Operating systems- Final work:

Q1:

0) סעיף התחלתי- לימוד והבנת הכלים. אסביר על הכלים במילותיי:

Explain for CLI Tools:

Nm- list symbols from object files.

print function's name and global variables from object files, nm can be running with flags for more options.

מדפיס את שמות הפונקציות והמשתנים הגלובליים מקובץ ה object , יכול לקבל דגלים לביצוע אפשרויות נוספות.

Size- list section size and total size.

Size CLI tool shows each section's size in bytes-

.text , .data , .bss , and the total size in decimal and hexadecimal formats.

The tool can be running with flags, for example: -o , for octal format.

הכלי size מראה את גודל ה sections והגודל הכללי בבתים, בפורמט דצימלי, הקסהדצימלי (וניתן בעזרת דגלים גם לקבל אוקטלי או אפשרויות נוספות).

Objdump- display information from object files (one or more).

The objdump tool must be running with flags. The tool sends the detailed information about the obj-file to the stdout, and shows sections, symbol table (in which segment each symbol is in), etc.. objdump is like "inclusion" of size tool.

הכלי objdump חייב לקבל לפחות דגל אחד בכדי להפיק תוצאות, ובעזרת הדגלים הנכונים ייתן מידע על קובץ ה object הרצוי. המידע יישלח ישירות למסך דרך stdout . הכלי הוא מעין פירוט של size ומוסיף בתוכו גם טבלת סימנים וכ'ו.

(1+2 סעיפי המימוש וההסברים:

בקובץ ה C שהוספתי, רשמתי את תשובותיי במקומות המתאימים לפי הוראות המטלה. כאן אפרט ואסביר:

The answers:

Line	The questions + answers
5	`char globBuf[65536];` /* 1. Where is allocated? */ **ANSWER: .bss , uninitialized data.** משתנה גלובלי שלא אותחל- נמצא ב .bss section ומכיל ערכי זבל (או אפסים) .
6	`int primes[] = { 2, 3, 5, 7 };` /* 2. Where is allocated? */ **ANSWER: .data** משתנה גלובלי שאותחל- ידוע בזמן קומפילציה נמצא ב.data section
9	`square(int x)` /* 3. Where is allocated? */ **ANSWER: .text** כל קוד נמצא ב .text segment ופונקציה היא קוד.
11	`int result;` /* 4. Where is allocated? */ **ANSWER: stack – declared in function.**
14	`return result;` /* 5. How the return value is passed? */ **ANSWER: by EAX Register.** רשמתי הסבר מפורט מאוד אחרי הטבלה הנ"ל.
18	`doCalc(int val)` /* 6. Where is allocated? */ **ANSWER: .text** כל קוד נמצא ב .text segment ופונקציה היא קוד.
23	`int t;` /* 7. Where is allocated? */ **ANSWER: stack- declared in function.** כל משתנה שמוצהר בתוך פונקציה- על המחסנית.
31	`main(int argc, char* argv[])` /* Where is allocated? */ **ANSWER:.text** כל קוד נמצא ב .text segment ופונקציה היא קוד.
33	`static int key = 9973;` /* Where is allocated? */ **ANSWER: .data** משתנה סטטי שאותחל ידוע בזמן קומפילציה ונמצא ב .data section
34	`static char mbuf[10240000];` /* Where is allocated? */ **ANSWER: .bss, uninitialized data.** משתנה סטטי שלא אותחל- נמצא ב .bss section ומכיל ערכי זבל (או אפסים) .
35	`char* p;` /* Where is allocated? */ **ANSWER: stack- declared new pointer.** אותחל משתנה חדש מסוג pointer על המחסנית. (משתנה שמוצהר בתוך פונקציית ה main - על המחסנית. אלא אם כן קיים malloc\new שיוצר גם משתנה ב Heap).

Explain for the answers:

Picture for question 5: objdump output

My CPU architecture is x86-64

```
process_layout_q.o:     file format elf64-x86-64
process_layout_q.o
architecture: i386:x86-64, flags 0x00000011:
HAS_RELOC, HAS_SYMS
start address 0x0000000000000000
```

And the result variable that returned in line 14, returned by "eax" register.

I will explain:

CPU architecture registers for return value is:

8 bit - al/ah, 16 bit – ax, 32 bit – eax, 64 bit – rax.

My CPU is 64 bit, and he has the 32bit's registers.

When the variable\data is 32 bit (in our case- int) the value is stored in EAX and return by this register. In another case- that we will declare "long long" variable- it will use RAX register. ("result" variable declared int in line 11).

ארכיטקטורת המעבד שלי הינה 64 ביט.

מעבד מסוג זה מכיל registers (אוגרים) מסוג 64 ביט, 32 ביט, 16 ביט, 8 ביט.

שמות האוגרים מפורטים מלעיל.

כאשר מתקבל משתנה\מידע מסוג 32 ביט- הוא מוכנס לתוך האוגר המתאים, במקרה שלנו- EAX שתפקידו להכיל ולהחזיר את הערך הרצוי ("result" שהוצהר כמשתנה מסוג int) . במקרה אחר, בו היה שימוש במשתנה מסוג של 64 ביט, לדוגמה כמו long long, הערך היה מוחזר דרך האוגר "RAX" שתומך ב64 ביט.

- צירפתי תמונה שמראה את "העבודה" באסמבלי-

```
omri@omri-VirtualBox:~/Desktop/fwork_206265233/q_1$ objdump -d process_layout_q.o

process_layout_q.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <square>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
   5:   48 89 e5                mov    %rsp,%rbp
   8:   89 7d ec                mov    %edi,-0x14(%rbp)
   b:   8b 45 ec                mov    -0x14(%rbp),%eax
   e:   0f af c0                imul   %eax,%eax
  11:   89 45 fc                mov    %eax,-0x4(%rbp)
  14:   8b 45 fc                mov    -0x4(%rbp),%eax
  17:   5d                      pop    %rbp
  18:   c3                      retq
```

Picture for the other questions, that shows the sections:

```
SYMBOL TABLE:
0000000000000000 l    df *ABS*  0000000000000000 process_layout_q.c
0000000000000000 l    d  .text  0000000000000000 .text
0000000000000000 l    d  .data  0000000000000000 .data
0000000000000000 l    d  .bss   0000000000000000 .bss
0000000000000000 l     F .text  0000000000000019 square
0000000000000000 l    d  .rodata        0000000000000000 .rodata
0000000000000019 l     F .text  0000000000000065 doCalc
0000000000000010 l     O .data  0000000000000004 key.2841
0000000000000000 l     O .bss   00000000009c4000 mbuf.2842
0000000000000000 l    d  .note.GNU-stack        0000000000000000 .note.GNU-stack
0000000000000000 l    d  .note.gnu.property     0000000000000000 .note.gnu.property
0000000000000000 l    d  .eh_frame      0000000000000000 .eh_frame
0000000000000000 l    d  .comment       0000000000000000 .comment
0000000000010000       O *COM*  0000000000000020 globBuf
0000000000000000 g     O .data  0000000000000010 primes
0000000000000000         *UND*  0000000000000000 _GLOBAL_OFFSET_TABLE_
0000000000000000         *UND*  0000000000000000 printf
000000000000007e g     F .text  000000000000002a main
0000000000000000         *UND*  0000000000000000 exit
```

- The explanation for each question is in the table.

Pictures for the Stack variables:

Result is printed by "info locals" so that shows that is a local variable- and is allocated in the stack.

```
Breakpoint 1, main (argc=0, argv=0x7fffffffe010) at process_layout_q.c:32
32      {
(gdb) s
38          doCalc(9973);
(gdb) info locals
mbuf = <error reading variable mbuf (value requires 10240000 bytes, which is more tha
n max-value-size)>
p = <optimized out>
(gdb) s
doCalc (val=21845) at process_layout_q.c:19
19      {
(gdb) info locals
No locals.
(gdb) s
20          printf("The square of %d is %d\n", val, square(val));
(gdb) info locals
No locals.
(gdb) s
square (x=0) at process_layout_q.c:10
10      {
(gdb) info locals
result = 32767
(gdb)
```

P is printed by "info locals" so that shows that is a local variable- and is allocated in the stack.

```
(gdb) run
Starting program: /home/omri/Desktop/fwork_206265233/q_1/process_layout_q

Breakpoint 1, main (argc=0, argv=0x7fffffffe010) at process_layout_q.c:32
32      {
(gdb) infolocals
Undefined command: "infolocals".  Try "help".
(gdb) info locals
mbuf = <error reading variable mbuf (value requires 10240000 bytes, which is more tha
n max-value-size)>
p = <optimized out>
(gdb)
```

t is printed by "info locals" so that shows that is a local variable- and is allocated in the stack.

```
Breakpoint 5, doCalc (val=9) at process_layout_q.c:25
25              t = val * val * val;
(gdb) info locals
t = 0
(gdb)
```