הטכניון – מכון טכנולוגי לישראל

**ארגון ותכנות המחשב**

**תרגיל 2 - חלק רטוב**

 המתרגל האחראי על התרגיל: בועז מואב.

שאלותיכם במייל כולל עניינים מנהליים, יופנו רק אליו.

**כתבו בתיבת subject: רטוב 2 אתם.**

שאלות בעל-פה ייענו על ידי כל מתרגל.

הוראות הגשה:

1. ההגשה בזוגות.
2. על כל יום איחור או חלק ממנו, שאינו בתיאום עם המתרגל האחראי על התרגיל, יורדו 5 נקודות.
3. שאלות ותשובות הנוגעות לתרגיל ייענו ב-piazza והבהרות יעודכנו במסמך זה בהתאם לצורך.
4. לתרגיל זה שני חלקים. חלק ב' מתבסס על חלק א'. עבור חלק א' אתם נדרשים להכיר את תרגולים 1-5 ועבור חלק ב', עליכם להכיר גם את תרגול 6.
5. הוראות הגשה מופיעות בסוף התרגיל. **אנא קראו בעיון.**
6. **העתקות ורמאויות יטופלו בחומרה!**

**חלק א – שגרות, קונבנציות ומה שביניהן**

1. **מבוא**

בתרגיל הבית הזה אנו נכתוב... מחשבון! כן, לא נשמע מרגש במיוחד, נכון? אבל זה יעזור לכם לתרגל הרבה מהנושאים של החצי הראשון של הקורס וגם, כשזה יעבוד – תוכלו להגיד לעצמכם "איזה מגניב שאת כל זה בניתי מאמסמבלי!".

**התרגיל מורכב משני חלקים וחלק ב' מבוסס על חלק א' ותלוי בו.** עליכם לסיים את חלק א' לפני שתתחילו את חלק ב'. בחלק א' אנו נבנה את המחשבון המלא ובחלק ב' נוסיף מנגנון שמתעסק במקרים של חלוקה ב-0, באמצעות כתיבת פונקציית טיפול בפסיקת התוכנה הרלוונטית. בחלק א' לא תידרשו לטפל בחלוקה ב-0 כלל, ובטסטים של חלק זה מובטח שלא תתבצע חלוקה ב-0.

המחשבון שלנו יידע לטפל ב-4 פעולות חשבון בלבד – חיבור, חיסור, כפל וחילוק. בנוסף, החישובים ייעשו במספרים שלמים בלבד. תוצאת החילוק תתעגל כלפי ה-0. למשל, וכו'.

**יש לתמוך בחישובים של מספרים בגודל עד 64 ביט.**

1. **מבנה ביטוי חשבוני תקין**

הקלט בתרגיל זה תמיד יסתיים בירידת שורה. ביטוי חשבוני יהיה מהצורה הבאה:

1. מחרוזת ריקה (מלבד תו ירידת שורה בסוף הקלט) – תוצאה של ביטוי כזה הוא 0.
2. “(num)” – כאשר num מספר כלשהו (חיובי או שלילי, כאשר שלילי מכיל את התו ‘-‘ לפניו) בתוך סוגריים. תוצאת ביטוי זה היא הערך של num.
3. “(num OP num)” – כאשר OP הוא אחד מהתווים הבאים ‘+’,’-‘,’/,’\*’.
4. “(EXP)” – ביטוי יכול להיות ביטוי כלשהו בתוך סוגריים – זו הגדרה רקורסיבית. "ביטוי" הוא סעיפים 2-5.
5. “(EXP OP EXP)” – ביטוי יכול להיות ביטוי כלשהו עם פעולה על ביטוי כלשהו אחר בתוך סוגריים – זו הגדרה רקורסיבית.
   * כמו כן, קיימת האופציה לביטויים מהסוג “(EXP OP num)” ו- “(num OP EXP)”.

דוגמאות לביטויים אפשריים (דוגמאות עם פלט ניתן למצוא בסעיף V בחלק זה של התרגיל):

(1)

(-1)

(1--1)

(1-(-1))

((2+3)\*2)

((1\*2)/(3-5))

(1+(1+(1+(1+1))))

**שימו לב שאין רווחים בביטוי חשבוני בתרגיל זה**. (הסיבה שיש רווח בין num ו-OP למשל בסעיף 3, היא כדי שזה יהיה קריא).

עליכים לחשב את הביטוי מהסוגריים הפנימיים ביותר, עד לחיצוניים, כשבכל חישוב ביניים (מהדרך בה הגדרנו ביטוי תקין) תידרשו לעשות פעולה אחת על שני מספרים בלבד. לכן, אין צורך בשמירת סדר פעולות חשבון. בכל ביטוי, הסוגריים יקבעו את סדר החישוב בצורה חד חד ערכית.

**מובטח שכל ביטוי שתקבלו הוא תקין מבחינת הסיטנקס ואין צורך לבדוק את זה!**

1. **מה עליכם לעשות?**

עליכם לממש את הפונקציה calc\_expr בעלת החתימה הבאה (ב-C) **לפי הקונבנציות שלמדנו**:

void calc\_expr(long long (\*string\_convert)(char\*), int (\*result\_as\_string)(long long));

**את השגרה תממשו באסמבלי**, בקובץ hw2\_sol.asm.

קודם כל, מה החתימה המפחידה הזו אומרת? אז תזכורת למי ששכח – כך נראית פונקציה שמקבלת שני פרמטרים מסוג "מצביע לפונקציה". במקרה שלנו, הפונקציה calc\_expr מקבלת שני פרמטרים:

1. ארגומנט בשם string\_convert, שהוא מצביע לפונקציה שמקבלת char\* ומחזירה long long (טיפוס מסוג 8 בתים עם סימן). פונקציה זו מבצעת המרה של מחרוזת **בעלת ספרות בלבד (או סימן מינוס)** למספר מטיפוס long long. למשל, הפעלת הפונקציה string\_convert(“-56784”) תחזיר -56784. אין לתת לפונקציה זו מחרוזת המכילה מספר שחורג מייצגו של 64 ביט (וגם לא תצטרכו זאת). עליכם לסיים את המחרוזת עם בית המכיל 0 (מוכר גם בשם null terminator, או ‘\0’), מכיוון שהפונקציה מפסיקה להמיר כאשר היא נתקלת בתו שאינו ספרה.
2. ארגומנט בשם result\_as\_string, שיקבל כפרמטר מספר בן 64 ביט עם סימן (long long) וממלאת במשתנה הגלובלי what\_to\_print מחרוזת כלשהי, שמכילה את המספר (שהתקבל כפרמטר) כמחרוזת. למשל, בדוגמה שקיבלתם, המחרוזת תהיה “Result is: “ ולאחר מכן המספר שהתקבל כפרמטר.

ערך ההחזרה של פונקציה זו הוא אורך המחרוזת שנכתבה ל- what\_to\_print(לא כולל null terminator בסופה).

**מה פונקציה זו צריכה לעשות?** עליכם לכתוב את calc\_expr כך שתקרא את הקלט של המשתמש מערוץ הקלט הסטנדרטי (stdin), שיהיה ביטוי חשבוני תקין (כאמור, מובטח לכם תקינות), ותחשב אותו. את תוצאת החישוב יש לשלוח ל- result\_as\_stringולהדפיס לערוץ הפלט הסטנדרטי (stdout) את תוכן what\_to\_print לאחר מכן (כמה bytes להדפיס מתוך what\_to\_print? זה בדיוק ערך ההחזרה של result\_as\_string).

**שימו לב –** אין התחייבות על אופן המימוש של שתי הפונקציות האלו. אתם קיבלתם דוגמאות למימוש, אך אין אף התחייבות למימוש שאיתו נבצע טסטים. מה כן מובטח? הפונקציות כמובן ישמרו את הקונבנציות שלמדנו בקורס (קונבנציית 64 ביט בשם System V) ובנוסף יחזירו ערכים כמובטח.

**מותר ומומלץ להשתמש בפונקציות עזר** כאשר אתם פותרים את התרגיל. כתבו כמה פונקציות עזר שתרצו, כל עוד הפונקציה calc\_expr תבצע את האמור לעיל.

1. **עם מה אתם מתחילים?**

קיבלתם קוד בשפת c, שנקרא calc.c. בקוד זה מוצג לכם מימוש לדוגמה של טסט בתרגיל – תהיה פונקציית main שתקרא לקוד שלכם, עם 2 פרמטרים של מצביעים לפונקציות. בדוגמה שקיבלתם, ישנן דוגמה אחת למימוש של כל אחת מהפונקציות.

בנוסף, בקובץ זה מוגדר המשתנה הגלובלי what\_to\_print.

איך להריץ?

as hw2\_sol.asm -o hw2\_sol.o

gcc -no-pie calc.c hw2\_sol.o -o calc

./calc

ולאחר מכן להתחיל להכניס דוגמאות ישירות לערוץ הקלט הסטנדרטי.

בשביל לבדוק את עצמכם, אתם יכולים לשחק עם הפונקציות שקיבלתם ובנוסף, כמובן, לבדוק עוד ועוד קלטים.

אם אינכם בטוחים לגבי פלט צפוי עבור קלט מסוים, אל תהססו לשאול ב-piazza.

1. **דוגמאות**

להלן מספר דוגמאות של קלטים והפלטים הצפויים עבורם:











1. **טיפים והערות**

כמה המלצות והערות לפני שתתחילו את התרגיל:

1. מומלץ לכתוב תחילה את הקוד בשפה עילית כלשהי, או אפילו סודו-קוד (pseudo code), על מנת להחליט אילו פונקציות עזר תצטרכו ובאופן כללי מהי דרך ההתמודדות שלכם עם התרגיל. לאחר מכן, תוכלו להמיר (בחוכמה וביעילות) את הרעיונות הללו לאסמבלי.
2. **מומלץ לתעד**.לא תתבצע בדיקה ידנית של הקוד, אך מומלץ מאוד לתעד אותו לעצמכם, אחרת קל מאוד ללכת לאיבוד.
3. **אין להשתמש בפקודות שלא למדנו.** אם בכל זאת תרצו להשתמש בפקודה מסוימת, תוכלו לשאול עליה בפיאצה ואם נחליט שהיא רלוונטית ומקלה, נאשר.

**חלק ב – פסיקות**

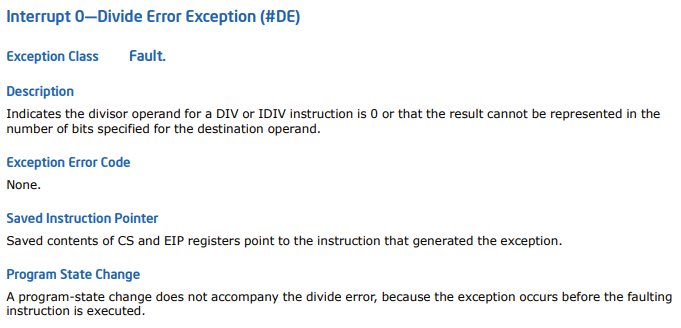
1. **מבוא**

קראו את כל השלבים בחלק זה, לפני שתתחילו לעבוד על הקוד.

בתרגיל זה נרצה לכתוב שגרת טיפול ב**שגיאת חלוקה** המתבצעת כאשר מבצעים חלוקה ב-0, או כאשר אופרנד היעד קטן מדי על מנת להחזיק את התוצאה שהתקבלה.

* שגרת הטיפול נמצאת בקרנל, ובלינוקס היא שולחת סיגנל SIGFPE לתוכנית שביצעה את שגיאת החלוקה.  
  אפשר לראות זאת כאן למשל:

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

* עוד מידע על פסיקה זו, תוכלו לקרוא בספר הגדול של אינטל. להלן סיכום עיקרי הדברים:

נרצה לשנות את **קוד הקרנל** כך ששגרת הטיפול בפסיקה תשתנה. נעשה זאת באמצעות [kernel module](https://en.wikipedia.org/wiki/Loadable_kernel_module).

1. **מה תבצע שגרת הטיפול החדשה?**

שגרת הטיפול בפסיקה שלנו (שאותה **אתם הולכים לממש** באסמבלי בעצמכם, בקובץ de\_handler.asm), תיקרא **my\_de\_handler** ותבצע את הדברים הבאים:

1. קריאה לפונקציה what\_to\_do עם הפרמטר הבא - ערך המחולק בפקודה שהובילה לשגיאה.
   * למשל, אם הפקודה שהביאה לשגיאה הינה 5/0 – הקריאה ל-what\_to\_do צריכה להכיל את המספר 5 כפרמטר.
2. בדיקת ערך החזרה של what\_to\_do
   * אם 0 – העברת השליטה לשגרת הטיפול המקורית.
   * אחרת – חזרה מהפסיקה, כך שהתוכנית תוכל להמשיך לרוץ והמנה של תוצאת החלוקה (לו הייתה תקינה) תהיה ערך ההחזרה של what\_to\_do.

למשל, עבור what\_to\_do שקיבלתם כדוגמה (שמחזירה את הפרמטר שקיבלה, פחות 1), נצפה שעבור 5/0 נקבל שתוצאת החלוקה היא 4.

* **שימו לב:** what\_to\_do הינה שגרה שתינתן על ידנו בזמן הבדיקה. אין להניח לגביה דבר, מלבד חתימתה (כלומר - שם השגרה, טיפוס פרמטר הקלט וטיפוס ערך החזרה)

1. **לפני תחילת העבודה – מה קיבלתם?**

בתרגיל זה תעבדו על מכונה וירטואלית דרך qemu (בתוך המכונה הוירטואלית - Virtualiception). על המכונה הזו, אנחנו נריץ kernel module[[1]](#footnote-1) שיבצע את החלפת שגרת הטיפול לזו שמימשתן בעצמכן.

היות והקוד רץ ב-kernel mode) ring 0), במקרה של תקלה מערכת ההפעלה תקרוס. אך זה לא נורא! עליכם פשוט להפעיל את qemu מחדש.

לרשותכם נמצאים הקבצים הבאים בתיקייה שקיבלתם:

* initial\_setup.sh - הריצו סקריפט זה לפני כל דבר אחר. סקריפט זה מכין את המכונה הוירטואלית לריצת qemu. **עליכם להריץ אותו פעם אחת בלבד.**
  + יכול להיות שתצטרכו להריץ את הפקודה הבאה, לפני ההרצה (בגלל בעיית הרשאות):

chmod +x initial\_setup.sh

* compile.sh - הריצו סקריפט זה בכל פעם שתרצו לקמפל את הקוד ולטעון אותו (עם המודול המקומפל) למכונה הוירטואלית של qemu (**שימו לב:** עליכם לצאת מ-qemu קודם).
  + גם כאן ייתכן ותזדקקו להרצה של chmod באותו אופן.
* start.sh - הריצו סקריפט זה כדי להפעיל את המכונה הוירטואלית של qemu, לאחר שקימפלתם את תיקיית code וטענתם אותה אל המכונה הוירטואלית של qemu.
  + גם כאן ייתכן ותזדקקו להרצה של chmod באותו אופן.
* filesystem.img - המכונה הוירטואלית אותה תריצו ב-qemu.
* קבצי הקוד שנכתוב, כחלק מהמודול (והיא זו שתקומפל ותרוץ לבסוף ב-qemu) וה-makefile:
  + de\_handler.asm, de\_main.c, de\_utils.c, Makefile

1. **איך הכל מתחבר - כתיבת המודול**

בתיקיה סיפקנו לכם מספר קבצים:

**de\_main.c** – initialize the kernel module – provided to you for testing.

**de\_utils.c** – implementation of de\_main's functionality – **YOUR JOB TO FILL**

**de\_handler.asm** – exception handling in assembly – **YOUR JOB TO FILL**

**Makefile** – commands to build the kernel module and the code from part 1 of this assignment.

ממשו את הפונקציות ב-de\_utils.c, כך שהשגרה my\_de\_handler תיקרא כאשר מנסים לבצע שגיאת חלוקה. איך? Well, זהו לב התרגיל, אז נסו להיזכר בחומר הקורס. כיצד נקבעת השגרה שנקראת בעת פסיקה? פעלו בהתאם.

לאחר מכן, ממשו את הפונקציה **my\_de\_handler** ב-de\_handler.asm שתבצע את מה שהוגדר בשלב II.

1. **זמן בדיקות - הרצת המודול**

לאחר שסיימתם לכתוב את המודול, בצעו את השלבים הבאים:

1. הריצו את ./**compile.sh** כדי לקמפל את קוד הקרנל ולהכניסו למכונת ה-QEMU

2. הריצו את ./**start.sh** כדי לפתוח מכונה פנימית באמצעות QEMU

a.      משתמש: **root**, סיסמא: **root**

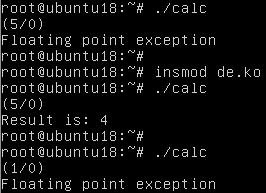
3. **./calc** כדי להריץ את הקוד שכתבתם בסעיף א'. הכניסו כעת פקודה פשוטה שתגרום לשגיאת חלוקה.

* למשל – (5/0).

4. **insmod de.ko** כדי לטעון את המודול שלכם (ודאו שהוא נטען ע"י הרצת **dmesg**)

5. **./calc** כדי להריץ שוב, אך לקבל התנהגות שונה מהקודמת, מכיוון שהפעם השגרה שלכם נקראה.

דוגמת הרצה תקינה ב-QEMU (עם מימוש של **what\_to\_do** שסופק לכם כדוגמה):



1. **פקודות שימושיות**

insmod de.ko

(טוען את המודול de.ko לקרנל ומפעיל את הפונקציה init\_ko שבמודול)

rmmod de.ko

(מפעיל את הפונקציה exit\_ko שבמודול ומוציא את המודול de.ko מהקרנל)

SHIFT + page up

(גלילת המסך למעלה)

SHIFT + page down

(גלילת המסך למטה)

1. **הערות כלליות**

על מנת להבין מה קורה בקרנל – תוכלו להשתמש בפונקציה print() המוגדרת בקובץ de\_main.c, ולראות את הודעות הקרנל ע"י dmesg.

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

הבדיקות בחלק זה יהיו מעטות ופשוטות ויכללו ביטוי חשבוני מהצורה “(X/0)”, עבור X כלשהו, לכן לא צפויות טעויות נגררות. המטרה בחלק זה היא לא לבדוק שוב את חלק א', אלא אך ורק את עניין וקטור הפסיקה, אך הבדיקה תתבסס על הקוד **שלכם** מחלק א'.

**חשוב (!)** - לכל אחת ואחד מכן עלול להיות מימוש מעט שונה של חלק ב', בהתאם לאופן שבו מימשתם את חלק א'. **שימו לב לאופן שבו מימשתם את פעולת החילוק בחלק א', מכיוון שזה הבסיס לאופן שבו תממשו את חלק ב'.**

**הוראות הגשה לתרגיל בית רטוב 2**

אם הגעתם לכאן, זו בהחלט סיבה לחגיגה. אך בבקשה, לא לנוח על זרי הדפנה ולתת את הפוש האחרון אל עבר ההגשה – חבל מאוד שתצטרכו להתעסק בעוד כמה שבועות מעכשיו בערעורים, רק על הגשת הקבצים לא כפי שנתבקשתם. אז קראו בעיון ושימו לב שאתם מגישים את כל מה שצריך ורק את מה שצריך.

עליכם להגיש את הקבצים בתוך zip אחד:

hw2\_wet.zip

אילו קבצים צריכים להגיש?

hw2\_sol.asm

de\_handler.asm

de\_utils.c

**אין להגיש קבצים נוספים.**

1. למי שלא מכיר את המונח kernel module, בלי פאניקה (כי panic זה רע, אבל זה עוד יותר רע בקרנל. פאניקה! בדיסקו זה דווקא בסדר) – מדובר בדרך להוסיף לקרנל קוד בזמן ריצה (ניתן להוסיף לקרנל קוד ולקמפל לאחר מכן את כל הקרנל מחדש, אך כאן לא הזמן ולא המקום לזה). למעשה, נכתוב קוד שירוץ ב-kernel mode ולכן יהיה בעל הרשאות מלאות. אנו נדרש לזה – הרי אנו רוצים לשנות את קוד הקרנל. [↑](#footnote-ref-1)