TECHNION
Israel Institute of Technology

Computer Science Department

Operating Systems – 234123

Homework Exercise 1 – Dry

:Teaching Assistant in charge

Mohammad 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.

(Part 1 – Input/Output/Signals (50 points

קלט/פלט וסיגנלים (50 נק')

א. מה יודפס על המסך עבור הקוד הבא? נמקו

```
int main() {

     int res = fork();

     if (res!=0) {

          close(STDOUT);

     }

     int fd = open("myFile", O_RDWR);

     if (res!=0) {

          printf("Hello from father\n");

     } else {

          printf("Hello from son\n");

     }

}
```

פלט למסך: "Hello from son".

הסבר: בעת פתיחת תהליך חדש ע"י פעולת fork, ה-FDT של תהליך האב מועתק לבן.

לאחר ביצוע fork, תהליך האב מבצע סגירה של הקובץ המוצבע ע"י STDOUT, במקרה זה, הפלט הסטנדרטי, כלומר, מנקודה זו, הפלט הסטדנרטי לא ניתן לגישה מתהליך האב.

כל שינוי ב-FDT אצל האב\בן לאחר fork לא נראה אצל השני.

שני התהליכים מבצעים פתיחה של קובץ myFile עם הרשאות Read,Write, במקרה והפעולה הצליחה נוסף מצביע לסוף טבלת FDT שמצביע ל-file object של הקובץ שנפתח.

בסוף הפונקציה מתבצעות הדפסות לפלט הסטנדרטי, בתהליך האב קובץ זה נסגר, לכן לא מודפס כלום. לעומת זאת בתהליך הבן קובץ הפלט הסטדנרטי פתוח ולכן מתרחשת הדפסה למסך.

ב. כיצד תשתנה התשובה לסעיף הקודם אם נשנה את הקוד כך:

```
int main() {

     close(STDOUT);

     int res = fork();

     int fd = open("myFile", O_RDWR);

     if (res!=0) {

          printf("Hello from father\n");

     } else {

          printf("Hello from son\n");

     }

}
```

פלט: אין פלט

הסבר: תהליך האב סוגר את הפלט הסטדנרטי, לאחר מכן מבצע fork הפותח תהליך חדש עם <u>טבלת FDT מועתקת מתהליך האב.</u>

בשונה מסעיף א', בטבלת ה-FDT של הבן לא נמצא ה-FD של הפלט הסטדנרטי, לכן פעולת הדפסה למסך לא תתרחש הן בתהליך האב והן בתהליך הבן.

ג. נתון קטע הקוד הבא (תניחו ש-buf מאותחל במקום כלשהו בקוד):

```
#define BUF_SIZE 65600 // bigger than 64K
int my_pipe[2];
pipe(my_pipe);
char buf[BUF_SIZE];
int status = fork();

//Filled buf with message...
if (status == 0) {  /* son process */
     close(my_pipe[0]);
     write(my_pipe[1],buf,BUF_SIZE*sizeof(char));
     exit(0);
 }
 else {    /* father process */
     close(my_pipe[1]);
     wait(&status); /* wait until son process finishes */
     read(my_pipe[0], buf, BUF_SIZE*sizeof(char));
     printf("Got from pipe: %s\n", buf);
     exit(0);
 }
}
```

1. מה הבעיה בקטע הקוד הנתון?

גודל הבאפר יוצר בעייתיות מהבחינה שהוא גדול מגודל נתיב הכתיבה של ה-PIPE שגודלו הסטנדרטי הינו 64KB.

בתוכנית, תהליך האב ממתין לסיום תהליך הבן כדי לקרוא מנתיב הקריאה של ה-PIPE.

אולם תהליך הבן לא יצליח לכתוב לתוך ה-PIPE מאחר והוא חוצה את קיבולת ה-PIPE בוודאות, ולכן יחכה עד שתהליך אחר יפנה את ה-PIPE ע"י קריאה ממנו.

התוכנית מגיעה למצב בו שני התהליכים הנ"ל מחכים אחד לשני עד אינסוף -ולכן תהליך הבן ימשיך לחכות שיפנו מקום ב-PIPE בעוד שאב יחכה עד אשר תהליך הבן יסתיים.

כמו כן, מאחר והבאפר גדול מגודל הPIPE, לא תספיק קריאה בודדת מתהליך האב כדי לקרוא את כל המחרוזת, שכן אפשר לקרוא לכל היותר 64KB בבת אחת, כלומר, בעיה נוספת היא שהאב לא קורא את כל המחרוזת שמספק הבן (איבוד מידע).

2. הראו כיצד ניתן לתקן את הבעיה ~~עבור הקוד שמבצע תהליך הבן~~ דרך עדכון קוד של האבא והבן?.

ניתן לבצע כתיבה בחלקים מתהליך הבן, תוך כדי קריאה מה-PIPE בתהליך האב.

בתהליך האב, נוריד את פקודת WAIT ונבצע קריאה מה-PIPE כל עוד מתרחשת כתיבה.

```
while (true) {
     int readCount = read(my_pipe[0], buf, 64000 *
sizeof(char));
     if (readCount == -1) {
          perror("error");
          break;
     } else if (readCount == 0) {
          printf("Finished Reading\n");
          break;
     } else {
          printf("Got from pipe: %s\n", buf);
     }
}
```

בתהליך הבן נסגור את ה-PIPE לאחר הכתיבה.

נכונות:

תהליך הבן יכתוב ל-PIPE מידע ששמור ב-buf עד שיסיים.

תהליך האב ימשיך לקרוא מה-PIPE עד אשר לא יהיה מידע ב-PIPE.

בסוף העבודה שני התהליכים סוגרים את ה-PIPE כנדרש.

ד. נתון הקוד הבא:

```
int x=0;
int i=3;
void catcher3(int signum) {
     i=1;
}
void catcher2(int signum) {
     if (i!=0) {
          x=5;
     }
}
void catcher1(int signum) {
     printf("%d\n", i);
     i--;
     if (i==0) {
          signal(SIGFPE, catcher2);
          signal(SIGTERM, catcher3);
     }
}
int main() {
     signal(SIGFPE, catcher1);
     x = 10/x;
     printf("Goodbye");
}
```

1. מה יודפס למסך כאשר מריצים את הקוד כמו שהוא? נמקו

3
2
1

כאשר אנו מבצעים חלוקה באפס (Erroneous arithmetic operation) הסיגנל SIGFPE נרשם ב-

PCB של התהליך, בחזרה ממצב גרעין למצב משתמש, המערכת מזהה את הסיגנל הממתין ומטפלת בו ע"י הרצת catcher1.

בשיגרת הטיפול catcher1, בסיום ריצת השיגרה אנו חוזרים לאותה נקודה בקוד שהיא **קבלת הסיגנל**, מה שגורם ל"לולאה" בה מבצעים הדפסות למסך עד קיום תנאי i==0.
בהגעה לתנאי, אנו מגדירים מחדש את אופן הטיפול בסיגנל SIGFPE לפונקציה catcher2.
מיד לאחר מכן חוזרים לקוד ו**מקבלים את אותו הסיגנל רק שהפעם השגרה הנקראת היא catcher2 עם i==0**, לכן שום דבר לא קורא ואנו תקועים ב"לולאה אינסופית" של חזרה לשגרה הראשית, קבלת סיגנל וטיפול ב-catcher2.

2. מה יודפס למסך אם נתון שלאחר 10 שניות מבצעים את הפקודה "kill <PID>" בטרמינל (כאשר ה-PID הנתון הוא של התהליך שמריץ את הקוד הנ"ל)

3
2
1
Goodbye

בעת שליחת סיגנל SIGTERM, השגרה catcher3 נקראת כפי שהוגדרה בתנאי i==0.
בשגרה זו משנים את המשתנה הגלובלי i ובגלל שאין ערך חזרה מהשגרה ממשיכים מהנקודה שהתוכנית נעצרה.
מלפני קבלת SIGTERM התוכנית נמצאת בלולאה אינסופית, אך כעת כאשר נגיע לשגרה catcher2, בגלל השינוי של המשתנה הגלובלי הנ"ל נעמוד בתנאי i!=0 ונשנה את ערכו של x מערכו הקודם 0.

כעת בגלל שאין ערך חזרה משגרה catcher2, התוכנית תחזור לנקודה בו התחרש הסיגנל והוא חלוקה ב-0, אך בגלל שינוי המשתנה תתרחש חלוקה חוקית והתוכנית לא תקבל סיגנל, תתבצע הדפסה אחרונה למסך והתוכנית תסתיים.

3. מה יודפס למסך אם נוסיף פקודת sleep(100); //100 seconds אחרי השורה (signal(SIGFPE,catcher
שנמצאת בפונקציה catcher1? (כאשר עדיין שולחים סיגנל בעזרת kill לאחר 10 שניות מתחילת ריצת התכנית) נמקו

3
2
1
ולאחר מכן התהליך הסתיים.
ברגע ביצוע sleep התהליך עובר למצב המתנה.
כאשר מבצעים את הפקודה "kill <PID>" , התהליך "מתעורר" ונרשם לו pending signal מסוג SIGTERM .
לכן רגע לפני חזרה ממצב KERNEL לUSER , התהליך יטפל בPENDING SIGNAL, ומאחר ולא הספקנו לשנות את הטיפול בSIGTERM לcatcher3 , התהליך יטפל בSIGTERM ע"י טיפול ברירת המחדל של הסיגנל - סיום התהליך.

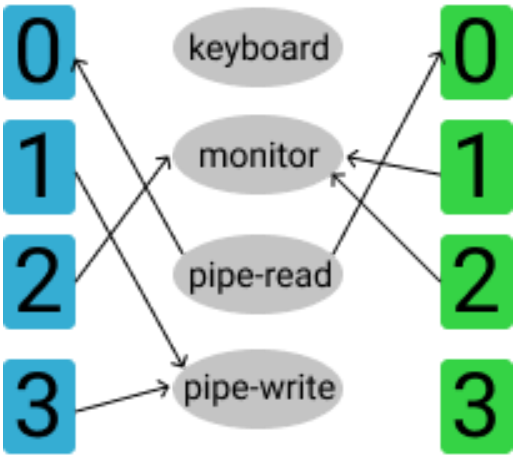
:(Part 2 - Pipes & I/O (50 points

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:

הבן מסומן בירוק,האב בכחול.



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:

הסבר לאבא: פעולת הדפסת למסך printf מבצעת כתיבה ל-stdout, כלומר ל-FD שמספרו 1.

עפ"י הדיאגרמה מסעיף קודם, FD-1 מבצע כתיבה ל-pipe-write.

כלומר, המחרוזת HiBye נשלחת ל-pipe-write.

הסבר לבן: בפונקציה transfer מתבצעת פעולת קריאה מ-FD שמספרו 0, וכתיבה ל-FD שמספרו 1.

עפ"י הדיאגרמה מסעיף קודם, מתרחשת קריאה מ-pipe-read וכתיבה ל-monitor.

כלומר, המחרוזת HiBye נקראת מ-pipe-read ומוצגת על המסך.

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:

תהליך אבא: כתיבה HiBye ל-FD-1 שהוא הערך ב- my_pipe_01.

נשים לב שהאב סוגר את FD 1,0 ובעת הגדרת ה-pipe הראשון, נתפסים ה-FD שנסגרו ע"י ה-pipe.

עבור הFDT של האב, המצב הוא ש-my_pipe0 תופס את FD 0,1, ו-my_pipe1 תופס את FD 3,4.

תהליך בן 1: קריאה מ-FD-0 שהוא הערך ב- my_pipe_00, וכתיבה ל-FD-1 שהוא הערך ב-my_pipe11.

כלומר התהליך קורא את HiBye וכותב אותו ל-my_pipe1.

תהליך בן 2: קריאה מ-FD-0 שהוא הערך ב-my_pipe_10, וכתיבה ל-FD-1 שהוא הערך ב-my_pipe01.

כלומר התהליך קורא את HiBye שהועבר מ-בן1 וכותב אותו ל-my_pipe0,

לסיכום, התוכנית רושמת מחרוזת דרך האב לתוך my_pipe1, הבן ה-1 קורא את המידע וכותב אותו ל-my_pipe2, הבן ה-2 קורא את המידע וכותב אותו חזרה ל-my_pipe1.

בשום שלב אין כתיבה לפלט הסטנדרטי לכן לא יודפס כלום.

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:

בכל איטרציה בלולאת for , תהליך האב (שמריץ את a.out) , יוצר שני בנים שמקושרים ביניהם דרך שני צינורות.

בסעיף קודם ראינו כי שני הבנים נכנסים ללולאה אינסופית שכל אחד שולח לשני תוכן דרך הצינורות.

לכן, מספיק לשלוח סיגנל סיום לאחד הבנים כדי שהאח השני יסתיים גם כן - סה"כ צריך N סיגנלים.

הסבר מעמיק:

נניח בה"כ כי נסיים את תהליך son1 (באיטרציה כלשהי), אזי אם הצינור דרכו son2 קורא נתונים מson1 ריק, אז הוא יקבל EOF מאחר ואין כותב בצינור זה.

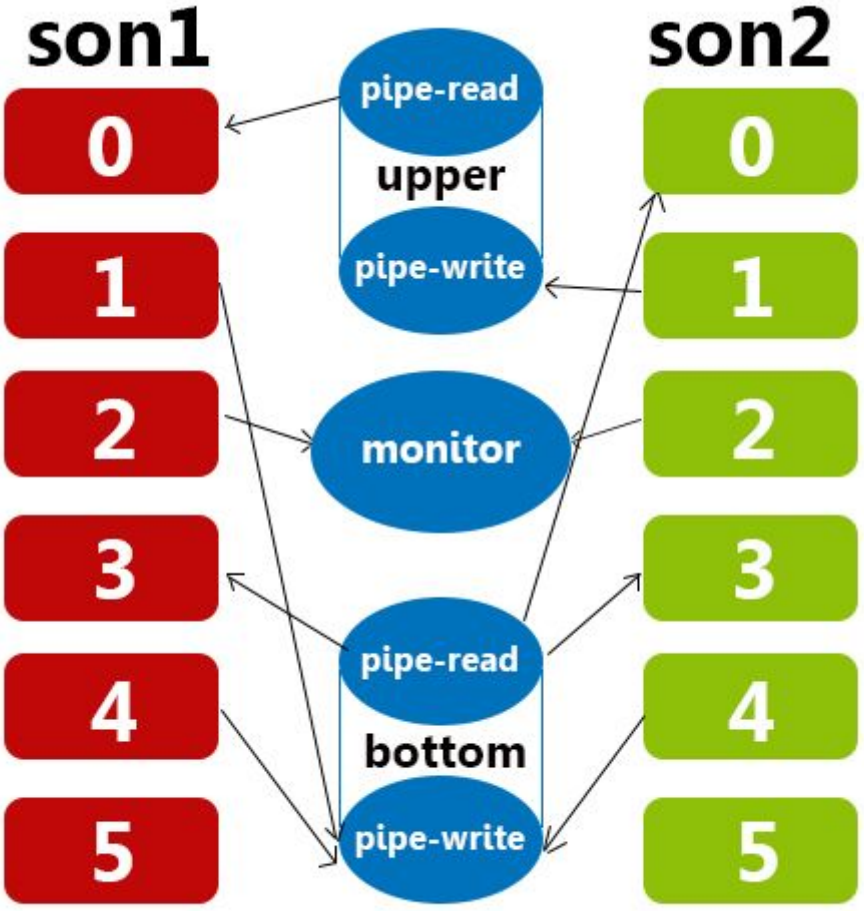
באופן סימטרי גם הצינור בו son2 כותב יסגר מאחר וson1 לא קורא יותר, ובסה"כ son2 יסיים את פעילותו וימות.

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:

עבור מחיקת שורות 5-6 מהקוד נקבל את הדיאגרמה הבאה:



בניגוד לסעיף קודם , האחים אינם סימטרים במובן שהריגת son1 אינה תוביל בהכרח להריגת son2.

ההסבר לכך הוא שתתכן נקודת זמן בה בזמן שson1 מסתיים, son2 מנסה לקרוא מהצינור התחתון, ויכנס להמתנה לנצח מאחר ועדין יש קצוות כותבים מהצד של son1.(גם אם לא יכתבו שום מידע)

על כן,עלינו לשלוח סיגנל לסיום תהליך son2. נפריד למקרים עבור מצב הצינור העליון.

אם הוא ריק כאשר מסתיים son2, אז son1 יקבל EOF וישלח סיגנל SIGPIPE שיוביל לסיום son1 גם כן.

אחרת, son1 יקרא תו וינסה לשלוח אותו לson2 דרך הצינור התחתון, ולא יקבל SIGPIPE מאחר ויש עדיין קוראים מהצד השני.

למרות זאת, בזמן מסוים son1 יסיים לקרוא את כל התווים בצינור העליון ויקבל EOF והתהליך יסתיים.