# ארגון ותכנות המחשב

## תרגיל 4 - חלק רטוב

המתרגל האחראי על התרגיל: בועז מואב.

**שאלות על התרגיל – ב- Piazza בלבד.**

הוראות הגשה:

- ההגשה בזוגות.
- על כל יום איחור או חלק ממנו, שאינו באישור מראש, יורדו 5 נקודות.
    - ניתן לאחר ב-3 ימים לכל היותר.
- הגשות באיחור יתבצעו דרך אתר הקורס.
- הוראות הגשה נוספות מופיעות בסוף התרגיל. אנא קראו בעיון.

# פענוח קובץ **ELF** וכתיבת דיבאגר

## הקדמה

רביבו עובד על פרויקט עם חברים שלו. לצערו של רביבו, חבריו מהפרויקט לא בודקים את ערכי החזרה של פונקציות, לכן רביבו רוצה לעקוב אחרי ריצת התוכנית בעצמו.

לשם כך, רביבו רוצה שתבנו לו תוכנית שתקבל:

1. שם של פונקציה שאותה ירצה לחקור ולבדוק את ערכי החזרה שלה במהלך התוכנית
2. שם קובץ הריצה
3. Command-line parameters של קובץ הריצה

התוכנית שלכם תריץ את קובץ הריצה שלו ותדפיס למסך את ערכי החזרה של כל קריאה לפונקציה במהלך התוכנית.

לביצוע התרגיל, אתם נדרשים להכיר את כל התרגולים עד תרגול 12, בו לומדים כיצד לשים breakpoints.

עם זאת, תוכלו לבצע את שלבים 1-4 כבר לאחר תרגולים 8-10, ולכן מומלץ להתחיל את התרגיל איתם, גם אם עוד לא למדתם את תרגול 12. את שלב 5 תוכלו לתכנן רעיונית כבר עכשיו, אך כדי לבצע אותו (ואת שלב 6) יש לעבור את תרגול 12.

## כתיבת תוכנית הדיבאגר

### ארגומנטים (**Command Line Arguments**)

התוכנית שתכתבו תקבל כמה ארגומנטים:

- ארגומנט ראשון – <u>השם</u> של הפונקציה הנבדקת (שבה צריך לבדוק את ערך החזרה).
- ארגומנטים 2 והלאה – התוכנית שיש להריץ (כולל ארגומנטים נוספים, command line arguments לתוכנית עצמה, למשל כמו הארגומנטים שמקבלת הפקודה time)

### אחרי מי התוכנית עוקבת?

כל המעקב מתבצע על פונקציה <u>גלובלית</u> אחת בלבד.

שם הפונקציה מסופק לכם כפרמטר ועליכם למצוא אותה.

מובטח לכם כי אם הפונקציה קיימת, הפונקציה מחזירה int.

### שלבים בהרצת התוכנית

#### שלב ראשון – בדיקת סוג הקובץ

תחילה יש לבדוק אם הקובץ executable. אם לא – עליכם להדפיס:

```
PRF:: <prog name> not an executable! :(\n
```

כאשר prog name הוא הארגומנט השני שקיבלתם. **לאחר מכן יש לסיים את ריצת התוכנית.**

אם אכן הקובץ הוא executable, יש להמשיך לשלבים הבאים.

### שלב שני – חיפוש הפונקציה

בשלב זה עליכם לבדוק האם קובץ הריצה שקיבלתם "מכיר" את הפונקציה שקיבלתם בארגומנט הראשון. כלומר, עליכם לבדוק האם הפונקציה קיימת בטבלת הסמלים של קובץ הריצה. אם לא – עליכם להדפיס:

```
PRF:: <function name> not found!\n
```

כאשר function name הוא הארגומנט הראשון שקיבלתם. **לאחר מכן יש לסיים את ריצת התוכנית.**

אם הפונקציה כן קיימת בטבלת הסמלים של קובץ הריצה, יש להמשיך לשלבים הבאים.

### שלב שלישי – בדיקת נראות הפונקציה

אם אין פונקציה גלובאלית בעלת השם המבוקש, אך כן קיים הסימבול בקובץ הריצה. עליכם להדפיס:

```
PRF:: <function name> is not a global symbol! :(\n
```

כאשר function name הוא הארגומנט הראשון שקיבלתם. **לאחר מכן יש לסיים את ריצת התוכנית.**

אם הפונקציה כן גלובאלית, יש להמשיך לשלבים הבאים.

### שלב רביעי – בדיקת מיקום הסימבול

אם הגענו לשלב זה, אנו יודעים שבקובץ הריצה מוכר הסימבול שהוא הארגומנט הראשון שקיבלנו, הפונקציה אחריה עוקבים, ובנוסף שמדובר בפונקציה גלובאלית. כעת יש שתי אפשרויות:

1. הפונקציה (הסימבול) מוגדרת באחד ה-sections של קובץ הריצה, לכן ניתן לדעת לפני תחילת הריצה לאיזו כתובת הוא ייטען (למה?).
   - ניתן להניח כי כל קובץ ריצה שיווצר בעזרת קומפיילר, יקומפל עם הדגל no-pie-.
   - במקרה זה ניתן לדלג לשלב השישי.
2. הפונקציה לא מוגדרת בקובץ, לכן Ndx=UND והיא תיטען רק בזמן ריצה וכתובת הטעינה שלה אינה ידועה מראש.
   - לכן נדרש שלב ביניים, שלב חמישי, למציאת מיקום הסימבול.

### שלב חמישי – מציאת מיקום סימבול בזמן ריצה

אם הגעתם למצב בו הסימבול שלכם לא מוגדר בקובץ הריצה, אך כן נמצא בשימוש בקובץ (ולכן הוגדר בטבלת הסימבולים, אך כ-UND) – אזי הוא מוגדר בספרייה דינמית.

אז איך מוצאים היכן הוא יהיה בזמן ריצה? את זה אתם אמורים לדעת. היזכרו בתרגול 10.

**<u>רמזים, הנחיות והנחות:</u>**

1. השדה symbol בטבלאות relocation הוא לא באמת שם הסימבול. גם כאן, בדיוק כמו במקומות אחרים בהם יש שימוש במחרוזות ב-ELF, המחרוזת לא נשמרת בשדה עצמו. אך לעומת מקרים אחרים, בהם פנינו לטבלאות strtab למיניהן, כאן הסיפור שונה. השדה symbol מכיל מספר שמייצג כניסה ב-symtab, שעל פיה עושים את התיקון.
   a. עבור טבלאות relocation של ספריות דינמיות יש להשתמש ב-dynsym ולא ב-symtab.[1]
   b. היעזרו במאקרו ELF64_R_SYM [2] על מנת לחלץ את ה-index המתאים ב-dynsym, מתור שורת relocation.
2. כל הטסטים שמשתמשים בקישור דינמי עושים זאת ב-Lazy Binding.[3]

---

[1] https://blogs.oracle.com/solaris/post/inside-elf-symbol-tables
[2] https://docs.oracle.com/cd/E19683-01/816-1386/chapter6-54839/index.html
[3] תופתעו לגלות כמה המצב דווקא מסתבך כשלא מדובר ב-Lazy Binding.

### שלב שישי – בדיקת ערכי החזרה בזמן ריצה

אם הגעתם לשלב הזה, יש בידיכם את המידע החשוב – היכן נמצאת (או היכן אפשר למצוא את המידע אודות מקום הימצאה בזמן ריצה) הפונקציה שאחריה אתם עוקבים.

כאמור, התוכנית שלכם צריכה לבדוק עבור פונקציה מסוימת את ערכי החזרה שלה. לכן, בכל קריאה לפונקציה, עליכם לבדוק בסופה את ערך החזרה שלה ולהדפיס:

```
PRF:: run #<call_counter> returned with <return_value>\n"
```

כאשר call_counter הוא משתנה שמתחיל ב1 ובכל קריאה לפונקציה יעלה באחד (בקריאה הראשונה יהיה 1, בקריאה השניה 2 וכו'), ו-return_value הוא ערך ההחזרה של הפונקציה בריצה זו.

**הערה**: עבור קריאות רקורסיביות, יש להדפיס את ההדפסה הנ"ל רק פעם אחת – בחזרה מהקריאה הראשונה (המקורית) לפונקציה (ולא בחזרה מקריאות הנוספות, במקרה בו לא חזרה עדיין מהראשונה).

## הערות

1. אין להשתמש בספריות או כלים חיצוניים כדי לנתח את קובץ הריצה!

   **פתרונות שיכללו שימוש בכלים חיצוניים (כדוגמת readelf) - יפסלו!**

2. שימו לב להוראות. כל הדפסות התוכנית צריכות להסתיים בירידת שורה ולהתחיל עם prefix של " PRF::"

## דוגמת ריצה

```
$ cat basic_test.c
int foo(int a, int b) {
    return a+b;
}

int main () {
    foo(3,4);
    foo(0,0);
    foo(42,42);
    return 0;
}

$ gcc basic_test.c -no-pie -o basic_test.out
$ gcc -std=c99 hw4.c -o prf
$ ./prf foo ./basic_test.out
PRF:: run #1 returned with 7
PRF:: run #2 returned with 0
PRF:: run #3 returned with 84
```

# מה עליכם להגיש

אנא קראו בעיון את החלק הזה ושימו לב שאתם מגישים את מה שצריך לפי ההוראות המופיעות כאן - חבל מאוד שתצטרכו להתעסק בעוד מספר שבועות מעכשיו בערעורים, רק על הגשת הקבצים לא כפי שנתבקשתם.

עליכם להגיש קובץ zip יחיד המכיל את כל הקבצים הנחוצים לבניית הפתרון. אין לשים תיקיות ב-**zip**.

אנא הגישו קבצים תקינים בלבד.

אין לצרף את הקובץ **elf64.h**! (אנחנו נצרף אותו בעצמנו)

אנו נבנה את התוכנית שלכם באופן הבא:

```
unzip SUBMITTED_ZIP_FILE

gcc -std=c99 *.c -o prf
```

(שימו לב לקמפול לפי c99).

**הערה:** אם ברצונכם לכתוב תוכניות בשפת C עבור טסטים, הקפידו לקמפל אותם עם דגל -no-pie (דבר זה יאפשר לכם לגלות לאן יטען כל דבר בזיכרון)

# הערות כלליות

תיעוד של ptrace.

תוכנית ניפוי (debugger) לדוגמא ניתן למצוא בחומר הקורס.

ניתן להניח כי:

1. הקלט תקין: התוכנית שלכם תמיד תקבל את מספר הפרמטרים הנדרש בסדר הנכון (כפי שצוין לעיל) והם יהיו בפורמט תקין.
2. בכל מקרה של שגיאת מערכת הפעלה בתוכנית שלכם יש לצאת מיד עם קוד שגיאה 1 (לבצע exit(1)).
3. ניתן להניח שאם התהליך (שמקבלים כקלט) רץ בפני עצמו הוא יסתיים בצורה תקינה.
4. אין לשנות את התנהגות התוכנית המדובגת (שמקבלים כקלט).
5. **ניתן להניח שהפונקציה הנבחרת לא תהיה הפונקציה main (או _start).**
6. איננכם נדרשים לכתוב את המימוש יעיל ביותר, אולם פתרונות איטיים עלולים להוביל להורדת ניקוד.