Operating Systems – 234123

**Homework Exercise 2 – Dry**

**מגישים:**

**נועה פריאנטה 206200305**

[pariente@campus.technion.ac.il](mailto:pariente@campus.technion.ac.il)

**ששון שמואל למעי 325172351**

sason-shmuel@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. (6 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 מקבלת מחרוזת כארגומנט ומדפיסה את המחרוזת אינסוף פעמים (כל פעם בשורה חדשה) עד שהתהליך נעצר (למשל על ידי שליחת סיגנל להרוג ctrl+C). אם אינה מקבלת ארגומנט הינה מדפיסה 'y' אינסוף פעמים עד שהתהליך נעצר.

1. (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.

תשובה: כאשר הרצנו make oldconfig לבד, הפקודה לא רצה מכיוון שהיא אמרה שהיא מצפה לארגומנטים. כאשר אנו משתמשים ב- yes ' 'כפי שציינו הוא מדפיס אינסוף פעמים את המחרוזת ' ' על ידי שימוש ב-pipe (מסומן ב-|) אני אומרים לפקודה yes שהערוץ פלט שלה הינה הפקודה לאחר מכן, כלומר make oldconfig ובכך אנו מעבירים ארגומנטים לפקודה.

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

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

הסבירו מה היתרונות ומה החסרונות בהגדלת הפרמטר GRUB\_TIMEOUT.

תשובה: GRUB\_TIMEOUT זה משך הזמן (בשניות) המוצג על המסך את התפריט של GRUB (מאפשר לבחור באיזה מערכת הפעלה וכו' להשתמש) לפני שהמערכת מאתחלת אוטומטית למה שמוגדר כברירת מחדל שלה.

יתרונות: נותן זה לבחור איזה מערכת הפעלה לעלות

חסרונות: מאט את עליית מערכת ההפעלה

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.

תשובה: ראשית כדי לענות את השאלה נציין מה ההבדל בין שתי הפונקציות:

()execve הינה פונקציית POSIX קריאת מערכת לשימוש של תוכניות C במצב userspace

()do\_execve הינה פונקציית גרעין, כלומר ניתנת להרצה רק ברמת הרשאהkernel

()run\_init\_process הינה פונקציה בקוד הגרעין (אחראית ליצירת תהליך הinit-

של מערכת ההפעלה) ולכן רצה בהרשאות הגרעין. היא אינה מכירה את הספרייה של posix אשר הינה ספריית שירות לקריאות מערכת לתוכניות חיצוניות לגרעין ולכן היא מריצה ()do\_execve ואם היא תנסה להריץ ()execve נקבל שגיאת קומפילציה:

implicit declaration of function ‘execve’

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.
2. (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.

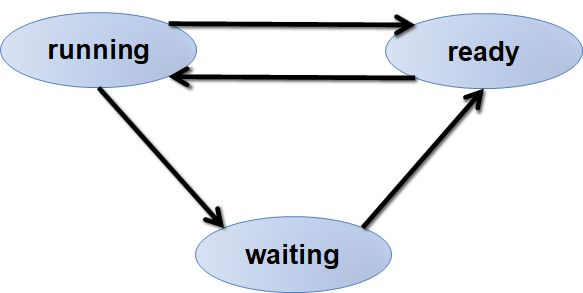
1. (10 נק') התבוננו בתוכנית הבדיקה test1.c שסופקה לכן והסבירו במילים פשוטות מה היא בודקת:

|  |
| --- |
| 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**;**  **}** |

# 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

runningכאשר התהליך רץ על המעבד (המחסנית שלו טעונה( והוא עובר ל-ready כאשר מערכת ההפעלה מעבירה אותו לשם על ידי preemptive scheduler כמו למשל RR

* 1. ready→running

ready זה המצב ממנו התנליך מתחיל שהוא מסיים אתחול, כלומר נוצר התהליך והוא עובר ל-runing שהוא מקבל זמן מעבד.

* 1. running→waiting

runningכאשר התהליך רץ על המעבד והוא ביקש ממערכת ההפעלה בקשת I\O ולכן מערכת ההפעלה העבירה אותו למצב waiting

* 1. waiting→ready

תהליך הוא המצב waiting כאשר הוא בהמתנה לאירוע. כאשר ה-event או בקשת ה-I\O שהתהליך חיכה לו הושלם הוא חוזר למצב ready כי הוא מוכן לחזור לרוץ.

1. נתון שהמערכת עובדת עם זמן תהליך מסוג RR (round robin):
   1. מה היתרון בשימוש ב quantum גדול?
   2. מה היתרון בשימוש ב quantum קטן?
   3. במידה והמערכת עמוסה (מכילה הרבה תהליכים מוכנים לריצה), מדוע עדיף להוסיף תהליכים חדשים בסוף התור?
2. בזמן תהליכים (CFS (completely fair scheduler, איזו בעיה פותרת ה min\_granularity?
3. 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
4. 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.

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?