Operating Systems – 234123

Homework Exercise 2 – Dry

Submitters:

Tom Guy

315155671

tom.guy@campus.technion.ac.il

Nikita Ivanov

322245101

Nikitaiv@campus.technion.ac.il

חלק 1 - שאלות בנושא התרגיל הרטוב (50 נק')

מומלץ לקרוא את הסעיפים בחלק זה לפני העבודה על התרגיל הרטוב, ולענות עליהם בהדרגה תוך כדי פתרון התרגיל הרטוב.

1. (5 נק') מה עושה פקודת yes בלינוקס? מה הארגומנטים שהיא מקבלת?
היעזרו ב-man page, ולאחר מכן השתמשו בפקודה ב-shell שלכן כדי לבדוק.

תשובה:
הפקודה yes מדפיסה את המחזורת אותה היא מקבלת (או את המחזורת y אם לא קיבלה פרמטרים) עד שמפסיקים אותה. כארגומנט אופציונלי היא מקבלת את המחרוזת אותה היא אמורה להדפיס עד שיעצרו אותה.

2. (5 נק') מדוע השתמשנו בפקודת yes עם מחרוזת ריקה במהלך הפקודה הבאה?

```
>> yes '' | make oldconfig
```

נסו להריץ את הפקודה make oldconfig לבדה והסבירו מה הבעיה בכך.

תשובה:
המחרוזת הריקה עוזרת 'לדלג' על שלבים של ה-$make$ הדורשים קלט כלשהו מהמשתמש – הפקודה yes תשלח כל הזמן את המחזורת הריקה ו'-תקדם' את התהליך. בגלל שהפקודה כותבת ל-pipe, בעת סיום התהליך שרץ היא תקבל סיגנל SIGPIPE שברירת המחדל שלו תהיה להרוג את הפקודה.

3. (5 נק') מה משמעות הפרמטר GRUB_TIMEOUT בקובץ ההגדרות של GRUB?

```
GRUB_TIMEOUT=5
```

הסבירו מה היתרונות ומה החסרונות בהגדלת הפרמטר GRUB_TIMEOUT.

תשובה:
הפרמטר משנה את כמות הזמן שהמחשב יחכה לקלט במסך ה-$GRUB$ עד שיבחר את האפשרות הדיפולטיבית. יתרון אפשרי הוא לתת יותר זמן למשתמש לחשוב ולבחור את הimage שאותו הוא רוצה להריץ. חסרון אפשרי הוא שהמשתמש לרוב ירצה לבחור את הimage הדיפולטיבי ולכן יעדיף timeout קטן כדי שלא יצטרך לבחור בעצמו/לחכות זמן מיותר.

4. (5 נק') מדוע הפונקציה (run_init_process) אשר נמצאת בקובץ init/main.c בקוד הגרעין קוראת לפונקציה (do_execve) במקום לקריאת המערכת (execve?)

```
static int run_init_process(const char *init_filename)
{
        argv_init[0] = init_filename;
        return do_execve(getname_kernel(init_filename),
                (const char __user *const_user *)argv_init,
                (const char __user *const__user *)envp_init);
}
```

נסו להחליף את הפונקציות זו בזו ובדקו האם הגרעין מתקמפל.

תשובה:
הפקודה execve היא לשימוש המשתמש ולא מערכת ההפעלה. היא מבצעת את החלפת ההקשר ואז קוראת ל-do_execve. הקוד מעלה לא יתקמפל כאשר נבצע את ההחלפה וזאת מפני שכדי לשמור על הפרדה בין מצב משתמש למצב גרעין, execve כלל לא מוגדרת במצב גרעין. באופן דומה, לא נרצה שמשתמש חיצוני יקרא ל-do_execve כי אז החלפת ההקשר לא תבוצע ותיפגע ההפרדה בין מצב גרעין למצב משתמש.

5. (5 נק') מה עושה קריאת המערכת syscall()? כמה ארגומנטים היא מקבלת ומה תפקידם? באיזו ספריה ממומשת קריאת המערכת syscall()? היעזרו ב-man page בתשובתכן.

תשובה:

קריאת המערכת $syscall()$ מבצעת את ה-$syscall$ עם המספר שהועבר לה כפרמטר. בנוסף לכך היא תקבל את כל הארגומנטים ההכרחיים לביצוע קריאת המערכת. קריאת המערכת ממומשת ב- *GNU C LIBRARY* ומוצהרת ב-*unitstd.h*. הפונקציה מבצעת החלפת הקשר (על כל הכרוך בכך – מעבר למצב גרעין, שמירת מצב תוכנית המשתמש), קוראת לקריאת המערכת המבוקשת ומחזירה למשתמש *long int* בעת סיום הריצה שלה.

6. (5 נק') מה מדפיס הקוד הבא? האם תוכלו לכתוב קוד ברור יותר השקול לקוד הבא?

```
int main() {
    long r = syscall(57);
    printf("sys_hello returned %ld\n", r);
    return 0;
}
```

רמז: התבוננו בקובץ arch/x86/entry/syscalls/syscall_64.tbl בקוד הגרעין.

תשובה:

פלט אפשרי אחד הוא -

$sys\_hello\ returned\ 0$

$sys\_hello\ returned\ < son - pid >$

(כאשר $< son - pid >$ הוא ה-pid של הבן).

הפקודה היא $fork$ ולכן הסדר יכול להיות שונה והprintf יכולים להחתך באמצע וכו. במידה והקריאה ל-fork נכשלה, יוחזר ל-r מינוס 1 ויודפס בהתאם.

קוד שקול ויותר ברור:

```
int main()
{
    long r = fork();
    printf("sys_hello returned %ld\n", r);
    return 0;
}
```

7. (6 נק') התבוננו בתוכנית הבדיקה test1.c שסופקה לכן והסבירו במילים פשוטות מה היא בודקת:

```
int main() {
    int x = get_status();
    cout << "status: " << x << endl;
    assert(x == 0);
    x = set_status(1);
    cout << "set_status returns: " << x << endl;
    assert(x == 0);
    x = get_status();
    cout << "new status: " << x << endl;
    assert(x == 1);
    cout << "===== SUCCESS =====" << endl;
    return 0;
}
```

תשובה:
התוכנית בודקת אם הסטטוס התתחלתי הוא אכן EE (0) ושלאחר פקודת set_status השייכות התעדכנה ל-cs (1).

8. (7 נק') מה היתרונות בשמירת התהליך שמבצע register_process ברשימה בתהליך ה- init לעומת לשמור אותו ברשימה בתהליך האבא שלו? מנו לפחות שתי יתרונות.

תשובה:
יתרונות:
- אם היינו שומרים לכל אבא את רשימת הבנים ה"חשובים", יכל לקרות מצב בו האבא עובר למצב זומבי ואז במימוש הנוכחי לא תהיה לנו גישה לרשימה של הבנים ה"חשובים", אז נצטרך לעושים פעולה שכל פעם שתהליך עובר למצב זומבי – צריך להעתיק את האיברים מהרשימה שלו ל-$init$.

- סיבוכיות זמן הריצה תהיה טובה יותר אם הרשימה היא תחת init וזה מפני שבעת גישה לאיברים ברשימה נעבור רק דרך init ולא נצטרך לעבור על כל התליכים במערכת ולחפש את הרשימות השמורות אצלם.

9. (7 נק') לפי הדרישות של התרגיל, אתם נדרשים לשחרר תהליך מרשימת התהליכים החשובים ברגע שהאבא ממתין לסיומו. אם הייתם נדרשים לעשות זאת (לשחרור את התליך מהרשימה) ברגע שהוא יוצא (כלומר, מבצע את הקריאה ל- exit), איך זה היה משנה את ההתנהגות של קריאות המערכת החדשות?

תשובה:
קוד לדוגמה בו התנהגות המערכת שונה:
נבצע את שלוש הפעולות הבאות אחת אחרי השנייה – exit לתהליך מהרשימה, get_all_cs ואז לבסוף wait לתהליך שביצע exit. לפי דרושות התרגיל, התהליך אמור להיכלל בחישוב הסכום אך אם נבצע לפי הכתוב בשאלה הוא לא יחשב בסכום.

חלק 2 - זימון תהליכים (50 נק')

ברצוננו לבנות זַמן (scheduler) במערכת בעלת מעבד יחיד שבה יש חשיבות לזמן סיום של כל תהליך. כל תהליך צריך לספק למערכת את זמן הריצה הנדרש וזמן סף לסיום (deadline). האלגוריתם צריך למצוא סידור) (אם קיים), כך שכל תהליך יסתיים לפני ה-deadline שלו. הערה: בכל הסעיפים ניתן להניח שעלות החלפת הקשר היא *0*.

א. (8 נק') האם SJF יכול לשמש אותנו לצורך בניית מערכת כזו, בהנחה שכל התהליכים הגיעו בזמן 0. אם כן, הוכיחו. אם לא, תנו דוגמה נגדית.

דוגמה נגדית: נניח והגיעו תהליכים 1ו-2, כאשר 1 רץ שעתיים והדדליין שלו הוא שעתיים וחצי ותהליך 2 רץ שעה והדדליין שלו הוא 3 שעות. האלגוריתם יריץ את תהליך 2 קודם ובכך יפגע בדדליין של תהליך 1 למרות שקיים אלגוריתם שיכול לבצע אותם בסדר שלא יפגע באף אחד מהם (סדר הפוך).

Earliest Due Date - זמן שמניח שכל תהליך יכול לרוץ בלי להיחסם. בדומה ל-SJF זה אלגוריתם ללא הפקעות (no preemption) ומניח שכל התהליכים מגיעים ביחד. הוא בוחר תהליך שזמן הסף שלו קרוב ביותר.

נגדיר: C_i זה זמן סף של תהליך i ו-F_i זמן סיום בפועל. נגדיר מדד חדש:

$$L_i = \begin{cases} 0 & \text{if } F_i \le C_i \\ F_i - C_i & \text{otherwise} \end{cases}$$

$$L = \frac{1}{N}\sum L_i$$

ב. (10 נק') הוכיחו או הפריכו אופטימליות של EDD עבור קבוצת תהליכים שמגיעים ביחד ביחס למדד

זה, כלומר: הוכיחו שאלגוריתם EDD יביא למינימום של המדד L או תנו דוגמה נגדית.

דוגמה נגדית: נסתכל על רשימת התהליכים הבאה לאחר סידור ע"י EDD.

	Run time	C_i	F_i
A	3	3	3
B	5	7	8
C	2	8	10
D	2	9	12

נקבל: $L = \frac{0+(8-7)+(10-8)+(12-9)}{4} = \frac{6}{4}$

אבל אם נחליף את סידור התהליכים לסידור הבא:

	Run time	C_i	F_i
A	3	3	3
C	*2*	*8*	*5*
D	2	*9*	*7*
B	*5*	*7*	12

נקבל: $L = \frac{0+0+0+(12-7)}{4} = \frac{5}{4}$

לכן האלגוריתם לא אופטימלי.

דוגמה לתהליכים עבורם זמן הסף חשוב: תהליכים מחזוריים (תהליכים שרוצים לרוץ R שניות כל C שניות.) תהליכים כאלה חייבים לסיים את המחזור הקודם שלהם לפני תחילת מחזור הבא.

ג. (10 נק') עבור N תהליכים מחזוריים במערכת עם מעבד יחיד, שלכל תהליך יש זמן ריצה (למחזור) R_i וזמן מחזור C_i, תנו נוסחה לחישוב הנצילות של המערכת תחת זימון EDD. הסבירו את תשובתכם.

תשובה: נטען שהנוסחה לניצולת היא $\sum_{i=1}^{N}\frac{R_i}{C_i}$ וזאת מפני שאם נסדר את התהליכים לפי EDD, ונדרוש שכל תהליך ירוץ לפחות פעם אחת בזמן המחזור שלו, אזי מתוך C_i יחידות זמן התהליך ירוץ R_i יחידות זמן. באפון פורמלי יותר: נסמן $lcm = lcm(C_1, C_2, \dots, C_N)$. במשך זמן של lcm כל אחד מהתהליכים ירוץ $\frac{lcm}{C_i} * R_i$, ולכן אם נסכום את כולם יחד, בפרק הזמן של lcm (לאחריו כל התהליכים יתחילו לרוץ מחדש -האלגוריתם נותן סידור מחזורי ביחס לפרק זמן זה) נקבל שהנצילות היא: $\sum_i \frac{\frac{lcm}{C_i}*R_i}{lcm} = \sum_i \frac{R_i}{C_i}$ כמו שטענו. הנצילות של המעבד יכולה להיות גבוהה יותר אם נקדם תהליכים שיכולים לרוץ למרות שהם כבר רצו בזמן המחזור הנוכחי. בפועל ניתן להגיע לכל נצילות i, כך ש-$\sum_{i=1}^{N}\frac{R_i}{C_i} \le i \le 1$ (כאשר התהליכים באמת יכולים לרוץ גם לאחר סיום ריצתם בזמן המחזור הנוכחי – לא יצאו להמתנה, לדוגמה).

ד. (8 נק׳) עבור אילו ערכים של ניצולת אין אפשרות למצוא סידור שיספק דרישות של כל התהליכים?

עבור כל נצילות שגדולה מ-1 לא יהיה ניתן לספק את הדרישות (כי אז המעבד יצטרך לעבוד יותר ממה שהוא יכול). כמו כן, אם הנצילות תהיה קטנה מהנוסחה שמצאנו בסעיף ג', אז יהיה תהליך כלשהו שלא יספיק לסיים את ריצתו במהלך המחזור שלו (לפי איך שפיתחנו את הנוסחה).

ה. 8(נק׳) זמן **Earliest Deadline First** בשימוש נרחב במערכות RT. זה אלגוריתם עם הפקעות.
כל פעם שתהליך מסתיים או יוצא להמתנה או נוסף לתור של תהליכים מוכנים לריצה, אלגוריתם זה בוחן את כל התהליכים הזמינים ובוחר את התהליך עם זמן סף הקרוב ביותר.

נתונה מערכת עם תהליכים הבאים (שכולם הגיעו למערכת בזמן 0):

תהליך	Ri	Ci
A	1	8
B	2	6
C	4	11

על פני 20 יחידות זמן ראשונות: מה יהיה הסידור שאלגוריתם EDF יפיק?

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
B	B	A	C	C	C	C	B	B	A	NOP	C	B	B	C	C	C	A	B	B

ז. (6 נק׳) מה הניצולת של מערכת זו?

תשובה: הנצילות של המערכת היא $\frac{1}{8}+\frac{2}{6}+\frac{4}{11}=0.8219$, כלומר 82.19% (ע"פ הנוסחה שמצאנו קודם לכן).