

ארגון ותכונות המחשב

תרגיל בית 3 (רטוב)

המתרגל האחראי על התרגיל: בועז מואב.

הנחיות:

- שאלות על התרגיל ב- Piazza בלבד.
- הגשתה בזוגות.
- על כל יום איחור או חלק ממנו, שאינם באישור מראש, יורדו 5 נקודות.
 - ניתן לאחר ב-3 ימים לכל היתר.
 - הגישות באיחור יתבצעו דרך אתר הקורס.
- את התרגיל יש להגיש באתר הקורס בקובץ zip.
- תיקונים לתרגיל, אם יהיו, יופיעו ממורכרים.

חלק א – שגרות, קונבנציות ומה שביניהם (80 נק')

בחלק א' של תרגיל הבית נமמש [parser](#) פשוט, אשר יקרא קובץ שמחולק לשורות ויסיק מספר מסקנות לגבי התוכן.

פונקציות לימוש בתרגיל

בעת נספיק הסברים על הפונקציות שתתמשו בעצמכם באסמלבי. מותר ואף מומלץ להשתמש בפונקציה שכבר מימשTEM בטור פונקציית עדר לימוש הפונקציה השנייה, אך עם זאת חשוב לשים לב שלכל פונקציה תיבדק בפני עצמה.

`compute_char_repeats`

חתימת הפונקציה הראשונה לימוש:

```
int compute_char_repeats(char* buffer, int line_length, char special_char);
```

קלט: `buffer` – מצביע לתחילת מערך תוים באורך לא ידוע. `line_length` – כמות התווים שיש לקרוא מהערך.
`special_char` –תו שעלייכם לספר כמה פעמים הוא מופיע.

פלט: עלייכם להחזיר כמה פעמים מופיע `special_char` ב-`line_length` התווים הראשונים של המערך `buffer`.
שימוש: `line_length` יכול להיות 0 ואך יש להחזיר 0.

הנחות: `buffer` לא מצביע ל-NULL ובאופן כללי תמיד חוקי ואין צורך לבדוק זאת, `line_length` לא יהיה מספר שלילי ותמיד יהיה קטן או שווה לגודל הזיכרון שהוקצתה ל-`buffer`.

`parse_lines`

חתימת הפונקציה השנייה לימוש:

```
int parse_lines(char* path, int* line_max_len, int* line_max_repeat);
```

קלט: `path` – מחרוזת של הכתוב אל קובץ הקולט. `line_max_repeat`-`line_max_len` הם למעשה חלק מהקלט ונסביר עליהם שם, כי עלייכם לכתבם עצמם ער.

פלט: הפונקציה תחזיר בערך החזרה שלה את כמות השורות בקובץ שנית ב-`path` ובנוסף תמלא את המצביע `line_max_len` עם אורך השורה הארוכה ביותר בקובץ **את** `line_max_repeat` עם כמות הפעמים שהזרה האות המינוחית (עליה נסביר בהמשך) בשורה בה חזרה במתו מקסימלית של פעמים.

שימוש: שורה חדשה מתחילה לאחר התו 'newline', והטו עצמו אין נספר בחלק מאורך שורה. קובץ ריק מכיל שורה אחת ריקה. עלייכם לקבל את האות המינוחית מהפונקציה `get_the_special_char` עליה מוסבר בהמשך העמוד. יש לסגור כל קובץ שפותחים.

הנחות: `path` חוקי ומהווע מקום של קובץ בעל הרשאות קרייה. `len`-`line_max_repeat`-`line_max_len` הם מצביעים חוקיים ואין צורך לבדוק זאת. אורך שורה בקובץ לא יעלה על 128 תוים (כולל תו ירידת שורה). כמות השורות אינה חורגת מגודל של `int` ואין צורך לבדוק זאת.

לוגנה

נכיה שהתקבל קובץ הקולט הבא:

I love ATAM

כלומר, קובץ עם 2 שורות, הראשונה באורך 11 והשנייה באורך 1. וכnicח ש-`get_the_special_char` החזירה את התו 'A' איז הפונקציה תחזיר 2, תמלא את `line_max_len` ב-11 ואת `line_max_repeat` ב-2.

פונקציית עזר נתונות

לתרגיל זה נתונה פונקציית העזר `get_the_special_char` שחתימתה היא:

```
char get_the_special_char();
```

הפונקציה מחזירה את האות המינוחית אותה צריך בפונקציה `parse_lines`. התו לא יכול להיות 'newline'.
יש ל לקרוא לפונקציית העזר פעם אחת בלבד **בכל ריצה של !parse_lines**

דרישות מימוש

את כל המימושים תשלימו בקובץ S.students_code. קובץ S הוא קובץ אסמבלי, המתאים ל-gcc. אין הבדל אמיתי בין לבין קובץ `asm`¹ (ויש שיטענו, בצדק, שדוקא קבצי S מתאימים יותר מ-asm לקורס). יש לכתוב את קוד האסמבלי בעצמכם.

הערות חשובות לגבי מימוש התרגילים:

1. קונבנציות!!! בתרגיל זה נכתבו קוד שמשלב אסמבלי -C. קוד שלא יכתב לפוי System, יכשל בטסטים.
2. אסור לכם להוסיף קבצים בלבד S.students_code. בפרט לא את `o.3w3h.aux`.
3. אסור לכם להוסיף משתנים ל-data section.
- a. אם ברצונכם להשתמש במשתנים מקומיים, אתם מבונים יכולם (ואף מומלץ לעשות זאת) – אך תצטרכו להשתמש זאת לפי הקונבנציות שנלמדו בקורס.
4. **מומלץ להיעזר ב-GDB בעת דיבוג הקוד.** מדריך לשימוש בדיבגר GDB זמין באתר הקורס. ניתן להיעזר במקרה לדיבוג אם תרצו, אך מוצג שאסור להשתמש בהם לכתיבת הקוד עצמו.
5. אנחנו מקמלים את הקבצים שלכם בצורה שגוררת עליהם את ההגילה הבאה – אסור להשתמש בשיטת מיוען אבסולוטית, או בקבוע שהוא כתובות ון אסור להשתמש באך פונקציית ספירה של C. בפרט, בשאותם יוצרים לעצמכם טסטים **אסור להוסיף את הדגלים `pie` או `no-pie`** לשורת הקימפוף בהמשך הקורס נלמד על הדגלים האלה ובנין למה הוספה הדגים קשורה לאיסור המדבר).
6. أنا ודואו שהתוכניות שלכם יוצאות (מסתיימת) באופן תקין, דרך `main` של קובץ הבדיקה שהוואר לפונקציה `syscall`, ולא על ידי `exit` שלהם, במקרה שבו השתמשתם באחד (וכמובן שגם לא בעקבות קירסת הקוד). הערה זו נכתבה בדם ביטים (של קוד של סטודנטים מסוימים קודמים).
- על מנת לוודא את ערך החזזה של התוכנית, תוכלו להשתמש בפקודת `bash`: `$? echo` (הזכורה: ערך החזזה של התוכנית, במידה ויצאה בצורה תקינה, הוא הערך ש-`main` מחזירה ב-`return` האחרון שלה).
- כתבו קוד סביר (מבחינת יעילות). קוד לא יעיל בצורה חריגה עלול להיכשל בטסטים.
- אם הכל עובד בשורה, אתם יכולים לעבור לחילק ב' של תרגיל הבית, ולאחריו לחילק ג', שהוא בסך הכל הוראות הגשה לתרגיל כלו (שים לב שאתם מגישים את שני החלקים יחד!).

¹ <https://stackoverflow.com/questions/34098596/assembly-files-difference-between-a-s-asm>

חלק ב – פסיקות (20 נק')

מבוא

לפני שאתם מתחילה את חלק זה, ארכנו ממליצים לכם לחתור על תרגול וסדנה 6 ולראות שאתם יודעים לענות על השאלות הבאות – מה זה TID? מהן הפוקודות sidt ו-lidt? מהן מכילות כניסה-OUT? מהי שגרת טיפול בפסיקה? באילו הרשותת היא רצתה? מהי חלקת האחוריות בין שגרת הטיפול לבין המעבד? באיזו מהנסנית משתמש? איך היא נראה בכל שלב? במה זה תלוי? בעת, קראו את כל השלבים בחלק זה לפניהם שתתחלו לעבוד על הקוד.

בתרגילים זה נרצה לכתבו שגרת טיפול בפסיקת המעבד המתבצעת כאשר מבצעים פקודה לא חוקית (כלומר, כאשר המעבד מקבל opcode שאינו מוגדר בו).

- כאשר המעבד מקבל קידוד פקודה שאינו חוקי, המעבד קורא לשגרת הטיפול בפסיקה ב-TID.
- שגרת הטיפול בlynokס שולחת SIGSEGV לתובנית שביצעה את הפקודה הלא חוקית. למשל:

<https://github.com/torvalds/linux/blob/16f73eb02d7e1765ccab3d2018e0bd98eb93d973/arch/x86/kernel/traps.c#L321>

נרצה לשנות את קוד הkernel כך ששגרת הטיפול בפסיקה תשתנה (שאלה למחשבה – למה חייבים לשנות את קוד הkernel?) ונעשה זאת באמצעות kernel module.

מה תבצע שגרת הטיפול החדש?

שגרת הטיפול בפסיקה שלנו (שאותה אתם הולכים למשב באסמלבי בעצמכם, בקובץ `asm_handler.asm` או), תיקרא my_ili_handler ותבצע את הדברים הבאים:

1. בדיקת הפקודה שהובילה לפסיקה זו. הנחות:
 - הניחו כי הפקודה השגיה היא פקודה של אופקוד בלבד. ככלומר, לפני ואחריו opcode השגוי אין עוד ביטים (אין prefix legacy, אין REX).
 - لكن, אורך הפקודה השגיה הוא באורך 3 bytes. בתרגיל זה הניחו כי אורך האופקוד השगוי הוא לפחות 2 ביטים.
2. קריאה לפונקציה `do_what_to` עם ה-`byte` האחרון של האופקוד הלא חוקי, כפרמטר.
 - היזכרו בחומר של קידוד פוקודות:
 - אם האופקוד אינו מתחיל ב-0x0, הוא באורך byte אחד.
 - אחרת (בן מתחיל ב-0x0), אם הוא אינו מתחיל ב-A-0x0F38 או 0x0F38 או 0x0, אז הוא באורך 2 ביטים. לבן, הניחו כי הביט השני באופקוד אינו 0x38 או 0x0 (אין צורך לבדוק זאת).
 - דוגמאות:
 - עבור האופקוד 0x27, שהינה פקודה לא חוקית בארכיטקטורת 64-68x, נבצע קריאה ל-`what_to_do` עם `0x27`.
 - עבור האופקוד 0x04, גם לא חוקית, נבצע קריאה ל-`what_to_do` עם הפרמטר 0x04.
3. בדיקת ערך החזרה של `do_what_to`
 - אם הוא אינו 0 – חרזה מהפסיקה, ברשימתוונת תוכל להמשך לרוח (תציגו לפקודה הבאה לביצוע מיד לאחר הפקודה הסוררת) וערכו של גיסטר rd%invalid הינה פסיקה מסוג `fault`. חשבו מה זה אומר על ערכו של גיסטר rd%invalid בעית החזרה משגרת הטיפול ושנו אותו בהתאם.
 - שים לב #1: שימו לב ש-`opcode`-rd%invalid הינה פסיקה מסוג `fault`. המדריך על הפסיקה שלנו, במקרה ² שימו לב #1: היעזרו בספר אינטלי 3 volume 3, עמוד 223, המדריך על הפסיקה שלנו, ב כדי לוודא את תשובהיכם ל"שים לב #1" וגם כדי להחליט האם יש או לא.
 - שים לב #2: שימו לב #2: היעזרו בספר אינטלי 3 volume 3, עמוד 223, המדריך על הפסיקה שלנו, ב כדי לוודא את תשובהיכם ל"שים לב #1" וגם כדי להחליט האם יש או לא.
 - שים לב #3: שימו לב #3: הינה שגרה שתיתנת על זמן הבדיקה. אין להניח לגיביה דבר, מלבד חתימתה (בלומר – שם השגרה, טיפוס פרמטר הקלט וטיפוס ערך החזרה).
 - אחרת (הוא 0) – העברת השליטה לשגרת הטיפול המקורי.

² ב"עולם האמיתי" אסור לשנות ערכים של גיסטרים וצריך להחזיר את מצב התוכנית כפי שקיבלתם אותו. בגיןכם מדרשים כן לשנות ערך של גיסטר, אך שמצוות התוכנית לא יהיה כפי שהוא בשחרור הפסיקה. זה בסדר, זה לצורך התרגיל ☺
<https://software.intel.com/content/dam/develop/external/us/en/documents-tps/325384-sdm-vol-3abcd.pdf>³

לפני תחילת העבודה – מה קיבלתם?

בתרגיל זה תעבדו על מכונה וירטואלית דרך `Qemu` (בתוך המכונה הווירטואלית - VirtualMachine). על המכונה זו, אנחנו נירץ⁴ kernel module שיבצע את החלתת שגרת הטיפול לו שמיימותם בעצמכם. היהת והקוד רץ ב-`kernel mode` (0), במקרה של תקלת מערכת הפעלה תקרו. אך זה לא נראה! עלייכם פשוט להפעיל את `Qemu` מחדש.

לרשומכם נמצא הקובץ הבא בתיקייה 2 part:

- `initial_setup.sh`
 - הריצו סקורייפט זה לפני כל דבר אחר. סקורייפט זה מכין את המכונה הווירטואלית לריצת `Qemu`. עליכם להריץ אותו פעם אחת בלבד (לא יקרה כלום אם תריצו יותר, אך זה לא נחוץ).
 - יכול להיות שתצטרבו להריץ את הפקודה הבאה, לפני ההרצה, בগল בעית הרשאות:
- `chmod +x initial_setup.sh`
- `compile.sh` - הריצו סקורייפט זה בכל פעם שתרצו לкомpile את הקוד ולתען אותו (עם המודול המקורי) למכונה הווירטואלית של `Qemu` (ישמו לב: עליכם לצאת מ-`Qemu` קודם).
 - גם כאן יתכן ותזדקקו להרצה של `chmod` באותו אופן כמו בסעיף הקודם.
- `start.sh` - הריצו סקורייפט זה כדי להפעיל את המכונה הווירטואלית של `Qemu`, לאחר שקיימפלתם את תיקיית `code` ועונתם אותה אל המכונה הווירטואלית של `Qemu`.
 - גם כאן יתכן ותזדקקו להרצה של `chmod` באותו אופן כמו בסעיף הקודם.
- `filesystem.img` - המכונה הווירטואלית אותה תריצו ב-`Qemu`.
- `makefile` - קבצי הקוד שנכתבו, חלק מהמודול (והיא זו שתקומפל ותורץ לבסוף ב-`Qemu`) וה-`ili_handler.asm`, `ili_main.c`, `ili_utils.c`, `inst_test.c`, `Makefile`

איך הכל מתחבר – כתיבת המודול

בתיקייה `code` סיפקנו לכם מספר קבצים:

- `inst_test.c` – simple code example that executes invalid opcode. Use it for basic testing.
- `ili_main.c` – initialize the kernel module – provided to you for testing.
- `ili_utils.c` – implementation of `ili_main`'s functionality – **YOUR JOB TO FILL**
- `ili_handler.asm` – exception handling in assembly – **YOUR JOB TO FILL**
- `Makefile` – commands to build the kernel module and `inst_test`.

ממשו את הפונקציות ב-`c.ili_utils`, כך שהשגרה ב-`c.ili_handler` יתירה כאשר מנוטים לבצע פקודה לא חוקית. אורך? Well, זהה ל-ב התרגיל, אך נסו להזכיר בחומר הקורס. כיצד נקבעת השגרה שנתקראת בעת פסיקה? פועל בהתאם. שימו לב כי ב-`c.ili_utils` אנו רוצים לגשת לחומרה. מהי הדרכך לשנות קוד כאשר אנו רוצים לבצע פקודות ב-low level? לאחר מכן, ממשו את הפונקציה `my_ili_handler` ב-`asm.ili_handler` שתשבע את מה שהוגדר בשלב השני.

⁴ למי שלא מכיר את המונח `kernel module`, ביל' פאניקה (פ' `panic` זה רע, אבל זה עוד יותר רע בקורס. פאניקה! בדיסקו זה דוחוק באסדר) – מדובר בדרך להוסיף קוד בזמן ריצה (ניתן להוסיף לkernel קוד ולקומפל לאחר מכן את כל kernel החדש, אך כאן לא הזמן ולא המקום לה). למעשה, נכתב קוד שירוץ ב-`kernel mode` ולבן והוא בעל הרשאות מלאות. אנו מדרשים לך – הרי אם רוצים לשנות את קוד הkernel.

זמן בדיקות - הרצה המודול

לאחר שסיימתם לבנות את המודול, בצעו את השלבים הבאים:

1. הריצו את `compile.sh`. כדי לкопיא את קוד הkernel ולהכיןו למוכנת-hEMU QEMU
2. הריצו את `start.sh`. כדי לפתח מוכנה פנימית באמצעות UQEMU
 - a. התעלמו מכל המל שיפוי, כולל הערות בצבעים אדומים וירוקים. זה תקין.
 - b. לאחר העילה – משתמש: root, סיסמה: root. בעת אתם בתוך hEMU וכל השלבים הבאים מתיחסים לריצה hEMU.
3. באתם בתוכה hEMU. כדי להריץ את הקוד `inst_test.asm`, עם הפקודה הלא חוקית (ולקבל הודעה שגיאה בהתאם). ניתן גם להריץ את `inst_test_2.asm`. כדי להריץ את הקוד ב-`asm`.
4. נאנו גם לטעון את המודול שלכם (ודאו שהוא נתען ע"י הרצת `dmesg`).
5. הריצו `insmod ili.ko` כדי לקבל התנהגות שונה, מכיוון שהפעם ה-`handler` שלכם נקבע.

שיםו לב שלאחר ביצוע שלב 1 קיבלם את האזהרה הבאה:

```
root@ubuntu18:~# ./modules
WARNING: could not find /home/student/Documents/atam2/part2/.ili_handler.o.cmd
or /home/student/Documents/atam2/part2/ili_handler.o
CC      /home/student/Documents/atam2/part2/ili_mod.o
```

יש להתעלם מזהירה זו. שגיאות אחרות עלולות להציגן על בעיה מהותית ואף לא יצירת הקבצים הרלוונטיים (בעיות בנייה, בעיות קישור ועוד), لكن שימו לב מהן העורחות שימושיות בעת הרצה `compile`. גם בעית עליית hEMU יוחז הרבה טקסט במסך, בצבעים משתנים של אדום, ירוק וצהוב. לא להיבהל, זה תקין.

דוגמת הרצה תקינה ב-hEMU (עם הטסיטים `what_to_do_inst_test_2` ו-`inst_test`) שסופק לכם כדוגמה:

```
root@ubuntu18:~# ./bad_inst
start
Illegal instruction
root@ubuntu18:~# insmod ili.ko
root@ubuntu18:~# ./bad_inst
start
root@ubuntu18:~# echo $?
35
root@ubuntu18:~# rmmod ili.ko
rmmod: ERROR: ../libkmod/libkmod.c:514 lookup_builtin_file() could not open built-in file '/lib/modules/4.15.0-60-generic/modules.builtin.bin'
root@ubuntu18:~# ./bad_inst_2
start
Illegal instruction
root@ubuntu18:~# insmod ili.ko
root@ubuntu18:~# ./bad_inst_2
start
Illegal instruction
```

(`what_to_do` ממחירה את הקלט שלו פחות 4. בטעט הראשון הפקודה הלא חוקית היא `0x27`, אך ערך החזרה הוא `0x23`, שזה 35. ערך זה גם ערך היציאה של התוכנית, כי כך נכתב הטסיט⁵, אך \$? echo הוא 0 והתוכנית חוזרת לשגרה המקורית הפקודת הלא חוקית היא `0xF04`, אך ערך החזרה של `so_to_do what_to_do` הוא 0 והתוכנית חוזרת לשגרה המקורית לטיפול, ששולחת את הסיגנל `Illegal Instruction`).

⁵ הטסיט נכתב כך שמיד לאחר הפקודה הלא חוקית יש ביצוע של קריית המערכת `exit`. אתם משים את `%id` בשגרת הטיפול, אך ערך היציאה של הטסיט ישתנה בהתאם.

פקודות שימושיות

insmod ili.ko

(טען את המודול ili.ko ל kernell ו מפעיל את הפונקציה so init_ko שבמודול)

rmmod ili.ko

(מפעיל את הפונקציה so exit_ko שבמודול ומוציא את המודול ili.ko מה kernell)

תקלות נפוצות (מתעדכן)

במקרה של תקלת "אין מקום בדיסק" שמתאפשרה בזמן הרצת ./compile filesystem.img – عليכם להוריד מחדש את הקובץ filesystem.img ולהחליפן את השוטק הישן באחד החדש ואז להריץ את ./compile שוב.

באופן כללי, במהלך הבדיקה יתכן שתצטרכו למחוק את img filesystem אצלכם ולהחליפן אותו בעותק שבאתר, לאחר filesystem.img שנישיטם לבצע שינויים ומשהו השתבש בעותק של הקובץ המקורי אתם עובדים. מומלץ לשמר עותק של img filesystem ללא שינויים בצד, כדי לבצע את ההחלפה הzo במחירות (ולא להוריד כל פעם מהאתר 😊).

הערות כלליות

נשים לב שבתרגיל זה שינינו את הקוד של הגראען! ומכיון שהזה קוד של גראען אז אין לנו דרך ללבג אותו ולהבין אם הוא אכן עובד כפי שציפינו שייעבד. **דיבוג קוד kernell הוא קשה**. כאן לא תוכלו להיעזר ב-gdb. תצטרכו, ברוב הפעמים לנסות "לבדוק בצד" את מה שכabbתם, על דוגמאות עצוצע, ולראות שהקוד עשה מה שהוא אמור לעשות. רוב המחשבה והדיבוג נעשים "בראש" וכן הקוד שאתם אמרוים לבתוב הוא יחסית פשוט וקצר.

ובכל זאת, על מנת לעזור מעט להבין מה קורה בקורס – תוכלו להשתמש בפונקציה print() המוגדרת בקובץ c.main_ili, ולראות את הודעות הקרנל ע"י בתיבה של הפקודה dmesg בטרמינל של qemu. מה print() ולא ?printf() ומה נושא ההפונקציה printf() היא פונקציה של הספרייה libc, ובאשר אנו כותבים קוד גראען או עושים לו מודיפיקציה אין לנו את האפשרות לגשת למספריה זו (ואנחנו גם לא צריכים כי יש לנו רמת הרשות 0, ו-libc היא ספרייה עבור משתמשים), לכן אנו צריכים לגשת לפונקציות שנמצאות בגרעין – אך עדנו לכם וכתבנו עבורכם מעטפת נחמדה 😊. דבר נוסף, אני קראו את הערות בקבי הקוד שעלייכם למלא. זה יכול רק לעוזר.

תיעוד של qemu ניתן למצוא在此: <https://qemu.weilnetz.de/doc/2.11/qemu-doc.html>

חלק ג' - הוראות הגשה לתרגיל הבית

אם הגיעتم לבאן, זו בהחלט סיבה לחגגה. אך בבקשתה, לא לנוכח על זרי הדפנה ולתת את הפוש האחרון אל עבר ההגשה – חבל מאוד שתצטרכו להתעסק בעוד מספר שבועות מעכשווי בערעריהם, רק על הגשת הקבצים לא כדי שנתבהקשתם. אז קראו בעיון ושימו לב שאתם מגישים את כל מה שצריך ורק את מה שצריך.

עליכם להגיש את הקבצים בתוך קובץ אחד (שם הקובץ לא משנה).

בתוך קובץ zip זה יהיו 2 תיקיות:

- part1
- part2

ובתוך כל תיקייה יהיו הקבצים הבאים (מחולק לפי תיקיות):

- part1:
 - students_code.S
- part2:
 - ili_handler.asm
 - ili_utils.c

בהצלחה!!!