מגישים:

אמיר פוסטילניק 316397843 **ampustilnik@campus.technion.ac.il**

יהונתן יוסף 203304480 **jonathanjose@campus.technion.ac.il**

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:
ראינו בהרצאה

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:
שרל'ה משתמשת במשתנה גלובלי על מנת להעביר את הpid לפונקציה במקום להשתמש בקריאות מערכת רבות אשר כידוע, עוצרות את התהליך ומעבירות את השליטה למערכת ההפעלה. מה שכמובן לוקח זמן רב.

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:
במידה ותהליך קרא לפחות פעם אחת ל ()getpid לפני ()fork . הפיצול ישמור את ה cached_pid גם לתהליך החדש שנוצר – ולכן אם הבן ייקרא ל ()getpid הוא יקבל את ה- pid של האב (או אב קדמון כלשהו).

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:
כפי שראינו בתרגול, fork() מחזירה 0 (שמאוכסן ב-res) אם התהליך הוא תהליך הבן. ולכן על מנת ש getpid() תעבוד כראוי – cached_pid צריך להיות מאותחל על 1-.

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.

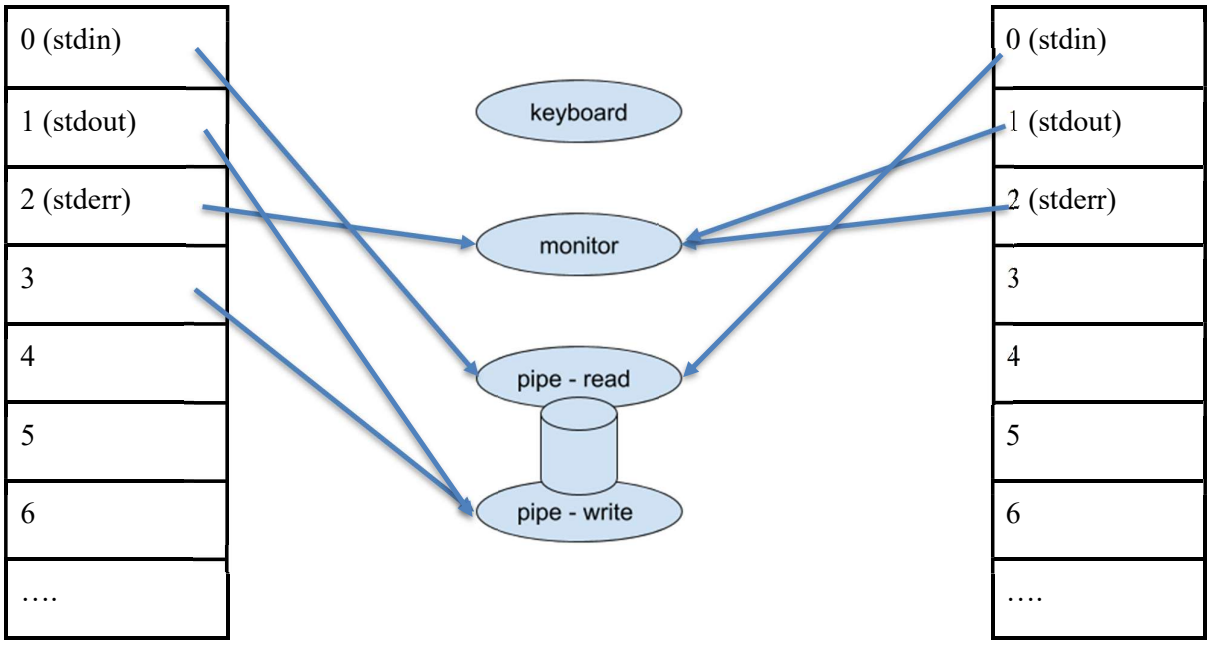
נימוק:
לדעתנו, התכנית המדוייקת תעבוד כראוי. נתייחס לתכנית בה האב קראה ל()getpid לפני הקריאה ל ()fork .
במידה והאב יישלח את הסיגנל לפני שהבן יסיים את הטיפול בשורה 8, יוכל להיווצר מצב שבאופן זמני ה – cached_pid של הבן יהיה בעצם הpid של האב (כי לא יספיק להתעדכן על -1).

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:



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:

התוכנה מדפיסה תחילה למסך את המילה Hi.
לפני ה fork התוכנה סוגרת את ערוץ הקלט הסטנדרטי (המקלדת).
ערך הקריאה של ה pipe מקבל את הערך הפנוי הנמוך ביותר (0) .
האב סוגר את ערוץ הפלט הסטנדרטי (המסך).
האב מבצע dup לערך הכתיבה של הצינור שמקבל את הערך הפנוי הנמוך ביותר (1) .
האב מעביר לבן את המילה Bye אשר קורא אותה מהצינור ופולט אותה למסך.

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

```
int my_pipe[2][2];
void plumber(int fd) {
  close(fd);
  dup(my_pipe[1][fd]);
  close(my_pipe[1][0]);
  close(my_pipe[1][1]);
  transfer();
}

int main() {
  close(0);
  printf("Hi");
  close(1);
  pipe(my_pipe[0]); //0,1
  pipe(my_pipe[1]); //3,4

  if (fork() == 0) { // son 1
       plumber(1);
  }
  if (fork() == 0) { // son 2
  plumber(0);
  }
  printf("Bye");
  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:

מודפס Hi למסך לפני שהמסך של האבא נסגר.
הבנים יוצרים צינור דו כיווני ביניהם כך שהבן הראשון קולט מ0 ופולט ל1 והבן השני מתנהג בדיוק להפך מה שייצור לולאה של קריאות וכתיבות בין הבנים. האב שולח לבן הראשון Bye . אף אחד מהתהליכים לא מדפיס למסך מכיוון שהם סגרו את ערוץ הפלט של המסך.

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:

כל תהליך מ- N התהליכים ייפתח עוד 2 וייסגר בעצמו. על כן יישארו 2N תהליכים פתוחים שמדברים בינהם ולא יפסיקו לעולם. על מנת לסגור את כלל התהליכים מספיק לשלוח סיגנל רק לאחד הבנים , הבן השני ייסגר כאשר הוא פעולת transfer תסתיים מכיוון שהיא משתמשת בexit. (פעולת transfer תסתיים כי לא יהיה אף אחד שייכתוב לערוץ הקלט שהתהליך מנסה לקרוא ממנו, ולכן read לא יחסום) .

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: נגדיר myPipe[0] = צינור 0

נשלח את הסיגנל לבן השני בכל אחד מ N התהליכים שנוצרו בלולאה, מכיוון שהוא ייסגר אף תהליך לא יצביע על כתיבה לצינור 0 ולכן כאשר הבן הראשון ינסה לקרוא מצינור 0 לאחר שהוא התרוקן הוא יגיע לEOF שנגרם מכך שאף תהליך אינו מצביע על צינור 0. והתהליכים יסתיימו.
אם היינו מנסים לסגור את הבן ראשון קודם, הבן השני היה מצביע על כתיבה לצינור 1 ולכן לא היה מסיים את תפקודו בצורה טבעית והיה מחכה לקלט נוסף לנצח.