ארגון ותכנות המחשב

תרגיל בית 2 (יבש)

המתרגל האחראי על התרגיל: לירן רדל.

הנחיות:

- שאלות על התרגיל ב- **Piazza** בלבד.
- 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.
- לכל שאלה יש לרשום את התשובה במקום המיועד לכך.
- את התרגיל יש להגיש באתר הקורס כקובץ **pdf**.
- יש לענות בטופס התרגיל. ניתן לעשות זאת באחת מהאפשרויות הבאות:
  - להקליד את התשובות במסמך ה-WORD ולבסוף לשמור כ-pdf.
  - לכתוב אותן על גבי גרסת ה-pdf, בעזרת הטאבלט החביב עליכן בכתב ברור וקריא.
  - להדפיס את גרסת ה-pdf ולכתוב על הטופס המודפס את התשובות בכתב יד ברור וקריא.
- תיקונים לתרגיל, אם יהיו, יופיעו ממורקרים.

חלק ראשון – פונקציות ומחסנית (60 נקודות)

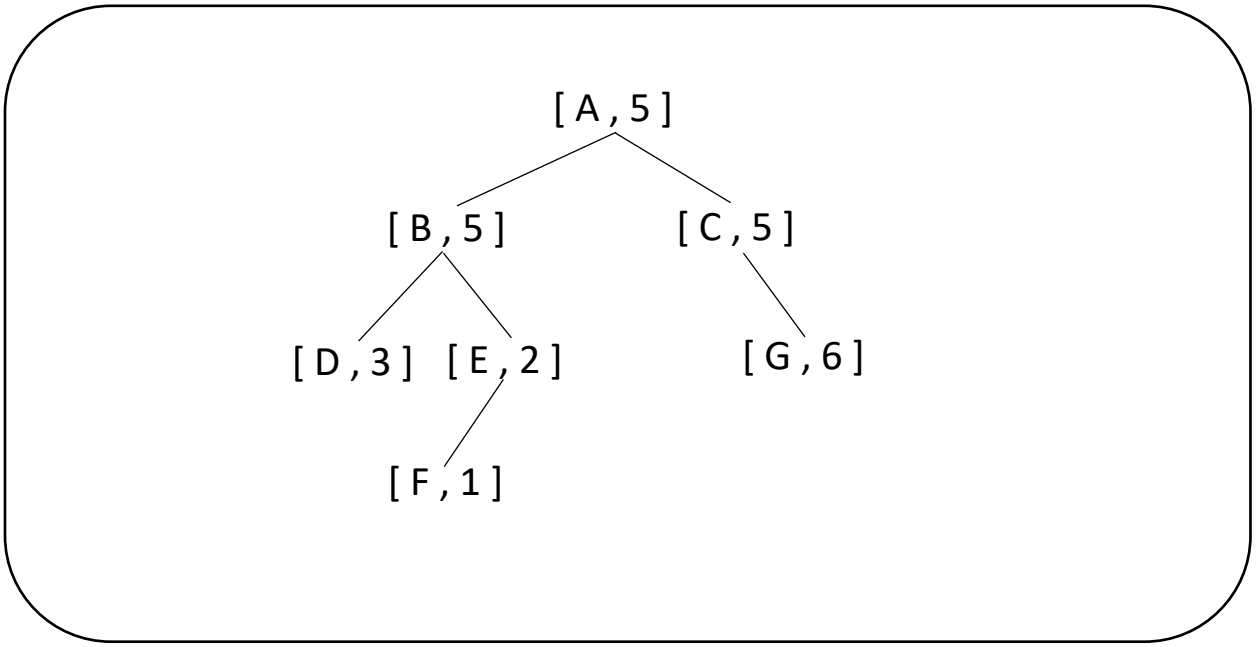
ניק רוצה לקחת ספינה מויקטוריה לאורביס. אך אבוי, קרתה תקלה במערכות הכרטיסים של הספינה. ניק מפחד שיפספס את הספינה, ולכן משתמש בכישורי האסמבלי שרכש בקורס את"מ כדי למצוא את התקלה!

ראשית, נתבונן במבנה הנתונים הבא. כמות הזיכרון שהוקצתה ל-buf אינה ידועה לניק, אך היא לפחות בית אחד.

1	.section .data		13		.quad 0
2	buf:	._____ 0	14		.quad 0
3	A:	.int 5	15	E:	.int 2
4		.quad B	16		.quad F
5		.quad C	17		.quad 0
6	B:	.int 5	18	F:	.int 1
7		.quad D	19		.quad 0
8		.quad E	20		.quad 0
9	C:	.int 5	21	G:	.int 6
10		.quad 0	22		.quad 0
11		.quad G	23		.quad 0
12	D:	.int 3			

סעיף 1 (5 נקודות)

עזרו לניק לצייר את הגרף המתקבל מפירוש מקטע הנתונים (מומלץ להסתכל בתרגול 3, תרגיל 1, ולהיזכר ביצד מפרשים את הזכרון לגרף). בכל צומת בגרף ציינו את התווית המתאימה לה ליד הערך המספרי המתאים לו. לדוגמה: [A , 5] מציין ש-A זה שם התווית ו-5 זהו הערך אשר נמצא בתווית זו.



כחלק מהמאבק במערכת הכרטיסים, ניק הצליח לקבל את קוד האסמבלי של המכונה, מהקובץ `tickets.asm`, והוא מצורף לכם כאן. קוד זה משתמש במבנה הנתונים המתואר בתחילת השאלה.

tickets.asm

```
1  .global _start, sod
2  .section .text
3  _start:
4    leaq A, %rdi
5    call sod
6    movq %rax, buf
7    leaq buf, %rsi
8    movl $1, %eax
9    movl $1, %edi
10   movl $1, %edx
11   syscall
12   movq $60, %rax
13   movl $0, %edi
14   syscall
15 sod:
16   pushq %rbp
17   movq %rsp, %rbp
18   subq $32, %rsp
19   movq %rdi, -24(%rbp)
20   cmpq $0, -24(%rbp)
21   jne .L2
22   movl $1, %eax
23   jmp .L3
24 .L2:
25   movq -24(%rbp), %rax
26   movq 4(%rax), %rax
27   movq %rax, %rdi
28   call sod
29   movl %eax, -4(%rbp)
30   movq -24(%rbp), %rax
31   movq 12(%rax), %rax
32   movq %rax, %rdi
33   call sod
34   movl %eax, -8(%rbp)
35   movl -4(%rbp), %eax
36   imull -8(%rbp), %eax
37   movl %eax, -12(%rbp)
38   movq -24(%rbp), %rax
39   movl (%rax), %eax
40   imull -12(%rbp), %eax
41   movl %eax, %edx
42   movq -24(%rbp), %rax
43   movl %edx, (%rax)
44   movq -24(%rbp), %rax
45   movl (%rax), %eax
46 .L3:
47   leave
48   ret
```

סעיף 2 (15 נקודות)

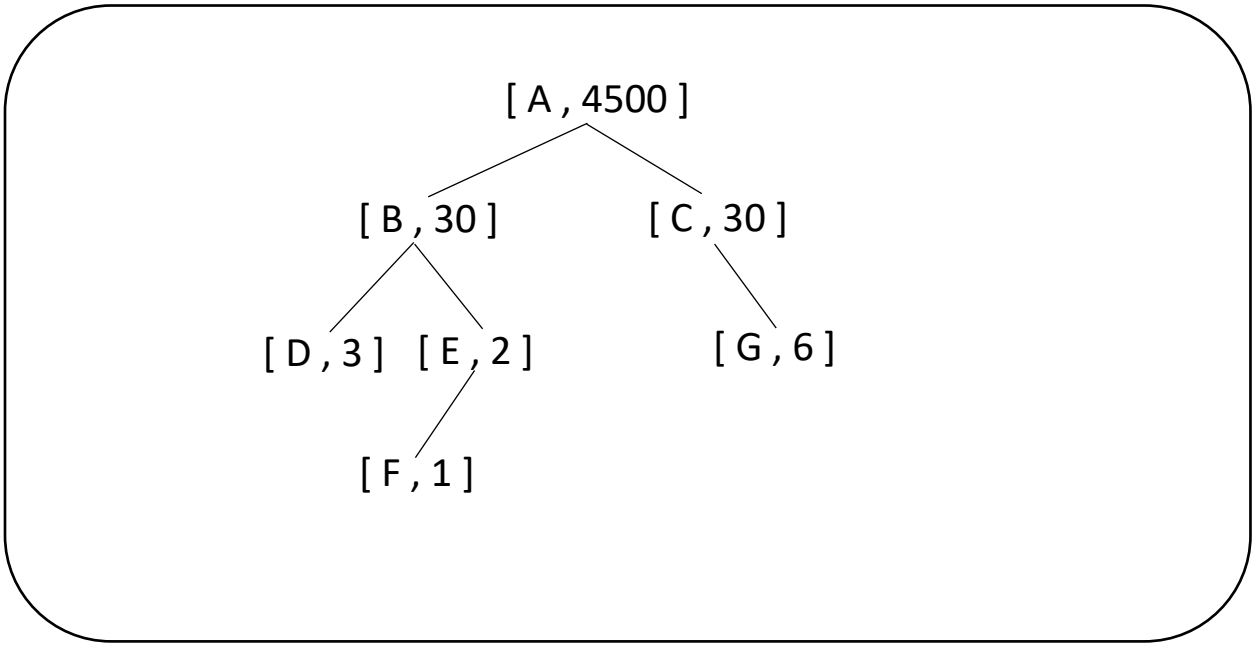
עזרו לניק לתרגם את הקוד של tickets.asm משפת אסמבלי לשפת C, על ידי כך שתשלימו את המקומות החסרים בקטע הקוד הבא. מותר להשלים יותר ממילה אחת בכל קו, אך לא יותר מפקודה אחת בכל קו!

```
struct TreeNode {

      int value;

      TreeNode * left;

      TreeNode *  right;

}__attribute__((packed)); // הסבר בתחתית העמוד


int sod ( TreeNode* root) {

      if (root == NULL) { //in our case, a null is a 0 address

                  return 1;

      int leftProduct = sod( root->left ) ;

      int rightProduct = sod( root->right ) ;

      int tmp = leftProduct * rightProduct;

      root->value *= tmp;

      return root->value;

}
```

__attribute__((packed)) משמש כדי לומר לקומפיילר לסדר בזכרון את איברי ה-struct ללא הוספת ביטים שישמשו כריפודים בזכרון, מאחר ובאופן דיפולטיבי הקומפיילר מכניס ריפודים בין איברי struct משיקולי alignment על מנת לייעל גישה לזכרון וביצועים. (כיף לדעת לידע כללי😊)

סעיף 3 (5 נקודות)

עזרו לניק לצייר את הגרף מחדש לאחר שהרצנו עליו את הקוד של `tickets.asm`:



סעיף 4 (5 נקודות)

נתון שבתחילת התוכנית ערך rsp הינו X, כאשר X הוא מספר **הקסדצימלי**. רשמו את טווח הערכים של rsp בעזרת X לאורך ריצת התוכנית על הקלט שניתן בתחילת השאלה (את טווח הערכים יש גם לרשום בבסיס הקסדצימלי).

$$X - 0XF0 \leq rsp \leq X$$

הסבר: המחסנית גדלה כלפי מטה ולכן הערך המקסימלי יהיה X, עבור הערך המינימלי: בכל קריאה לcall sod אנחנו שמים את rip על המחסנית, עושים pushq rbp, ומאתחלים מקום של 32 בית על המחסנית כלומר 48 ביתים בכל קריאה, עומק הקריאות המקסימלי הוא 5 עבור A->B->E->F->NULL ואז סה"כ 5*48 ביתים כלומר 1920 בטים ולכן זהו rsp -0xF0 (בביתים).

סעיף 5 (5 נקודות)

הציעו פקודה שתחליף את 2 הפקודות בשורות 44-45, ואשר חוסכת בפניות לזיכרון מבלי לשנות את התוכנית.

שתי הפקודות 44 ו-42 הן זהות וזהו רק שכפול של קוד ולכן נשים רק את פקודה 45 במקום 44 ו-45 יחד כלומר:

```
movl (%rax), %eax
```

סעיף 6 (5 נקודות)

מה מחזירה הפונקציה sod? יש להסביר באופן כללי ולציין את ערך החזרה הספציפי בריצה עבור מבנה הנתונים הנתון בתחילת השאלה.

הפונקציה sod עוברת על הגרף עד לעלים ואז עולה בגרף חזרה ומכפילה את הvalue של כל צומת במכפלת הvalues של כל ילדיה, ובסוף היא תחזיר את מכפלת כל הvalues בגרף.

עבור הקוד הנתון נקבל בסוף 4500 בדצימלי או 0x1194 בהקסאדצימלי בפונקציה sod.

סעיף 7 (5 נקודות)

מה יהיה הפלט (לערוץ הפלט הסטנדרטי) של התוכנית בקוד `tickets.asm`? יש לכתוב תשובה **בהקסדצימלי**.

הפלט יהיה: 0x94

סעיף 8 (5 נקודות)

ניק שם לב שהפלט בסעיף 7 אינו תואם את התשובה שקיבל בסעיף 6. הסבירו מה הבאג בתוכנית שגורם לכך.

את הבעיה נובעת מכך שבקריאה לsyscall של write, שמנו בתוך הרגיסטר rdx את המספר 1, rdx מסמן את מספר הבתים שיש להדפיס, במקרה שלנו אנחנו אמורים להדפיס 1194 בהקסאדצימלי כלומר 2 בתים כלומר יש בית חסר ולכן המספר לא מודפס במלאו.

סעיף 9 (5 נקודות)

הציעו לניק תיקון לקוד, כך שהפלט לערוץ הסטנדרטי יהיה תואם לערך החזרה של הפונקציה sod. יש לציין במפורש אילו שורות שיניתם ומה השינוי.

את שורה 10: `movl $1, %edx` נשנה ל: `movl $2, %edx`

אם רוצים שזה יהיה שינוי עבור כל פלט ולא רק את הפלט הנתון אצלנו ב data section נשנה את השורה ל:

```
movl $4, %edx
```

סעיף 10 (5 נקודות)

לאחר שניק תיקן את קוד המכונה, הגיע בלרוג רשע ושינה בקובץ `tickets.asm` כל הופעה של הרגיסטר rdi ברגיסטר rsi (בפרט, לא משפיע על שורות 9 ו-13). כך, למשל, בשורה 27 הקוד יהיה כעת `movq %rax, %rsi`.
מה תהיה הבעיה בשינוי כזה? הניחו כי הפונקציה sod צריכה בעתיד לשמש עוד מכונות של חברות אחרות.

הבעיה עכשיו היא שאנחנו לא עומדים בקונבנציות קריאה לפונקציות, לפי מה שלמדנו בתרגול 4 יש סדר מסוים להעברת פרמטרים לפונקציות, את 6 הפרמטרים הראשונים מעבירים דרך rdi->rsi->rdx->rcx->r8->r9 בסדר הספציפי הזה, ואת השאר מעבירים על המחסנית, ולכן אם בעתיד חברה אחרת תרצה להשתמש בקוד שלנו היא תעביר את הארגומנט הראשון דרך rdi ולא דרך rsi, ולכן הפונקציה לא תפעול עם הארגומנטים הנכונים.

חלק שני – קריאות מערכת (40 נקודות)

ניק הגיעה לאורביס בהצלחה, במשימה ללמוד את הסודות של קריאות מערכת על מנת שיוכל לבנות מערכת יעילה שתוכל להביס את זאקום. עזרו לניק להבין לעומק על ידי מענה השאלות הבאות:

סעיף 1 (10 נקודות)

עבור כל אחת מהטענות הבאות, סמנו נכון/לא נכון. אין צורך בהסברים.

1. בעת קריאת מערכת הפעלה (syscall), החלפת מחסנית המשתמש למחסנית גרעין מתבצעת על ידי קוד ה-handler[1]. נכון / לא נכון
2. לאחר ביצוע קריאת מערכת הפעלה (syscall), ערך החזרה מתקבל ב-rax. נכון / לא נכון
3. הפקודה sysret מעדכנת רק את רגיסטר RIP. נכון / לא נכון
4. בביצוע קריאת מערכת הפעלה (syscall) בארכיטקטורת 64bit, כתובת החזרה מה-handler לקוד המשתמש עוברת דרך המחסנית. נכון / לא נכון
5. רמת ההרשאה הנוכחית של תהליך נשמרת ב-2 ביטים ברגיסטר CS. נכון / לא נכון

המשך השאלה בעמוד הבא

[1] כפי שנלמד בהרצאות, ה-handler בלינוקס (ובקורס) הוא entry_SYSCALL()

ניק רוצה להבין לעומק מהי חלוקת האחריות בין המשתמש ומערכת ההפעלה כאשר מתבצעת קריאת מערכת.

סעיף 2 (5 נקודות)

מלאו את הטבלה הבאה, כך שתהיה ברורה חלוקת האחריות בין המשתמש ומערכת ההפעלה בהקשרים של גיבוי ערכים רלוונטיים לתוכנית. יש להתייחס רק למה שנלמד בקורס.

סוג הקוד	ערכים בתוכנית שבאחריותו לגבות
משתמש	Rcx,r11,rax,rdi,rsi,rdx,r10,r8,r9 במידה ונעשה בהם שימוש
מערכת ההפעלה	את כל הריגסטרים

על מנת להילחם בזאקום, ניק החליט להיעזר בבישוף החכם עדן שיעזור לו לכתוב קריאת מערכת להבסתו, מכיוון שזאקום היא מפלצת חזקה ודורשת שימוש בכל משאבי המערכת יחד, מה שמשתמש אינו יכול לקבל ללא קריאת מערכת.

סעיף 3 (5 נקודות)

שניהם החליטו על השם sys_attack כשם לקריאת המערכת החדשה, והם רוצים להוסיפה למערכת ההפעלה. כיצד יוכלו לעשות זאת? (רק לפי מה שנלמד בקורס)

אנחנו צריכים להוסיף את NR_syscalls ולהוסיף את הקריאה sys_attack בטבלת הפונקציות sys_call_table

המשך השאלה בעמוד הבא

סעיף 4 (5 נקודות)

לזאקום 8 ידיים, שנדרש להפילן על מנת להביסו. לכן ניק רוצה להיות מסוגל לתקוף את 8 הידיים של זאקום בבת אחת, ולכן נדרש להעביר 8 פרמטרים כאשר כל פרמטר הוא מיקום (ערך מספרי שלם) של יד ספציפית לקריאת המערכת שלו. האם ניק מסוגל לעשות זאת ביצירת קריאת המערכת? **נמקו.**

כן , אנחנו יכולים להעביר את את חמשת המיקומים של הידיים של זאקום ברגיסטרים rdi,rsi,rdx,r10,r8 ובריגסטר r9 אנחנו מעבירים פוינטר למערך שמכיל את 3 המיקומים הנותרים, ובכך העברנו את כל המיקומים לקריאה

סעיף 5 (5 נקודות)

לאחר יצירת קריאת המערכת בסעיף 3, ניק החליט שאין צורך בחזרה מקריאת המערכת עם sysret, אלא שישתמש רק ב-ret מאחר שאינו משנה את רגיסטר הדגלים. עדן טוען שניק טועה. מי מהם צודק, ומדוע?

עדן, בגלל שב ret אנחנו טוענים את ה return address מהמחסנית ל rip , אבל ב syscall אנחנו צריכים לפני שנחזור להחזיר את ה CPL ל 3 (רמת המשתמש) כדי שלא ניתן למשתמש הרשאה לבצע פעולות privileged של מערכת ההפעלה (אז לא יהיה נחוץ ל syscall בכלל בגלל שהמשתמש יהיה לו את כל ההרשאות לבצע כל פעולה)

המשך השאלה בעמוד הבא

סעיף 6 (10 נקודות)

עדן שיתף מחוכמתו ואמר לניק שיצטרך קריאת מערכת נוספת, sys_defense אשר תגן עליו מהמתקפות אש של זאקום, אך שתקרא לאחר כל שימוש של קריאת המערכת הראשונה שיצר sys_attack. כלומר sys_defense תבוא אחרי כל קריאה של sys_attack ואף פעם לא בנפרד. בנוסף, שתיהן מגיעות תמיד באותו הסדר. כיצד עליהם לבצע זאת בצורה היעילה ביותר? נמקו בעזרת המידע הנלמד בקורס.

אנחנו יכולים לעשות פונקציית מעטפת כך שקודם אנחנו מגבים את כל הרגיסטרים הנחוצים ואז אנחנו מעברים את המידע לרגיסטרים ואת מספר השירות ל rax של sys_attack ואז מעבירים את המידע ברגיסטרים כולל מספר שירות ב rax של sys_defense ואז חוזרים לקוד על ידי פקודת ret רגילה , וכך שבל פעם שאנחנו רוצים לקרוא sys_attack תבוא מייד אחריה sys_defence

בהצלחה!