ארגון ותכנות המחשב

תרגיל 4 - חלק רטוב

המתרגל האחראי על התרגיל: עידו סופר.

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.
- **היזהרו מהעתקות!!!** תרגילים שנחשדים בהעתקות בסבירות גבוהה, יקבלו 0 על תרגיל הבית, ואף תיתכן התערבות משמעתית.

כתיבת דיבאגר

הקדמה

שרלוק הולמס אוהב לחקור פונקציות חשודות בקבצים שהוא מקבל. לכן, שרלוק רוצה שתבנו לו תוכנית שתקבל:

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

התוכנית שלכם תריץ את קובץ הריצה ששרלוק קיבל (יחד עם ה-command-line arguments) ותדפיס למסך ערכי החזרה של כל קריאה לפונקציה החשודה במהלך התוכנית, כמו גם את הפרמטר הראשון של אותה פונקציה.
שימו לב שחלקים מהתרגיל כבר השלמתם בתרגיל בית 3 (שלבים 1-4). וודאו שחלקים אלו עובדים לכם.
את שלב 5 אתם יכולים לפתור כבר לאחר תרגול 10 (מדובר בשלב מחשבתי בלבד).
שלב 6 ואחרון – שלב בניית הדיבאגר, דורש את תרגולים 11-12.

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 ופרמטר ראשון).
- ארגומנטים 2 והלאה – התוכנית שיש להריץ (שם התוכנית ראשון ולאחר מכן ייתכנו ארגומנטים נוספים, ה-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. ניתן להניח כי אם קיים סמל בטבלת הסמלים ששמו הוא שם הפונקציה שהתקבל כארגומנט הראשון ב-command line arguments – מדובר בפונקציה. אין צורך לערוך בדיקות לגבי type הסמל.

בנוסף, מובטח לכם כי אם הפונקציה קיימת, הפונקציה מקבלת כפרמטר ראשון int ומחזירה 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. בנוסף, אם הקובץ אינו executable אין להמשיך לשלבים הבאים ויש לדלג ישירות לסיום הריצה בצורה תקינה (כלומר, ביצוע return 0 מ-main).

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. אם הפונקציה לא קיימת בטבלת הסמלים של קובץ הריצה אין להמשיך לשלבים הבאים ויש לדלג ישירות לסיום הריצה בצורה תקינה (כלומר, ביצוע return 0 מ-main).
אם הפונקציה כן קיימת בטבלת הסמלים של קובץ הריצה, יש להמשיך לשלבים הבאים.

שלב שלישי – בדיקת נראות הפונקציה

אם אין פונקציה גלובאלית (סמל גלובלי) 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. אם לא קיים סמל גלובלי עם השם הזה, אין להמשיך לשלבים הבאים ויש לדלג ישירות לסיום הריצה בצורה תקינה (כלומר, ביצוע return 0 מ-main).
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 שייווצר 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. שימו לב שגם אם הקריאה עוברת ב-PLT, שם הסימבול בטבלאות הסמלים לא כולל `@plt` (שהוא רק הכינוי לכניסה המתאימה ב-PLT). בנוסף, לא תידרשו לעקוב אחר פונקציות ספריה של C (להן כן יש סיומות שונות, כגון `@GLIBC` ועוד) – ובקיצור, שם הסימבול שתקבלו הוא בדיוק זה שיופיע בטבלאות הסמלים.

---

[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

שלב שישי – בדיקת ערכים 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, בכל קריאה לפונקציה, עליכם לבדוק שני ערכים:

1. בתחילתה, את ערך הפרמטר הראשון שלה:

```
PRF:: run #<call_counter> first parameter is <param1>\n"
```

כאשר call_counter הוא משתנה שמתחיל ב1 ובכל קריאה לפונקציה יעלה באחד (בקריאה הראשונה יהיה 1, בקריאה השניה 2 וכו'), ו-param1 הוא ערך של פרמטר מטיפוס int.

2. בסופה, את ערך החזרה שלה ולהדפיס:

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

כאשר call_counter הוא כפי שהוגדר, ו-return_value הוא ערך ההחזרה (גם int) של הפונקציה בריצה זו.

הערה: עבור קריאות רקורסיביות, יש להדפיס את ההדפסות רק פעם אחת – בכניסה ובחזרה מהקריאה הראשונה (המקורית) לפונקציה (ולא בחזרה מקריאות הנוספות, אם לא חזרה עדיין מהראשונה). כמו כן, call_counter עולה רק בקריאה הראשונה לפונקציה רקורסיבית. שימו לב שייתכנו גם רקורסיות הדדיות (foo->bar->foo->...->bar וכו') וגם בהן יש להדפיס רק בקריאה הראשונה לכל אחת מהן. בנוסף, אם נדרשתם לעקוב אחרי פונקציה מספריה דינמית, ניתן להניח כי הקריאה הראשונה במהלך הריצה תגיע מקוד של קובץ הריצה, אך ייתכנו לאחריה קריאות נוספות מתוך ספריות דינמיות אחרות וגם אחריהן תדרשו לעקוב באותו האופן.

הערות

1. שימו לב שקיבלתם קובץ עזר בשם elf64.h. השתמשו בו ובהגדרות שהוא מספק, על מנת להקל על הפרסור (parsing) של קובץ ה-ELF. אין לשנות את הקובץ, מכיוון שלא תגישו אותו לבסוף.
2. מלבד elf64.h, מותר להשתמש בספריות הסטנדרטיות של C ובספריית string.h, אך אין להשתמש בספריות ייעודיות כלשהן לניתוח קבצים ובנוסף אין להשתמש בכלים חיצוניים כדי לנתח את קובץ הריצה!
   פתרונות שיכללו שימוש בכלים חיצוניים (כדוגמת **readelf**) - יפסלו!
3. יש לסיים את התוכנית תמיד דרך ה-main בעזרת return 0. בפרט, אין לבצע exit בעצמכם.
4. אפשר להניח כי כל קריאות המערכת יצליחו ואין צורך לבדוק זאת בקוד המוגש. עם זאת, בשלב ה-debugging, זה כנראה יועיל לכם לבדוק זאת עבור עצמכם.
5. שימו לב להוראות. כל הדפסות התוכנית צריכות להסתיים בירידת שורה ולהתחיל עם prefix של " PRF::".
6. ניתן להניח שלא יהיו הדפסות של התוכניות המדובגות.

דוגמת ריצה

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

int main () {
    foo(3,4);
    foo(0,0);
    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 first parameter is 3
PRF:: run #1 returned with 7
PRF:: run #2 first parameter is 0
PRF:: run #2 returned with 0
```

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. יש להניח כי הוא יהיה באותה תיקיה עם קבצי ה-**c** שלכם)

אנו נבנה את התוכנית שלכם באופן הבא:

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

(שימו לב לקמפול לפי c99)

הערות

1. 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 וגם ידאג שקובץ הפלט יהיה executable)
2. כל הפונקציות ישמרו על קונבנציות System V, אבל לא בהכרח ייכתבו בשפת C (כלומר, ייתכנו טסטים שנכתבו באסמבלי. לא בשפות עיליות אחרות).
3. לא יעלו טסטים, מלבד דוגמת ההרצה שקיבלתם בעמוד הקודם. הכנת הטסטים לתרגיל זה היא באחריותכם.

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. ניתן להניח שלא יהיו שגיאות של מערכת ההפעלה.
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.