Operating Systems – 234123

<u>Homework Exercise 2 – Dry</u>

Dan Sdeor dansdeor@campus.technion.ac.il 209509181

Levi Hurvitz levihorvitz@campus.technion.ac.il 313511602

Teaching Assistant in charge:

Niv Kaminer

Assignment Subjects & Relevant Course material

Modules, Scheduling (Lectures 4--5, Tutorials 4--5)

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 –

   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 **HW2 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 **hw2-dry**, put them in the **hw2-dry** 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/28uesBSL1P

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. (6 נק') מה עושה פקודת yes בלינוקס? מה הארגומנטים שהיא מקבלת?
היעזרו ב-man page, ולאחר מכן השתמשו בפקודה ב-shell שלכן כדי לבדוק.
תפקיד הפקודה yes הוא להדפיס את המחרוזת שמעבירים לתוכנית כארגומנט שוב ושוב עד שהיא נהרגת. אחד השימושים הבולטים בפקודה הוא להעביר את הפלט שלה כpipe לפקודה אחרת שמצפה לקבל את אותה מחרוזת. במידה ולא מעבירים לה ארגומנטים אז היא פשוט תדפיס y כל פעם בשורה חדשה שוב ושוב.

2. (6 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.
Yes עם מחרוזת ריקה יוצרת new line (מדמה לחיצה על enter). למעשה כשעשינו pipe בינה לבין make oldconfig עשינו אוטומציה של לחיצה על enter ובחירה בכל שלב של הסקריפט בחירה באופציה ברירת מחדל. בלי לעשות את זה היינו צריכים לבצע ידנית את הסקריפט ולבחור ידנית כל פעם את האופציה שנרצה שמבחינתנו היא הברירת מחדל אם אנחנו לא צריכים לעשות משהו ספציפי.

3. (6 נק') מה משמעות הפרמטר GRUB_TIMEOUT בקובץ ההגדרות של GRUB?

```
GRUB_TIMEOUT=5
```

הסבירו מה היתרונות ומה החסרונות בהגדלת הפרמטר GRUB_TIMEOUT.
פרמטר זה קובע את משך הזמן שבו תפריט הgrub מוצג למסך בתחילת הboot sequence. ככל שנגדיל את הפרמטר כך הוא יוצג יותר זמן ואז למשתמש יהיה חלון זמן גדול יותר להחליט איזה kernel (או מערכת הפעלה אחרת כמו windows עם os prober) להריץ. מצד שני החסרון הוא שכל עוד לא הייתה בחירה על ידי המשתמש תוספת הזמן שנקבעה על ידי GRUB_TIMEOUT תתווסף לזמן הboot הרגיל וזה מאריך את זמן ההגעה לריצה רגילה של המערכת.

4. (6 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.

הפונקציה רצה כבר בקוד הגרעין ולכן שימוש בקריאת מערכת הוא שגוי. קריאת מערכת היא הדרך של קוד משתמש לבקש שירותים ממערכת ההפעלה. כאשר תוכנית בuser mode קוראת ל()execve. היא בעצם קוראת לwrapper שמבצע את פקודת syscall כאשר ברגיסטר rax שמור מס השירות של קריאת מערכת. אותו מספר הוא אינדקס לטבלת syscalls ששמורה בקרנל ובאינדקס המתאים שמור בין היתר הכתובת לdo_execve. אבל מאחר וrun_init_process כבר רץ בקרנל כל מה שצריך לעשות הוא לקרוא לפונקציה עצמה ולא להשתמש במנגנון שעובר לקרנל ואז קורא לה. מצב שבו קוד קרנל מבצע syscall פקודת מכונה שמטרת לעבור לקרנל לביצוע קריאת מערכת הוא מצב לא מוגדר בלינוקס כפי שכתוב בdoc:

Do not call System Calls in the Kernel

System calls are, as stated above, interaction points between userspace and the kernel. Therefore, system call functions such as `sys_xyzzy()` or `compat_sys_xyzzy()` should only be called from userspace via the syscall table, but not from elsewhere in the kernel. If the syscall functionality is useful to be

כתוצאה משימוש בexecve במקום do_execve הקומפיילר יזרוק שגיאה כפי שקיבלנו כאן:

```
init/main.c: In function 'run_init_process':
init/main.c:954:9: error: implicit declaration of function 'execve' [-Werror=implicit-function-de
claration]
  return execve(getname_kernel(init_filename),
```

5. (6 נק') מה עושה קריאת המערכת ()syscall? כמה ארגומנטים היא מקבלת ומה תפקידם? באיזו ספריה ממומשת קריאת המערכת ()syscall? היעזרו ב-man page בתשובתכן.

קריאת המערכת ()syscall היא wrapper גנרי לקריאות מערכת. הארגומנט הראשון הוא מס' הsyscall המתאים לביצוע (מה שייכתב לתוך rax) ושאר הארגומנטים הם הארגומנטים שצריך להעביר לאותו syscall ספציפי (לכן מספר הארגומנטים שמועבר לא קבוע אבל חסום ל6 ארגומנטים). סה"כ ()syscall יכולה לקבל בין ארגומנט אחד ל7 ארגומנטים. פונקציה זו ממומשת תחת glibc (GNU Libc) של פרוייקט GNU. לרוב משתמשים בפונקציה זו כאשר אין לנו פונקציית wrapper ספציפית לsyscall שאנו מעוניינים להריץ.

6. (10 נק') מה מדפיס הקוד הבא? האם תוכלו לכתוב קוד ברור יותר השקול לקוד הבא?

```
int main() {
    long r = syscall(39);
    printf("sys_hello returned %ld\n", r);
    return 0;
}
```

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

הקוד משתמש בפונקציית syscall הגנרית כדי לעשות קריאת מערכת getpid שהיא קריאת מערכת בעלת מס' 39. הקוד ידפיס "sys_hello returned <process pid>" כאשר ב <process pid> רשום הpid של התהליך. קוד יותר מובן יראה כך:

```
int main() {
    pid_t pid = getpid();
    printf("sys_hello returned %d\n", pid);
    return 0;
}
```

7. (10 נק') התבוננו בתוכנית הבדיקה test1.c שסופקה לכן והסבירו במילים פשוטות מה היא בודקת:

```
int main() {
    int x = get_weight();
    cout << "weight: " << x << endl;
```

```
        assert(x == 0);
        x = set_weight(5);
        cout << "set_weight returns: " << x << endl;
        assert(x == 0);
        x = get_weight();
        cout << "new weight: " << x << endl;
        assert(x == 5);
        cout << "===== SUCCESS =====" << endl;
        return 0;
}
```

הקוד קורא לget_weight שקורא לקריאת המערכת שיצרנו שמחזירה את הweight של הprocess. מאחר וכל תהליך יורש מאביו את המשקל שלו וinit אותחל עם משקל 0. הציפייה שנקבל גם כאן לאחר הקריאה weight=0. לאחר מכן משתמשים בset_weight(5) שקורא לקריאת המערכת שיצרנו בקרנל שמשנה את הweight של הprocess שקרה לה ל5 ומחזירה 0 אם הכל קרה כשורה. ושוב קוראים לget_weight כדי לוודא שאכן הweight של אותו process שונה ל5. ודואגים לעשות את ההדפסות המתאימות בכל שלב.

חלק 2 - זימון תהליכים (50 נק')

1.

a. (4 נק') אילו משיטות התזמון הבאות עלולות לגרום לאפקט השיירה (convoy effect)? (שימו לב שעשויה להיות יותר מתשובה אחת נכונה)

i. FCFS
ii. RR
iii. SRTF
iv. כל מדיניות זימון עם הפקעה (preemption)
v. כל מדיניות זימון בלי הפקעה (preemption)
vi. אף תשובה אינה נכונה

ניקח לדוגמה מקרה שבו מגיע job בעל זמן ריצה ארוך מאוד ביחס לכל job אחר שיגיע לתור ההמתנה. אם אותו job הוא היחיד שנמצא בתור המתנה והמעבד מוכן להריץ job חדש, כל מדיניות זימון ללא הפקעות תבחר להריץ אותו. אם בעת הריצה יגיעו jobs בעלי זמני ריצה קטנים משמעותית, הם יאלצו לחכות לסיומו של הjob האיטי (תחת הנחה שאין ליבות אחרות שזמינות לריצה) בלי קשר לאופן הסידור של הטור או אם יהיה ניתן לעשות backfilling אחר כך בעתיד והסיבה לכך היא שלא ניתן להפקיע את הjob האיטי .גרמנו למצב שיתווסף לwaitTime של כל הjobs שמחכים הrunTime של הjob האיטי (מה שיאריך את זמן ההמתנה הממוצע) וזה בדיוק הconvoy effect. סימנו גם את FCFS כנכון כי דיברנו עליו רק בהקשר של batch scheduling.

b. (3 נק') כפי שלמדנו, תחת תנאים מסוימים אלגוריתם SJF הינו אופטימלי עבור מדד זמן תגובה ממוצע. מהם שלושת התנאים?

התנאים לכך שSJF הוא אופטימלי הם:

1. אנו רצים מעל ליבה בודדת והתהליכים רצים עליה בצורה סדרתית.
2. כל התהליכים מגיעים באותו הזמן למערכת.
3. זמני הריצה של כל התהליכים ידועים מראש.

2. (7 נק') כזכור, בתרגול ראיתן נוסחה לקצב התקדמות הזמן הווירטואלי ונוסחה לחישוב הקוונטום של כל תהליך. בסעיף זה ננסה לבחון את הקשר בין 2 הנוסחאות. הניחו כי:

a. בסוף כל epoch לכל התהליכים יש את אותו זמן וירטואלי.
b. הזמן הווירטואלי מתקדם ביחס הפוך לעדיפות התהליך: $VR_i \mathrel{+}= \frac{c}{w_i} dT$ עבור $C > 0$ כלשהו.
c. אורך ה-epoch הוא קבוע $\sum_i \quad Q_i = sched\_latency$.

הראו כי הנחות אלו גוררות את הנוסחה שראינו בתרגול לחישוב הקוונטום של כל תהליך: $Q_i = (\frac{w_i}{\sum_j \quad w_j}) \cdot sched\_latency$.

הערה: שימו לב כי הנחות אלו הן הנחות מקלות שנועדו להקל עליכן לפתור את התרגיל. הנוסחה שהוצגה בתרגול נכונה כמובן גם ללא הנחות אלו, שאינן נכונות במקרה הכללי.

בתחילת הepoch יש לתהליכים אותו זמן וירטואלי ולפי הנחה a הדבר נכון גם בסוף הepoch לכן $VR = VR_i = VR_j$ מכאן ש:

$$VR = \sum \frac{c}{w_i} dT = \frac{c}{w_i} \sum dT = \frac{c}{w_i} * Q_i$$

$$w_i = \frac{c}{VR} * Q_i => \sum w_i = \sum \frac{c}{VR} * Q_i = \frac{c}{VR} * \sum Q_i = \frac{c}{VR} * sched\_latency$$

נציב את הביטוי הראשון של שמצאנו במקום VR ונקבל:

$$\sum w_i = \frac{cw_i}{cQ_i} * sched\_latency$$

$$Q_i = \frac{w_i}{\sum w_i} * sched\_latency$$

3. (24 נק') הילה, מומחית עולמית לאלגוריתמי זימון, החליטה לנסות לשפר את אלגוריתם CFS המוכר. הילה הציעה שיפורים שונים לאלגוריתם. בכל אחד מהסעיפים הבאים, מוצע שינוי/שיפור לאלגוריתם CFS. כתבו חיסרון שנקבל כתוצאה מאותו שינוי.

a. (4 נק') תהליך שחזר מהמתנה יישאר עם אותו זמן וירטואלי שהיה לו כשהוא יצא להמתנה.
ניקח לדוגמה תהליך שמבצע פעולת IO בלולאה כך שכל פעם הוא מבצע קריאת מערכת ונכנס להמתנה עד שמתקבלת שגרה שמחזירה אותו לready. תהליך במצב כזה ינצח לרוב בתחרות מול תהליכים אחרים שלא נכנסים להמתנה כי הזמן הוירטואלי שלו לא משתנה בזמן שהזמן הוירטואלי של האחרים גדל, לכן התהליך יגרום להרעבה של שאר התהליכים.

b. (4 נק') שימוש במבנה נתונים של רשימה במקום עץ אדום-שחור.
סיבוכיות החיפוש, ההכנסה וההוצאה של תהליכים בעץ אדום-שחור היא $O(\log(n))$ ביחס לסיבוכיות חיפוש והוצאה של רשימה שהיא $O(n)$ ולכן סיבוכיות זמן הריצה של האלגוריתם תיפגע.

c. (4 נק') הסרת המינימום על גודל הקוונטום של תהליך (min_granularity)
ברגע שמספר התהליכים שלנו יהיה גדול מאוד אז גודל הקוונטום של כל תהליך יהיה קטן ביחס לעלות הבסיסית של החלפת הקשר ולכן רוב הcpu running time ילך על החלפות הקשר, כלומר התקורה שלנו נהיתה גדולה וזמן התגובה של התהליכים גדל משמעותית ולא בגללם באופן מלא.

d. (4 נק') הגדרת הקוונטום של כל תהליך, להיות בלתי תלוי במספר התהליכים, ובפרט $Q_i = a \cdot W_i$ (עבור קבוע חיובי aכלשהו)
ניקח דוגמה למצב שבו יש לנו מספר גדול של תהליכים. מאחר וכבר אין תלות של הקוונטום במספר התהליכים והוא לא קטן כתוצאה מכך. זמן ההמתנה של תהליך להריץ את הקוונטום שלו בין epoch אחד לשני גדל והתנהגות המערכת מפסיקה להיות אינטרקטיבית.

e. (4 נק') שינוי קצב התקדמות הזמן הוירטואלי להיות מהצורה $VR_i \propto \frac{1}{\sqrt{W_i}}$
המשמעות היא שקצבי ההתקדמות של תהליכים בעלי weights גדולים נהיים קרובים יותר זה לזה ביחס לנוסחת קצב ההתקדמות המקורית. זאת מכיוון שנגזרת של $\sqrt{w_i}$ שואפת ל0 ככל שw_i גדל ביחס לנגזרת של w_i שהיא 1. לכן אנחנו מאבדים מהעידון שהיה לנו בהגדרת המשקלים המקורית וכעת יכול להיות לנו מצב שבו משקלים גדולים שונים מקבלים קצב התקדמות יחסית דומה.

(4 נק') תהליכים חדשים נכנסים לתור הריצה עם זמן וירטואלי השווה לזמן הווירטואלי הממוצע של שאר התהליכים הקיימים במערכת.
הבעיה היא שזמן הווירטואלי הממוצע לא משקף את הזמן הווירטואלי של תהליכים ספציפיים בעלי זמן וירטואלי גבוה יותר. המשמעות היא שנכניס תהליכים שיקבלו תיעדוף גבוה יותר לריצה ביחס לתהליכים האלה בגלל שהזמן הוירטואלי שלהם יהיה קטן יותר ואותם תהליכים בעלי זמן וירטואלי גדול מהממוצע יורעבו.

4. (6 נק') שירי, קולגה של הילה, רצתה להפחית את התקורה של חימום מטמונים בעת החלפת הקשר. לכן היא הציעה שתהליכים שמוכנים לריצה וחיים באותו מרחב זיכרון כמו התהליך שרץ אחרון על המעבד, יקבלו עדיפות על המעבד וירוצו לפני כל התהליכים האחרים. עומר, סטודנט למערכות הפעלה, טוען שהפתרון בעייתי כי בצורה זו תהליך יכול לגנוב זמן מעבד ולהרעיב תהליכים אחרים. הסבירו כיצד.

נניח שיש לנו תהליך עם מספר חוטים גדול ממספר הליבות שעומדות לרשותנו במעבד ושאנו נמצאים במצב שבו כל ליבה מריצה חוט כלשהו של אותו התהליך. במצב שבו ליבה מחליטה לבצע החלפת הקשר. הscheduler יבחר בחוט של אותו תהליך משום שהוא חולק את אותו מרחב זיכרון של החוט שרץ לפניו (חוטים של אותו תהליך חולקים מרחב זיכרון זהה עד כדי מחסנית שונה) זאת מכיוון שקבענו מספר גדול של חוטים ממספר ליבות ולכן בטוח שיש חוט אחד זמין לריצה תמיד. לכן אנחנו נמצאים במצב שתמיד אותו תהליך רץ על כל הליבות של המעבד והדבר גורם להרעבה של תהליכים אחרים שנמנע מהם לרוץ בגלל החלטת הscheduler.

5. (6 נק') נזכיר, שבאלגוריתם CFS, לכל תהליך יש משקל שנקבע לו בהתאם לערך ה-nice שלו ($-20 \leq nice \leq 19$):

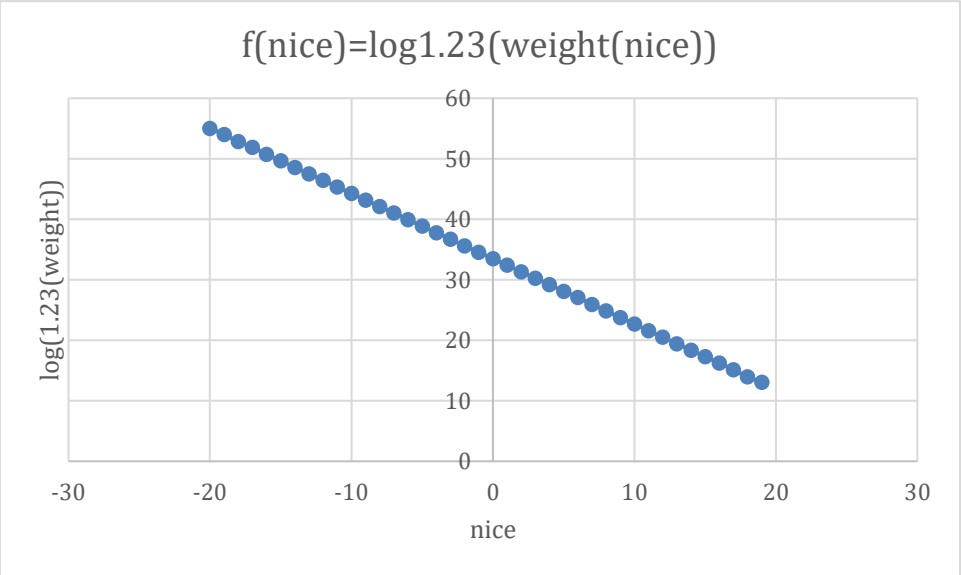
```
static const int prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```

כמו כן, ראינו כי גודל הקוונטום של תהליך, נמצא ביחס ישר למשקל התהליך:

$$Qi = \left(\frac{w_i}{\sum_j w_j}\right) \cdot sched\_latency$$

לכן, לתהליך בעל ערך nice נמוך, יש משקל גבוה, וכתוצאה מכך קוונטום ארוך יותר. בסעיף זה, נרצה לבחון את הרציונל מאחורי בחירת ערכי המשקלים.

a. (3 נק') ציירו (בעזרת אקסל או כל תוכנה אהובה לבחירתכן) גרף של לוג משקל התהליך (בבסיס 1.23), כפונקציה של ערך הnice המתאים לה. צרפו תמונה של הגרף לפתרון.



b. (3 נק') על סמך הגרף, מהו הקשר המתמטי בין ערך הnice למשקל התהליך?

$$\log_{1.23} weight(nice) = -1.0732 * nice + 33.48308$$

$$weight(nice) = 1.23^{-1.0732*nice+33.48308}$$

<u>כפי שניתן לראות מהנוסחה שקיבלנו מפירוש הגרף הקשר בין ערך ה*nice*</u>

<u>ל*weight* הוא אקספוננציאלי.</u>