה תהיה התשובה עבור הסעיף הקודם אם נסיר את שורות 5-6 מהקוד?

a. 0
b. 1
c. N
d. N/2
e. 2N
f. לא ניתן לדעת, תלוי בתזמון של התהליכים…

נימוק:

הורדת שורות 5-6 לא תמנע מהתהליכים הבנים להפסיק לרוץ, לכן נצטרך לשלוח אותו מספר של סיגנלים כמו בשאלה 4 כדי להרוג אותם.