Operating Systems – 234123

**Homework Exercise 2 – Dry**

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**

1. The submission should contain the following:
   1. The first page should contain the details about the submitters - Name, ID number and email address.
   2. Your answers to the dry part questions.
2. 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/YPuTiqpvQW>

# 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.
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.

השתמשנו ב yes כדי "לעשות enter" כלומר להשתמש בערכים ה default של config. בלי זה make לא הייתה עובדת כמצופה, כי בעצם make oldconfig מצפה לקבל ארגומנטים מהמשתמש אשר מתקבלים מ yes ‘’.

1. (6 נק') מה משמעות הפרמטר GRUB\_TIMEOUT בקובץ ההגדרות של GRUB?

|  |
| --- |
| GRUB\_TIMEOUT=5 |

הסבירו מה היתרונות ומה החסרונות בהגדלת הפרמטר GRUB\_TIMEOUT.

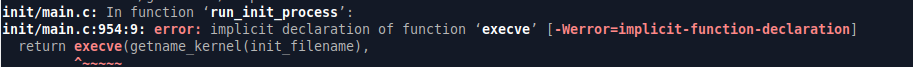
פרמטר GRUB\_TIMEOUT מציין מספר השניות לעשות boot(לטעון) ל entry (מערכת ההפעלה) הדיפולטיבית אחרי הצגת הרשימה של האפשרויות השונות שניתן לעשות להן boot. היתרונות בהגדלת הפרמטר היא שיהיה לנו יותר זמן לבחור ה entry הרצוי לפני שנבחר הדיפולטיבי(זה טוב אם יש הרבה אופציות שונות) והחסרונות בהגדלת הפרמטר היא שיקח הרבה זמו לעשות boot ל entry הדיפולטיבי בלי התערבות המשתמש.

1. (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); } | 944 945 946 947 948 949 950 |

נסו להחליף את הפונקציות זו בזו ובדקו האם הגרעין מתקמפל.

בקוד הגרעין קוראים לפונקציה do\_execve() כי בעצם היא זאת שמבצעת העבודה של execve() בפועל אשר מוגדרת במרחב של הגרעין ולעומת זאת, הפונקציה execve() אינה מוגדרת במרחב הגרעין אלא מוגדרת בספריות של libc שהוא בעצם מרחב המשתמש וכיוון שקוד הגרעין הוא בלתי תלוי בקוד המשתמש ואינו מכיר את הפונקציות שמוגדרות שם (כי למשל אם נסמוך על המשתמש לבצע מה שהוא רוצה במשתנים של הגרעין ייתכנו צרות באבטחה) מבצעים קריאה לפונקציה do\_execve() בתוך הגרעין ולא לפונקצית משתמש. ולכן כאשר מחליפים הפונקציות הקוד לא מתקמפל ומקבלים השגיאה הבאה:



1. (6 נק') מה עושה קריאת המערכת ()syscall? כמה ארגומנטים היא מקבלת ומה תפקידם? באיזו ספריה ממומשת קריאת המערכת ()syscall? היעזרו ב-man page בתשובתכן.

פקודת syscall מבצעת את קריאת המערכת אשר היא מקבלת את מספרה בפרמטר הראשון שלה, היא מצפה לקבל לפחות פרמטר אחד, הפרמטר הראשון מציין מספר קריאת המערכת שרוצים לבצע ואם קריאת המערכת הספציפית מצפה לעוד פרמטרים אז צריך להכניס אותם אחרי מספר קריאת המערכת (מספר הפרמטרים שונה בין קריאות המערכת השונות). Syscall() ממומשת בתוך ספריית glibc ומשתמשת ב h.sys/syscall וגם משתמשת במידע מ unistd.h.

1. (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.

Syscall(39) הינה בעצם קריאה ל getpid() ןלכן נקבל את ה pid של ה process שמבצע main() ולכן כדי להפוך הקוד ליותר ברור נחליף syscall(39) ב getpid(). נסמן ב RET\_PID ה pid שמוחזר מ getpid(), ובעצם יודפס : “sys\_hello returned RET\_PID\n” \n כדי לציין ירידת שורה(לא מודפס \n כמחרוזת).

1. (10 נק') התבוננו בתוכנית הבדיקה test1.c שסופקה לכן והסבירו במילים פשוטות מה היא בודקת:

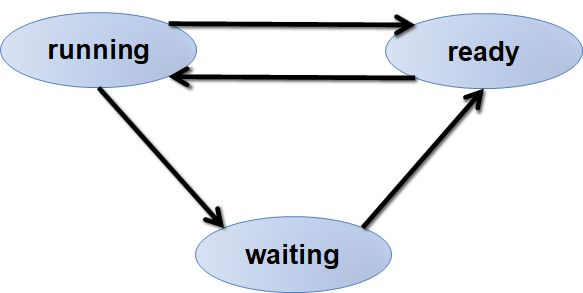
|  |
| --- |
| int main**()** **{**  int x **=** get\_weight**();**  cout **<<** "weight: " **<<** x **<<** endl**;**  assert**(**x **==** 0**);**  x **=** set\_weight**(**5**);**  cout **<<** "set\_weight returns: " **<<** x **<<** endl**;**  assert**(**x **==** 0**);**  x **=** get\_weight**();**  cout **<<** "new weight: " **<<** x **<<** endl**;**  assert**(**x **==** 5**);**  cout **<<** "===== SUCCESS =====" **<<** endl**;**  **return** 0**;**  **}** |

התוכנית מבצעת קריאה לקירות המערכת get\_weight() אשר מקבילה ל syscall(335) וקריאת המערכת set\_weight(weight) אשר מקבילה ל syscall(334,weight) וה wrapper\_function שלהן בתוך hw2\_test.c ולאחר קריאות המערכת, מוודאים שהוחזר הערך הנכון (המצופה שיוחזר) ואם עוברים כל הבדיקות אז מדפיסים "===== SUCCESS =====" בסוף. אחרת, נופלים ב assert.  
לסיכום, קוראים ל get\_weight() ומאתחלים את ערך ההחזרה שלו ב x ומוודאים שהוחזר 0. ואז משנים את ה weight ל 5 ומוודאים שהפונקציה הצליחה, כלומר החזירה 0. ולאחר מכן קוראים ל get\_weight() שוב כדי שהפונקציה הקודמת הצליחה באמת לשנות את weight ל 5 ואם מוודאים את זה אז יודפס SUCCESS.

# חלק 2 - זימון תהליכים (50 נק')

**נא לנמק את תשובותיכם לכל הסעיפים**

1. נתון התרשים המופשט של מצבי התהליך:



עבור כל מעבר תנו תרחיש המוביל לאותו מעבר:

* 1. running→ready

אם התהליך שרץ כרגע (RUNNING) סיים את ה QUANTUM שלו אך עוד לא סיים לרוץ ועכשיו התור של תהליך אחר לרוץ, הוא יעבור למצב READY.

* 1. ready→running

אם יש תהליך שמוכן לריצה (לא ממתין לאף דבר) כלומר כרגע במצב READY והזמן תהליכים משבץ אותו לרוץ עכשיו אז הוא יעבור למצב RUNNING.

* 1. running→waiting

אם התהליך שרץ כרגע (RUNNING) מבצע פעולת I/O הוא יצא להמתנה (WAITING).

* 1. waiting→ready

אם יש תהליך שביצע פעולת I/O ויצא להמתנה כלומר כרגע הוא במצב WAITING, ופעולה זאת שהוא ביקש הסתיימה עכשיו אז הוא יעבור למצב READY.

1. נתון שהמערכת עובדת עם זמן תהליך מסוג RR (round robin):
   1. מה היתרון בשימוש ב quantum גדול?

היתרון של quantum גדול הוא שנבזבז פחות זמן על החלפות הקשר של המערכת, תוכניות חישוביות אשר צריכות זמן חישוב גדול יהיה להן מספיק שמן ריצה כדי לבצע את החישובים שלהם.

* 1. מה היתרון בשימוש ב quantum קטן?

היתרון של quantum קטן הוא שהתוכניות יהיו יותר שזמן ההמתנה לריצה הבאב של התוכנית היא נמוכה, ולכן תוכניות שהן I/O bound כלומר אינטיראקטיביות יעדיפו quantum כזה.

* 1. במידה והמערכת עמוסה (מכילה הרבה תהליכים מוכנים לריצה), מדוע עדיף להוסיף תהליכים חדשים בסוף התור?

עדיף להוסיף תהליכים חדשים בסוף התור כדי שלא ירעיבו את שאר התהליכים הרצים במערכת. כי ייתכן תרחיש שכל הזמן מוסיפים תהליך חדש לפני שעוברים לתהליך הבא לביצוע ואז לעולם לא נגיע לתהליכים שהיו במערכת ונרעיב אותן.

1. בזמן תהליכים (CFS (completely fair scheduler, איזו בעיה פותרת ה min\_granularity?

ה min\_granularity פותרת לנו הבעיה כאשר זמן quantum קטן מדי גורר הרבה החלפות הקשר במערכת ורוב זמן המערכת מבוזבז על ביצוע החלפות הקשר. בשביל כך הוגדר min\_granularity אשר מבטיח שלא נרד מתחת לסף שלה ובזה הוא נותן חסם על כמה אנחנו מוכנים לבזבז על זמני החלפות הקשר.

1. במערכת עם ליבה אחת, בה כל התהליכים מגיעים יחד וזמני הריצה שלהם ידועים מראש. איזה אלגוריתם batch scheduling (כלומר בלי הפקעות תהליכים) ימזער את ה- average response time (זמן התגובה הממוצע)?
   1. RR (round robin) algorithm
   2. FCFS (first come first serve) algorithm
   3. SJF (shortest job first) algorithm
   4. EASY (FCFS + back-filling) algorithm

ראינו בכיתה שאלגוריתם SJF הינו האופטימלי מבחינת average response time מבין אלגוריתמי batch scheduling כאשר יש לנו ליבה אחת.

1. נגדיר מערכת בעלת 3 ליבות (המסוגלות להריץ תהליכים במקביל בהתאם לדרישות של התהליכים), בה ישנם אך ורק 4 תהליכים המעוניינים לרוץ:

תהליך 1 דורש 2 ליבות וירוץ למשך 2 שניות עד לסיום.

תהליך 2 דורש 1 ליבות וירוץ למשך 3 שניות עד לסיום.

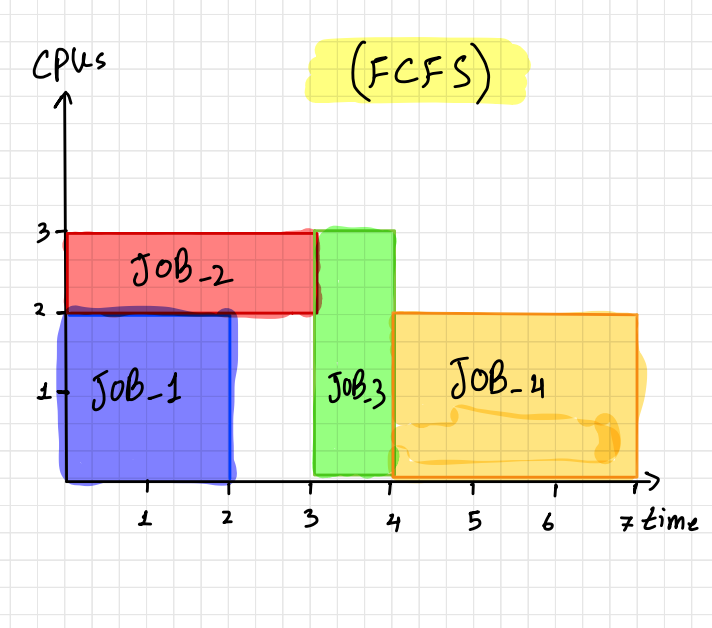
תהליך 3 דורש 3 ליבות וירוץ למשך 1 שניות עד לסיום.

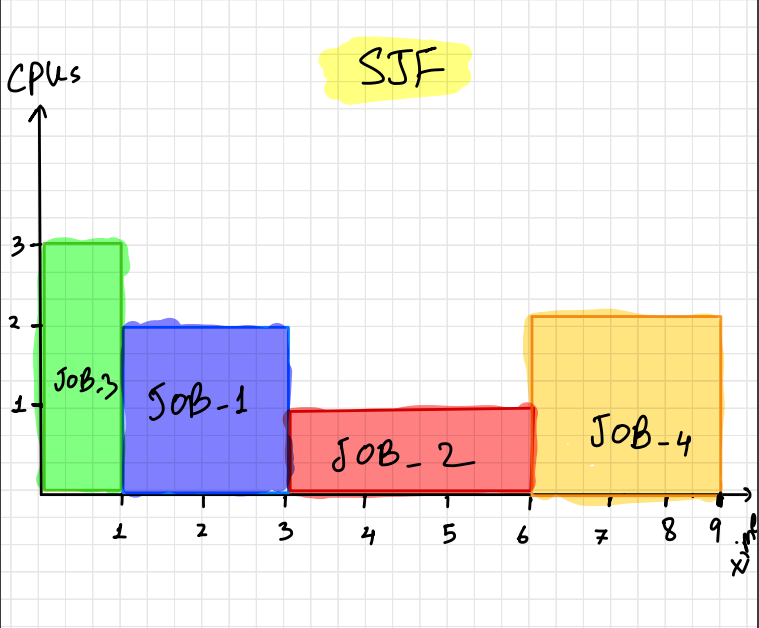
תהליך 4 דורש 2 ליבות וירוץ למשך 3 שניות עד לסיום.

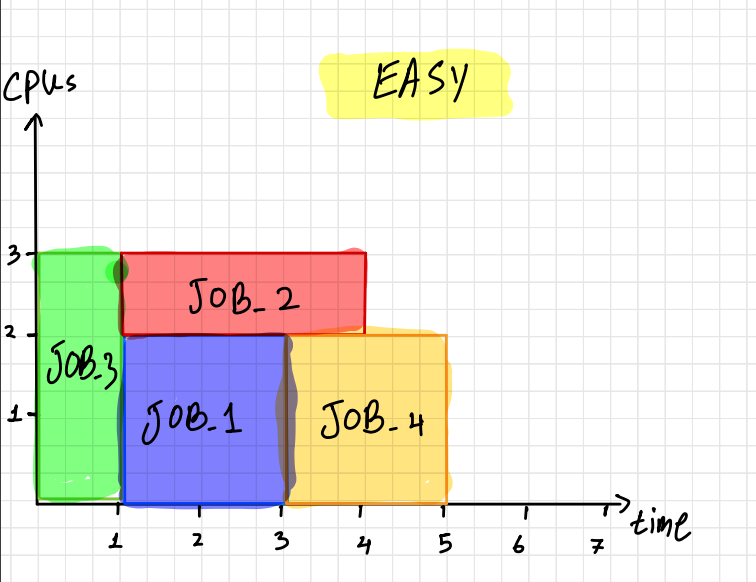
התהליכים נשלחים למעבד בסדר זה. איזה אלגוריתם יגרום לסיום כל התהליכים ראשון? נמקו.

* 1. FCFS
  2. SJF
  3. EASY
  4. תשובות b ו c נכונות.

נצייר תרשים לשיבוץ של התהליכים לפי כל מדיניות ונבחן איזו מדיניות עדיפה:







ולפי התרשימים שהתקבלו לכל אחד מהזמנים נקבל ש EASY הוא זה שיגרום לסיום כל התהליכים ראשון.

1. במערכת בה תהליכים מגיעים בזמנים שרירותיים, באיזה אלגוריתם תזמון נעדיף להשתמש – SRTF או SJF כדי לקבל את ה- average response time (זמן התגובה הממוצע) הקטן ביותר? מדוע?

נעדיף להשתמש ב SRTF כיוון שהתהליכים מגיעים בזמנים שרירותיים. SJF הוא אלגוריתם batch scheduling אשר מניח שהוא יודע זמן ריצה של כל התהליכים ושכל התהליכים הגיעו ביחד אך כפי שמתואר במערכת שבשאלה התהליכים לא מגיעים ביחד אלא בזמנים שרירותיים.