Operating Systems – 234123

Homework Exercise 3 – Dry

Submitters:

Tom Guy

315155671

tom.guy@campus.technion.ac.il

Nikita Ivanov

322245101

Nikitaiv@campus.technion.ac.il

שאלה 1 - זיכרון :

עדן, זמרת פופולרית, סבלה ממחסור בזיכרון פיזי במחשב שלה (בעל מעבד IA-32 וזיכרון פיזי בגודל 4GB) ולכן הציעה תכן חדש של מעבד המרחיב את מרחב הזיכרון הפיזי מ-32 ל-40 ביט. כתוצאה מכך, במימוש של עדן יש שלוש רמות תרגום בטבלת הדפים.

שאר נתוני המעבד של עדן זהים לאלו של מעבד IA-32, כלומר נתוני המערכת החדשה הם:

רוחב כתובת וירטואלית	32bit
רוחב כתובת פיזית	40bit
גודל דף/מסגרת/מגירה	4KB
גודל מסגרת של טבלת דפים (בכל הרמות)	4KB
מספר ביטים לדגלים והרשאות בכל כניסה בטבלת הדפים	12bit

סעיף 1

בהנחה שגודל כניסה בטבלת הדפים מעוגל למעלה לחזקה שלמה של 2, מהו אופן חלוקת הכתובת הוירטואלית לשדות בתהליך תרגום כתובות (page walk)? **נמקי.**

	offset	index1	index2	index3
א.	12	9	9	2
ב.	10	10	10	2
ג.	12	10	9	1
ד.	10	9	9	4
ה.	12	10	10	2
ו.	10	9	9	2

נימוק:

תשובה:

ראשית, מכך שגודל הדף נשאר זהה – 4KB, גם החישוב של מציאת offset בתוך הדף יישאר זהה ולכן דרושים 12 ביטים למציאת מיקום הכתובת הפיזית המבוקשת בתוך הדף. כעת נבין את ההיררכיה הרצויה על מנת לתמוך בכתובות פיזיות בגודל 40 ביטים. גודל כל כתובת מעוגל לחזקה שלמה של 2 ולכן יהיה 64 ביטים. גודל כל page table הוא כגודל דף – 4KB ומכך שכל כתובת בו היא 64 ביטים = 8B יש $\frac{2^{12}}{2^3} = 2^9$ PTE's ולכן נצטרך 9 ביטים כדי למפות כל PTE ב-page table. יש לנו מספיק ביטים כדי לעשות 2 *page table* כאלה ול-*page table* האחרון יישארו 2 ביטים.

סעיף 2

לבעלה של עדן, שוקי, אין שום תואר מהטכניון, ולמרות זאת הוא הבחין כי המימוש של עדן בזבזני בגלל שגודל הכניסות בטבלת הדפים מעוגל למעלה לחזקה שלמה של 2.

מהו הגודל המינימלי האפשרי של כניסה בטבלת הדפים אם לא מעגלים אותו למעלה? **נמקי.**

a) 3 בתים
b) 4 בתים
c) 5 בתים
d) 6 בתים
e) 7 בתים

f) אף תשובה לא נכונה

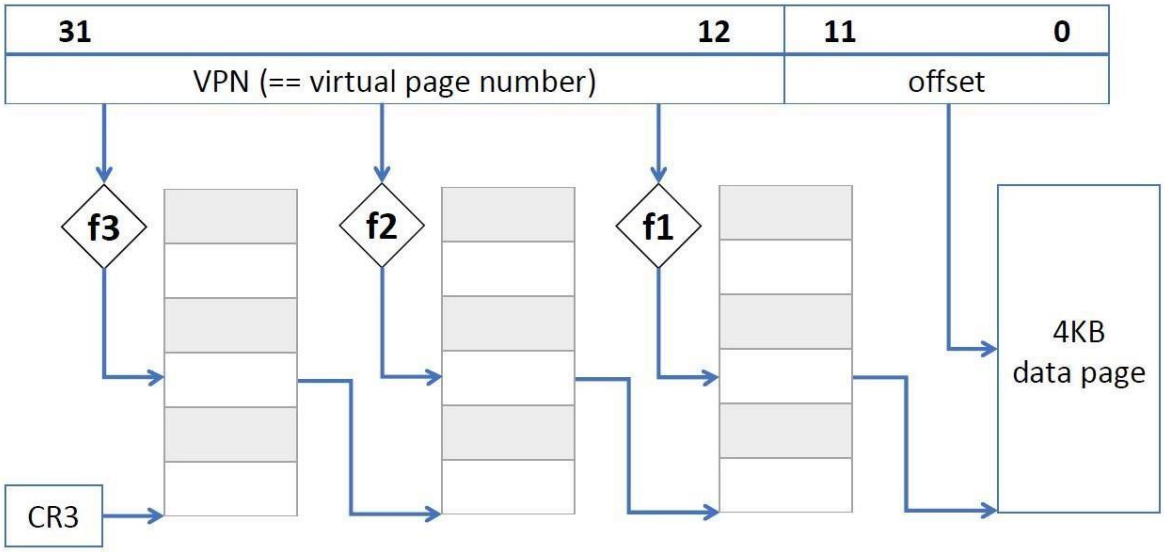
<u>נימוק:</u>

<u>5B = 40b ולכן כל כניסה בטבלת הדפים שמכילה 12 ביטים של דגלים ו- 28 (=40-12, 12 ביטים של offset), ביטים למיפוי כתובת יכולה להישמר בלפחות חמישה בייטים.</u>

<u>סעיף 3</u>

בהמשך לסעיף הקודם, שוקי (בעלה של עדן) הציע מימוש חדש לטבלת הדפים שבו כל כניסה בטבלת הדפים (בכל הרמות) היא בגודל המינימלי מהסעיף הקודם. במימוש של שוקי, כמו במימוש המקורי של טבלת הדפים במעבדי אינטל, דפים סמוכים בזיכרון הווירטואלי נשמרים בכניסות סמוכות בטבלת הדפים. שוקי הבחין שבמימוש החדש הכתובת הווירטואלית אינה מתפרקת לשדות של אינדקסים ויש צורך בחישובים מורכבים על מנת למצוא את האינדקס המתאים בכל טבלה (כלומר בכל רמה בעץ). להלן שרטוט

הממחיש את אופן התרגום:



בשרטוט רואים שלוש פונקציות (f1, f2, f3) המקבלות את מספר הדף הווירטואלי VPN ומחזירות, בהתאמה, שלושה אינדקסים לשלושת הרמות בטבלת הדפים.

בכל הסעיפים הבאים, הפעולות חלוקה "\" ומודולו "%" הן פעולות בשלמים. למשל:

1024/819=1

1024%819=205

מהי הפונקציה f1? **<u>נמקי</u>**.

a. $f_1(vpn) = vpn/819$
b. $f_1(vpn) = vpn\%819$
c. $f_1(vpn) = (vpn/819)\%819$
d. $f_1(vpn) = (vpn/819)/819$
e. $f_1(vpn) = (vpn\%819)/819$
f. $f_1(vpn) = ((vpn/819)\%819)/819$

<u>תשובה: לסעיף הנוכחי ולשניים אחריו</u>

<u>מסעיף קודם, ראינו כי גודלה של כל כתובת (כניסה בטבלה) הוא 5 בייטים ולכן יש סה"כ $4096/5 = 819.2$ כתובות אפשריות בכל דף – כמובן שנשתמש במספר שלם של כתובות ולכן יש סה"כ 819 כתובות בדף. את המיפוי עבור כתובת וירטואלית כללית נבצע בצורה הבאה:</u>

- <u>נציג את הכתובת בצורה הבאה: $a * 819^2 + b * 819 + c$ כאשר $0 \le a, b, c \le 818$</u>

- המיפוי לפונקציות יהיה:
  - $f1(vpn) = c = vpn\%819$
  - $f2(vpn) = b = (vpn/819)\%819$
  - $f3(vpn) = a = (vpn/819)/819$

סעיף 4 מהי הפונקציה f2? **נמקי**.

a. $f_2(vpn) = vpn/819$
b. $f_2(vpn) = vpn\%819$
c. $f_2(vpn) = (vpn/819)\%819$
d. $f_2(vpn) = (vpn/819)/819$
e. $f_2(vpn) = (vpn\%819)/819$
f. $f_2(vpn) = ((vpn/819)\%819)/819$

תשובה:

ענינו גם על סעיף זה מעלה.

סעיף 5 מהי הפונקציה f3? **נמקי .**

a. $f_3(vpn) = vpn/819$
b. $f_3(vpn) = vpn\%819$
c. $f_3(vpn) = (vpn/819)\%819$
d. $f_3(vpn) = (vpn/819)/819$
e. $f_3(vpn) = (vpn\%819)/819$
f. $f_3(vpn) = ((vpn/819)\%819)/819$

תשובה:

ענינו גם על סעיף זה מעלה.

סעיף 6

מה היתרון של המערכת שהציע שוקי על פני המערכת שהציעה עדן? **נמקי.**

a) מיפוי של מרחב זיכרון וירטואלי גדול יותר.
b) מיפוי של מרחב זיכרון פיזי גדול יותר.
c) ה-TLB אפקטיבי יותר בגלל שהוא מכסה יותר זיכרון.
d) טבלאות הדפים של תהליכי משתמש תופסות נפח קטן יותר בזיכרון.
e) פחות פרגמנטציה חיצונית, כלומר יותר זיכרון רציף.
f) אף תשובה אינה נכונה.

נימוק:

במימוש של עדן, כל טבלת דפים הכילה 512 PTEs' לכל היותר (עבור $\frac{4096}{8} = 512$) בעוד במימוש של שוקי כל טבלה מכילה 819 PTEs'.

סעיף 7

לכל אחד מהסעיפים הבאים, סמני האם התשובה נכונה או לא, **ונמקי**.

הגדרה: מצביע **חוקי** הינו כזה שמצביע על כתובת בזיכרון שנמצאת בתוך אזור המוקצה בשביל התהליך\חוט גרעין\.. שמחזיק את המצביע.

1) אם למחשב יש 16GB של **RAM**, יתכן שלתוכנית **משתמש** כלשהי יש מצביע חוקי לכתובת הגדולה מ- 0x200000001 (בסיס HEX).

רמז: האם הכתובת ששמורה בתוך מצביע היא פיזית או ווירטואלית ?

תשובה: נכון \ לא נכון.

נימוק:

הכתובת היא מצביע בתוכנית - כתובת וירטואלית ולכן יתכן בהחלט כי המשתמש יקבל אותה בתור מצביע בעוד שהיא לא קיימת בזיכרון הפיזי, אך היא תמופה לכתובת פיזית הקטנה מ-$0x200000001$.

2) אם למחשב יש 16GB של זיכרון **ווירטואלי**, יתכן שיש מצביע חוקי (לא בהכרח לתהליך משתמש) לכתובת הגדולה מ 16,000,000,000 (בסיס דצימלי).

תשובה: נכון \ לא נכון.

נימוק:

ייתכן בהחלט. לדוגמה, אם גודל כתובת פיזית גדול מגודל כתובת וירטואלית (יש יותר זיכרון פיזי מוירטואלי), כמו לדוגמה בסעיף א' (רק עם גודל כתובת שונה) יתכן כי הכתובות הגדולות מ-16,000,000,000 קיימות ותהליכי גרעין שנגישים לכתובות בצורה ישירה ללא זיכרון וירטואלי ייגשו אליהן בצורה חוקית כחלק מריצת תהליך.

3) בהינתן מחשב גודל דף = גודל מסגרת = 2KB, יתכן שהכתובת הוויראלית 0x800 ו 0x808 ממפות לכתובת במרחק גדול מ 0x8 אחת מהשנייה בזיכרון הפיזי.

תשובה: נכון \ לא נכון.

נימוק:

נשים לב כי כעת גודל הדף הוא $2KB$ ולכן גודל ה-$offset$ חייב להיות 11 ביטים.
עבור $0x800$ ועבור $0x808$ כל הביטים אחרי הביט ה-11 שווים ורק ה-$offset$ ביניהם שונה. לכן הם ממופים לאותו **דף**.
בגלל שההפרש ב-$offset$ הוא 8, ההפרש בין שתי הכתובות הפיזיות הוא בדיוק 8.

4) מנגנון הזיכרון הווירטואלי (כמו שהוא) מאפשר לנו לדמות **דיסק** (אחסון עמיד) גדול יותר ממה שיש לנו בפועל.

תשובה: נכון \ לא נכון.

נימוק:
נשים לב כי אמנם חלק מהמטרה של זיכרון וירטואלי הוא לדמות שמירת זיכרון רב יותר ממה שיש למחשב לתת, אבל זה לא אפשרי לאחסן יותר מידע מכמות הזיכרון הפיזי שיש לנו ולכן לא יווצר מצב ששמור לנו יותר מידע בזכרון הווירטואלי מאשר כמות הזיכרון הפיזי (ושמירת חלק מהמידע על ה-RAM לא נחשב כדימוי לדיסק גדול יותר).

שאלה 2 – ניהול זיכרון :

שאלה זו כתובה באנגלית מאחר ושהיא מכילה לא מעט מושגים ושמות אשר קל יותר לבטאם באנגלית. נא לפתור אותה באיזה שפה שתרצנה\ו.

In the wet part of this homework, you implemented an interface that manages dynamic memory in for a process.

In this part of the homework, you will analyze the existing `malloc()` (from `<cstdlib>`) while learning about some new Linux tools.

NOTE: Do NOT submit code you write in this homework with your wet submission. Simply copy your code to your dry submission file, wherever requested.

<u>Section 1:</u>

1. Look up the "strace" utility online, read a little bit, and try to use it yourself by running `` `strace ls` `` in your OS terminal. Finally, explain here in a few words what it does.

   <u>The "strace" utility collects all the system calls the specified command used while running, and returns them out after the command exists.</u>

2. Write a simple program in C that receives a number "x" from the command line and allocates (using `malloc()`) a block of memory that is "x" bytes long. You can assume there's always one input it will always be a positive integer. Run strace with your compiled program.
   Finally, attach the code of the program and a screenshot of the output of running strace with your compiled program below

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    if (argc != 2)
    {
      printf("Usage: %s <number> :(\n", argv[0]);
    }
    int x = atoi(argv[1]);
    malloc(x);

    return 0;
}
```

```
student@pc:~/Desktop$ strace ./a.out 8
execve("./a.out", ["./a.out", "8"], 0x7ffd3f789418 /* 46 vars */) = 0
brk(NULL)                               = 0x55bbeb485000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=96204, ...}) = 0
mmap(NULL, 96204, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f1287656000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f1287654000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f1287054000
mprotect(0x7f128723b000, 2097152, PROT_NONE) = 0
mmap(0x7f128743b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f128743b000
mmap(0x7f1287441000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f1287441000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f12876554c0) = 0
mprotect(0x7f128743b000, 16384, PROT_READ) = 0
mprotect(0x55bbea752000, 4096, PROT_READ) = 0
mprotect(0x7f128766e000, 4096, PROT_READ) = 0
munmap(0x7f1287656000, 96204)           = 0
brk(NULL)                               = 0x55bbeb485000
brk(0x55bbeb4a6000)                     = 0x55bbeb4a6000
exit_group(0)                           = ?
+++ exited with 0 +++
student@pc:~/Desktop$
```

3. The output you received from running strace on your program was probably very messy. There's no way to tell which system call was used during the execution of malloc. Suggest a simple addition to your C code, such that you will be able to spot the system call used during the execution of malloc anyway. You're not allowed to add flags to strace. Your change must be made in the C code.

ננצל את העובדה ש-strace מנסה להדפיס את פקודות המערכת שהיא מקבלת ברצף ונכניס שתי פקודות write לפני ואחרי ביצוע ה-malloc ובכך נוכל למצוא בקלות את קריאות המערכת בהם השתמשה התוכנית בעת ביצועו.

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
    if (argc != 2)
    {
      printf("Usage: %s <number> :(\n", argv[0]);
    }
    int x = atoi(argv[1]);
    write(1, "malloc is next", 14);
    malloc(x);
    write(1, "malloc was here", 15);


    return 0;
}
```

```
student@pc:~/Desktop$ strace ./a.out 8
execve("./a.out", ["./a.out", "8"], 0x7ffe76617128 /* 46 vars */) = 0
brk(NULL)                               = 0x564cf2c73000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=96204, ...}) = 0
mmap(NULL, 96204, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd6468dd000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fd6468db000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd6462db000
mprotect(0x7fd6464c2000, 2097152, PROT_NONE) = 0
mmap(0x7fd6466c2000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fd6466c2000
mmap(0x7fd6466c8000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd6466c8000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd6468dc4c0) = 0
mprotect(0x7fd6466c2000, 16384, PROT_READ) = 0
mprotect(0x564cf0ef6000, 4096, PROT_READ) = 0
mprotect(0x7fd6468f5000, 4096, PROT_READ) = 0
munmap(0x7fd6468dd000, 96204)           = 0
write(1, "malloc is next", 14malloc is next)          = 14
brk(NULL)                               = 0x564cf2c73000
brk(0x564cf2c94000)                     = 0x564cf2c94000
write(1, "malloc was here", 15malloc was here)         = 15
exit_group(0)                           = ?
+++ exited with 0 +++
student@pc:~/Desktop$
```

Section 2:

In the wet part of this homework, you wrote/will write a `malloc()` alternative that uses both `sbrk()` and `mmap()`. Your job in this section is to determine which memory functions the `malloc()` function that is included in your stdlib uses.

Hint: Use the program and the tools from the last section to help you out!

1. Which **two** system calls does the stdlib standard `malloc()` use in its implementation? Attach screenshots that prove your answer.

1) brk 2) mmap

```
student@pc:~/Desktop$ strace ./a.out 8989898
execve("./a.out", ["./a.out", "8989898"], 0x7ffc3c8c5c88 /* 46 vars */) = 0
brk(NULL)                               = 0x55aa94393000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=96204, ...}) = 0
mmap(NULL, 96204, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6bb343f000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6bb343d000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6bb2e3d000
mprotect(0x7f6bb3024000, 2097152, PROT_NONE) = 0
mmap(0x7f6bb3224000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f6bb3224000
mmap(0x7f6bb322a000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6bb322a000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f6bb343e4c0) = 0
mprotect(0x7f6bb3224000, 16384, PROT_READ) = 0
mprotect(0x55aa94021000, 4096, PROT_READ) = 0
mprotect(0x7f6bb3457000, 4096, PROT_READ) = 0
munmap(0x7f6bb343f000, 96204)           = 0
write(1, "malloc is next", 14malloc is next)          = 14
brk(NULL)                               = 0x55aa94393000
brk(0x55aa943b4000)                     = 0x55aa943b4000
mmap(NULL, 8990720, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6bb25aa000
write(1, "malloc was here", 15malloc was here)         = 15
exit_group(0)                           = ?
+++ exited with 0 +++
student@pc:~/Desktop$
```

2. Find the **threshold** that malloc uses to transition from using one function to the other. In other words, what is the number of bytes, after which calling malloc with that number, would result in using one system call instead of the other? Attach screenshots that prove your answer.

תשובה: 134537

התשובה הנתונה פה מתייחסת רק למקרה בו ה-malloc שעשינו הוא הראשון בתוכנית וכמות הבייטים היא הראשונה בה נשתמש ב-mmap בנוסף ל-brk. במקרה זה, מכיוון שמערכת ההפעלה דואגת ליישור הכתובת, היא תקצה יותר מקום מהדרוש – כמה שביקשנו ועוד כמות הזיכרון שמערכת ההפעלה מקצה לשימושה בכל בלוק – כמו מה שאנחנו מימשנו בתרגיל. באופן כללי, ה-threshold דינמי ומשתנה במהלך ריצת התוכנית, אך מערכת ההפעלה לא תקצה פחות מ-135168 בייטים בעת שימוש ב-mmap. כל זאת למרות שב-mmap man כתוב כי ה-threshold מוגדר להיות 128KB=131072 דיפולטיבית.

```
student@pc:~/Desktop$ strace ./a.out 134536
execve("./a.out", ["./a.out", "134536"], 0x7ffd886a4598 /* 46 vars */) = 0
brk(NULL)                               = 0x5628ee16e000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=96204, ...}) = 0
mmap(NULL, 96204, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f2bebc2b000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f2bebc29000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f2beb629000
mprotect(0x7f2beb810000, 2097152, PROT_NONE) = 0
mmap(0x7f2beba10000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f2beba10000
mmap(0x7f2beba16000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f2beba16000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f2bebc2a4c0) = 0
mprotect(0x7f2beba10000, 16384, PROT_READ) = 0
mprotect(0x5628eddd5000, 4096, PROT_READ) = 0
mprotect(0x7f2bebc43000, 4096, PROT_READ) = 0
munmap(0x7f2bebc2b000, 96204)           = 0
write(1, "malloc is next", 14malloc is next)          = 14
brk(NULL)                               = 0x5628ee16e000
brk(0x5628ee18f000)                     = 0x5628ee18f000
write(1, "malloc was here", 15malloc was here)         = 15
exit_group(0)                           = ?
+++ exited with 0 +++
student@pc:~/Desktop$
```

```
student@pc:~/Desktop$ strace ./a.out 134537
execve("./a.out", ["./a.out", "134537"], 0x7fff1a7715f8 /* 46 vars */) = 0
brk(NULL)                               = 0x55b626c48000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=96204, ...}) = 0
mmap(NULL, 96204, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f3648630000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\240\35\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030928, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f364862e000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f364802e000
mprotect(0x7f3648215000, 2097152, PROT_NONE) = 0
mmap(0x7f3648415000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f3648415000
mmap(0x7f364841b000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f364841b000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f364862f4c0) = 0
mprotect(0x7f3648415000, 16384, PROT_READ) = 0
mprotect(0x55b625569000, 4096, PROT_READ) = 0
mprotect(0x7f3648648000, 4096, PROT_READ) = 0
munmap(0x7f3648630000, 96204)           = 0
write(1, "malloc is next", 14malloc is next)          = 14
brk(NULL)                               = 0x55b626c48000
brk(0x55b626c69000)                     = 0x55b626c69000
mmap(NULL, 135168, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f364860d000
write(1, "malloc was here", 15malloc was here)         = 15
exit_group(0)                           = ?
+++ exited with 0 +++
student@pc:~/Desktop$
```