Operating Systems – 234123

Homework Exercise 4 – Dry

Dan Sdeor dansdeor@campus.technion.ac.il 209509181

Levi Hurvitz levihorvitz@campus.technion.ac.il 313511602

Teaching Assistant in charge:

Or Keret

Assignment Subjects & Relevant Course material

Virtual Memory & Memory Management

Recitations 10-11, Lectures 8-9

Submission Format

1. Only **typed** submissions in **PDF** format will be accepted. Scanned handwritten submissions will not be graded.
2. The dry part submission must contain a single PDF file named with your student IDs –

   DHW4_123456789_300200100.pdf
3. The submission should contain the following:
   a. The first page should contain the details about the submitters - **Name, ID number and email address**.
   b. Your answers to the dry part questions.
4. Submission is done electronically via the course website, in the **HW4 – Dry** submission box.

Grading

1. **All** question answers must be supplied with a **full explanation**. Most of your grade is determined by your **explanation** and **evident effort**, and not on the absolute correctness of your answer.
2. Remember! your goal is to communicate the knowledge that you have acquired. Full credit will be given only to correct solutions which are **clearly** described. Convoluted and obtuse descriptions will receive low grades.

Questions & Answers

- The Q&A for the exercise will take place at a public forum Piazza **only**. Please **DO NOT** send questions to the private email addresses of the TAs.
- Critical updates about the HW will be published in **pinned** notes in the piazza forum. These notes are **mandatory** and it is your responsibility to be updated.

A number of guidelines to use the forum:

- **Read previous Q&A carefully before asking the question**; repeated questions will probably go without answers
- Be polite, remember that course staff does this as a service for the students
- You're not allowed to post any kind of solution and/or source code in the forum as a hint for other students; In case you feel that you have to discuss such a matter, please come to the reception hour
- When posting questions regarding **hw4**, put them in the **hw4** folder

Late Days

- Please **DO NOT** send postponement requests to the TA responsible for this assignment. Only the **TA in charge** can authorize postponements. In case you need a postponement, please fill out the attached form: https://forms.office.com/r/NLBU1nykjE

שאלה 1 - זיכרון

מוקה קנתה מחשב ישן עם מערכת הפעלה לינוקס וארכיטקטורה דומה לזאת שנלמדה בקורס. הכתובות הוירטואליות הינן בגודל bit-16 והכתובות הפיזיות הינן בגודל bit-32.

החלוקה של הכתובת הוירטואלית היא כזאת:

Virtual Page Number		OFFSET
bit-4	bit-4	bit-8
15 12	11 8	7 0

נתון שגודל דף (PAGE) שווה לגודל מסגרת (FRAME), וגם שווה לגודל כל טבלה בטבלת הדפים (PAGE TABLE).

הערה: בכל סעיף שלא מתייחס לכך באופן מפורש, ניתן להניח שהמחשב הזה תקין.

סעיף 1

כמה דפים יכולים להיות לתהליך לכל היותר? פרט\י את החישוב בקצרה.

תשובה סופית: 256 דפים

נימוק:

גודל דף לפי החלוקה של הכתובת הווירטואלית הוא $2^8 bytes$ שזה 256 בתים. כמו כן לפי הנתון זהו גם הגודל של כל frame. יש לנו 8 ביט שמוקצים לטובת מספור כל הדפים שתהליך יכול לקבל שזה בדיוק 256 דפים. או אם רוצים להיעזר בחישוב אז מרחב הזיכרון הוירטואלי הוא $2^{16} bytes$ ולכן מחלוקה שלו בגודל נקבל 256 דפים.

סעיף 2

טוקיו, בת דודה של מוקה, טוענת שמספר המסגרות בזיכרון הפיזי **שווה** למספר הדפים שחישבתם\ן בסעיף הקודם. מוקה לא מסכימה וטוענת שמספר המסגרות שונה ממספר הדפים.

מי מהן צודקת? ***מוקה \ טוקיו (הקף\י את התשובה הנכונה)***

אם מוקה צודקת, ציין\י את מספר המסגרות. נמק\י את תשובתך בקצרה.

נימוק:

מוקה צודקת. מספר הדפים המקסימלי שתהליך יכול להחזיק הוא 256 וזה נובע מכך שמרחב הכתובות שיש לתהליכים בגודל 2^{16}. אבל מרחב הזיכרון הפיזי שיש למחשב הוא בגודל 2^{32} לכן מספר הframes שיכולים להיות בזיכרון הוא $2^{32-8} = 2^{24} => 16M$ מסגרות.

סעיף 3

טוקיו למדה בקורס מערכות הפעלה שהרגיסטר CR3 מכיל את הכתובת לשורש טבלת הדפים של התהליך הרץ. טוקיו טוענת שהכתובת השמורה ב CR3 הינה כתובת פיזית, אבל מוקה טוענת שהיא וירטואלית.

מי מהן צודקת? ***מוקה \ טוקיו (הקף\י את התשובה הנכונה)***

מה הבעיה העיקרית באופציה הלא נכונה?

נימוק:

טוקיו צודקת. כדי שנוכל לגשת לכתובת פיזית מסיומת, אנו חייבים קודם כל לתרגם את הכתובת הווירטואלית שממופה אליה לאותה כתובת פיזית באמצעות טבלת הדפים (תחת ההנחה שזו הפעם הראשונה שניגשנו לכתובת והיא עדיין לא שמורה בTLB cache) אבל בשביל למצוא את שורש טבלת הדפים אנחנו צריכים את הכתובת הפיזית שלה (במידה וcr3 מכיל כתובת וירטואלית) וכדי למצוא כתובת פיזית צריך לגשת לטבלת הדפים. כלומר, יש לנו כאן מעין מצב של ביצה ותרנגולת שבו אנחנו נמשיך לתרגם כתובות ברקורסיה אינסופית כי אין לנו את האינפורמציה המספקת כדי לפתור את הבעיה של למצוא את השורש הפיזי של טבלת הדפים ללא טבלת הדפים. לכן cr3 יהיה חייב להכיל את הכתובת הפיזית של שורש טבלת הדפים.

סעיף 4

מוקה וטוקיו רוצות לדעת את הגודל של ה PTE, ולכן הן פנו אליך!

מהו הגודל של PTE? מהו הגודל המקסימלי שניתן להקצות לדגלים בכל PTE? פרט\י את החישוב.

הנחה: אין בזבוז בדפי טבלת הדפים (כלומר הם מכוסים במלואם ע"י PTEs)

תשובה סופית: גודל ה PTE: 16 בתים , גודל הדגלים המקסימלי: 13 בתים

נימוק:

גודל דף הוא 256 בתים שזה לפי נתון של השאלה גם הגודל של כל טבלת דפים. במבנה של הכתובת הווירטואלית יש שני אינדקסים לשני טבלאות שונות. גודל כל אינדקס הוא 4 ביט שזה $2^4 = 16\ enteries$. לכן הגודל של כל entry בpage table יהיה $\frac{256}{16} = 16$ בתים. עבור כל רשומה בטבלת דפים ברמה הראשונה: 3 בתים משמשים למציאת הכתובת הפיזית של טבלת הדפים בזיכרון (תחת הנחה שיש alignment של 8 bit עבור טבלאות הדפים). עבור כל עמודה בטבלת הדפים המשנית 3 בתים הולכים למציאת הframe number שהוא הרישא של הכתובת שהסיפא שלה היא הoffset שהוא בגודל בית אחד ולכן $16 - 3 = 13$ בתים יכולים לשמש בתור הדגלים לכל page שהקצנו.

סעיף 5

בהינתן שהרגיסטר CR3 מכיל את הערך xBB000, ונתונות גם **הרישות** (הPTEs הראשונים) של חלק מטבלאות הדפים:

הנחה 1: הערכים המוצגים בטבלאות הבאות הינן **רק** הframe number, כלומר לא מוצגים פה דגלים או ביטים אחרים המשומשים לניהול המערכת.

הנחה 2: **אין** מטמונים מכל סוג (בסעיף זה **בלבד**).

הנחה 3: ללא קשר לתוצאת סעיף 4, הנח\י בסעיף זה שגודל הFN הינו BYTE=8bit1.

index	1	X	3	X	5
	טבלת הדפים שנמצאת בכתובת הפיזית $0xAB00$:	טבלת הדפים שנמצאת בכתובת הפיזית $0xC000$:	טבלת הדפים שנמצאת בכתובת הפיזית $0x8000$:	טבלת הדפים שנמצאת בכתובת הפיזית $0xBB00$:	טבלת הדפים שנמצאת בכתובת הפיזית $0x4000$:
0	$0xAB$	$0x40$	$0xC0$	$0x40$	$0x20$
1	$0xAA$	$0x80$	$0xC4$	$0xC0$	$0x30$
2	$0x50$	$0x48$	$0x65$	$0x80$	$0x31$
3	$0x54$	$0x44$	$0x99$	$0xAB$	$0x32$
...					

א. מה היא הכתובת הפיזית (בבסיס hex) שמתאימה לכתובת הוירטואלית x13250?

ב. כמה גישות לזיכרון יהיו במהלך **התרגום** (לא כולל הגישה למידע הסופי)?

ג. סמנ\י '**X**' מעל הטבלאות שהשתתפו בתרגום.

הערה: ספק\י חישוב מפורט לחלק הראשון, והסבר מתאים לחלק השני.

תזכורת: ספרה של HEX מקודדת ל 4 סיביות, למשל: x1234 = 0001 0010 0011 01000

תשובה סופית: הכתובת הפיזית: 0x4425, מספר הגישות: 2

נימוק:

א. נפרק את הכתובת לפי השיטה בשאלה:
אינדקס ראשי: 1
אינדקס משני: 3
0x25 :Offset
גישה ראשונה תהיה לפי הרגיסטר CR3 לטבלה הראשית שמספרה הוא 4. נבחר באינדקס 1 שמכיל את 0xc0.
נעשה shift ב8 ביט וניגש בפעם השנייה לטבלה שנמצאת בכתובת הפיזית 0xc000 שזו טבלה מספר 2. ניגש לאינדקס 3 בטבלה זו. למספר 0x44 נשרשר את הoffset לקבלת הכתובת הפיזית 0x4425.
סה"כ ביצענו שתי גישות לטבלאות 4 ו2.

ב. כפי שהוסבר 2 גישות.

ג. טבלאות 4 ו2 השתתפו בתרגום.

סעיף 6

מוקה טוענת שהזיכרון **הפיזי** שלה מבוזבז ושהיא לא יכולה לנצל אותו במלואו, אבל טוקיו טוענת שהיא טועה.

מי מהן צודקת? ***מוקה \ טוקיו (הקף\י את התשובה הנכונה)***

אם טוקיו צודקת, תאר\י כיצד ניתן לנצל את מרחב הזיכרון הפיזי במלואו? אחרת, הסבר\י בקצרה.

נימוק:

טוקיו צודקת. המגבלה על גודל הכתובת הווירטואלית היא למעשה מגבלה על מרחב הכתובות המקסימלי שתהליך בודד יכול להחזיק שכן לכל תהליך יש בpcb מצביע לכתובת הפיזית (שיינתן לcr3 בהחלפת הקשר) של שורש טבלת הדפים הראשית המוקדש רק לו שמצביעה לטבלאות דפים משניות ששייכות לו. במערכת ההפעלה עשויים להיות תהליכים רבים שמחזיקים דפים רבים (תחת מגבלה של 256 דפים לתהליך) כך שבסופו של דבר אנו עשויים לנצל את כל הכתובות הפיזיות שעומדות לרשותנו.

סעיף 7

טוקיו טוענת ש**לא** יתכן מחשב שבו הזיכרון **הווירטואלי גדול** יותר **מהפיזי**, מוקה לא הסכימה וטענה זה מצב תקין.

מי מהן צודקת? ***מוקה \ טוקיו (הקף\י את התשובה הנכונה)***

אם טוקיו צודקת אז הסבר\י מדוע, ואם מוקה צודקת תאר\י איך זה אפשרי.

נימוק:

מוקה צודקת. וראינו דוגמה מצוינת לכך בהרצאה. עבור ארכיטקטורת x86-64, אמנם כתובת וירטואלית היא בגודל 64 ביט. אבל מרחב כתובות פיזי כזה גדול בצורה לא פרופורציונלית ביחס לדרישה ולטכנולוגיה הקיימת במחשבים מודרניים ולכן מה שעושים הוא להשתמש רק ב48 ביטים למיפוי כתובות ולביט ה48 עושים sign extension. את אותו הדבר נוכל לעשות גם עבור כתובת פיזית של 32 ביט כאשר הכתובת הוירטואלית יכולה להיות יותר גדולה ופשוט לא נשתמש בכל מרחב הזיכרון הוירטואלי שאותה כתובת יכולה להציע.

סעיף 8

מוקה רוצה לאפשר שימוש בדפים גדולים, בלי ביצוע שינויים למבנה טבלת הדפים (כלומר באופן דומה למנגנון שראינו בתרגולים), איזה גדלים של דפים ניתן לאפשר? נמק\י למה אלו כל האופציות האפשריות.

תשובה סופית: 256B ,4KB

נימוק:

יש לנו חלוקה של הכתובת הווירטואלית לשני אינדקסים לטבלאות דפים וoffset. לכן כדי להגדיל את גודל הדף נוכל להוסיף רק אינדקס אחד כי אם נשתמש בשני אינדקסים אז כבר אין paging והכתובת הווירטואלית היא גם הפיזית. הוספת אינדקס לoffset מגדילה אותו להיות offset של 12 ביט. לכן גודל הדף יהיה $2^{12} = 4KB$.

סעיף 9

איזה מהדפים הבאים **יתכן** שיצטרך הגנה ב COW? הקף\י את התשובות הנכונות (יתכן שיש רק תשובה אחת נכונה)

1. דפים **בלי** הדגל VM_MAYWRITE במתאר אזור הזכירון המתאים.
2. דפים **בלי** הדגל VM_WRITE במתאר אזור הזכירון המתאים.
3. דפים **עם** הדגל VM_SHARE במתאר אזור הזכירון המתאים.
4. דפים **עם** הדגל VM_MAYREAD במתאר אזור הזכירון המתאים.
5. אף אחת מהתשובות אינה נכונה.

הסבר\י בקצרה למה כל תשובה היא נכונה\לא נכונה.

2.דפים בלי הדגל WRITE_VM במתאר אזור הזכירון המתאים.

4.דפים עם הדגל MAYREAD_VM במתאר אזור הזכירון המתאים.

תשובה 1 לא נכונה כי אם הדגל VM_MAYWRITE כבוי אז מנגנון COW לא יפעל על הדף ובמקרה של fork לא יהיה מצב שבו נעתיק דף חדש עבור גישה לדף מאחד התהליכים.

תשובה 2 נכונה כי גם עם הדגל VM_WRITE כבוי, לאחר fork אחד התהליכים יכול לבקש ממערכת ההפעלה לשנות את הרשאות הכתיבה לדף ולנסות לכתוב אליו. במצב הזה מנגנון COW יכנס לפעולה והדף יועתק לדף חדש עבור התהליך שמבצע את הכתיבה.

תשובה 3 לא נכונה כי אזורי זיכרון עם דגל VM_SHARE דלוק הם אזורי זיכרון שמשותפים לאב והבן אחרי fork והכוונה היא שהבן רוצה לכתוב למשל לדף מבלי שיוצר דף חדש כדי שהדף של האבא לא יושפע.

תשובה 4 נכונה כי אין שום משמעות לדגל הזה מהבחינה של האם הדף ישתתף בCoW או לא. הדפים שמשתתפים בCoW הם אלה שהVM_MAYWRITE שלהם דלוק והVM_SHARE שלהם כבוי.

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 the it does.**

<u>Strace משמשת להדפסות/קליטה של כל קריאות המערכת שאותו תהליך שנוצר על ידי הפקודה שהעברנו לstrace עשה ובנוסף מדפיסה/מקליטה את כל הsignals שהתהליך קיבל.</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

3. <u>הקוד שכתבנו:</u>

```c
#include <stdlib.h>

int main(int argc, char const* argv[])
{
    int size = atoi(argv[1]);
    void* addr = malloc(size);
    free(addr);
    return 0;
}
```

פלט הstrace:

```
student@pc:~$ strace ./dry_malloc 16
execve("./dry_malloc", ["./dry_malloc", "16"], 0x7ffda6ec32a8 /* 50 vars */) = 0
brk(NULL)                               = 0x55c713642000
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=101870, ...}) = 0
mmap(NULL, 101870, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff470a47000
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\20\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) = 0x7ff470a45000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff470446000
mprotect(0x7ff47062d000, 2097152, PROT_NONE) = 0
mmap(0x7ff47082d000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e700
0) = 0x7ff47082d000
mmap(0x7ff470833000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0
x7ff470833000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7ff470a464c0) = 0
mprotect(0x7ff47082d000, 16384, PROT_READ) = 0
mprotect(0x55c711ebc000, 4096, PROT_READ) = 0
mprotect(0x7ff470a60000, 4096, PROT_READ) = 0
munmap(0x7ff470a47000, 101870)          = 0
brk(NULL)                               = 0x55c713642000
brk(0x55c713663000)                     = 0x55c713663000
exit_group(0)                           = ?
+++ exited with 0 +++
student@pc:~$
```

4. **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.

5. הוספנו לקוד שלנו הדפסות למסך באמצעות write:

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

#define MALLOC_START ("****** START MALLOC CALL ******\n")

#define MALLOC_END ("****** END MALLOC CALL ******\n")

int main(int argc, char const* argv[])
{
    int size = atoi(argv[1]);
    write(1, MALLOC_START, sizeof(MALLOC_START));
    void* addr = malloc(size);
    write(1, MALLOC_END, sizeof(MALLOC_END));
    free(addr);
    return 0;
}
```

כך שכאשר נסתכל על הoutput של strace נוכל לזהות קריאות מערכת של malloc בין קריאות מערכת write עם המחרוזות התואמות:

```
write(1, "****** START MALLOC CALL ******\n"..., 33****** START MALLOC CALL ******
) = 33
brk(NULL)                               = 0x558b4875e000
brk(0x558b4877f000)                     = 0x558b4877f000
write(1, "****** END MALLOC CALL ******\n\0", 31****** END MALLOC CALL ******
```

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**

Malloc משתמש בקריאות המערכת brk וmmap. כאשר מדובר בהקצאות קטנות הוא ישתמש בbrk אחרת בmmap.
הראינו סעיף 3 קריאה לbrk. עבור קריאה לmmap.

```
student@pc:~$ strace ./dry_malloc 134537
execve("./dry_malloc", ["./dry_malloc", "134537"], 0x7ffc186a54d8 /* 49 vars */) = 0
brk(NULL)                               = 0x55857f668000
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=101870, ...}) = 0
mmap(NULL, 101870, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f53121b5000
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\20\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) = 0x7f53121b3000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5311bb4000
mprotect(0x7f5311d9b000, 2097152, PROT_NONE) = 0
mmap(0x7f5311f9b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f5311f9b000
mmap(0x7f5311fa1000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5311fa1000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f53121b44c0) = 0
mprotect(0x7f5311f9b000, 16384, PROT_READ) = 0
mprotect(0x55857f174000, 4096, PROT_READ) = 0
mprotect(0x7f53121ce000, 4096, PROT_READ) = 0
munmap(0x7f53121b5000, 101870)          = 0
write(1, "****** START MALLOC CALL ******\n"..., 33****** START MALLOC CALL ******
) = 33
brk(NULL)                               = 0x55857f668000
brk(0x55857f689000)                     = 0x55857f689000
mmap(NULL, 135168, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5312192000
write(1, "****** END MALLOC CALL ******\n\0", 31****** END MALLOC CALL ******
) = 31
munmap(0x7f5312192000, 135168)          = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

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.**

<u>134537</u>

<u>החל מהמספר 134537 malloc מתחילה להשתמש גם בקריאת המערכת mmap ולא רק brk.</u>

<u>דוגמה למצב שבו מעבירים לmalloc 134536:</u>

```
student@pc:~$ strace ./dry_malloc 134536
execve("./dry_malloc", ["./dry_malloc", "134536"], 0x7ffcfc7f28f8 /* 49 vars */) = 0
brk(NULL)                               = 0x55bc395ad000
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=101870, ...}) = 0
mmap(NULL, 101870, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fd04a011000
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\20\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) = 0x7fd04a00f000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fd049a10000
mprotect(0x7fd049bf7000, 2097152, PROT_NONE) = 0
mmap(0x7fd049df7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fd049df7000
mmap(0x7fd049dfd000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fd049dfd000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fd04a0104c0) = 0
mprotect(0x7fd049df7000, 16384, PROT_READ) = 0
mprotect(0x55bc37c76000, 4096, PROT_READ) = 0
mprotect(0x7fd04a02a000, 4096, PROT_READ) = 0
munmap(0x7fd04a011000, 101870)          = 0
write(1, "****** START MALLOC CALL ******\n"..., 33****** START MALLOC CALL ******
) = 33
brk(NULL)                               = 0x55bc395ad000
brk(0x55bc395ce000)                     = 0x55bc395ce000
write(1, "****** END MALLOC CALL ******\n\0", 31****** END MALLOC CALL ******
) = 31
exit_group(0)                           = ?
+++ exited with 0 +++
```

<u>כאן אפשר לראות בבירור שלא בוצע mmap.</u>

<u>בשונה מהדוגמה הקודמת:</u>

```
student@pc:~$ strace ./dry_malloc 134537
execve("./dry_malloc", ["./dry_malloc", "134537"], 0x7ffc186a54d8 /* 49 vars */) = 0
brk(NULL)                               = 0x55857f668000
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=101870, ...}) = 0
mmap(NULL, 101870, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f53121b5000
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\20\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) = 0x7f53121b3000
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f5311bb4000
mprotect(0x7f5311d9b000, 2097152, PROT_NONE) = 0
mmap(0x7f5311f9b000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f5311f9b000
mmap(0x7f5311fa1000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5311fa1000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f53121b44c0) = 0
mprotect(0x7f5311f9b000, 16384, PROT_READ) = 0
mprotect(0x55857f174000, 4096, PROT_READ) = 0
mprotect(0x7f53121ce000, 4096, PROT_READ) = 0
munmap(0x7f53121b5000, 101870)          = 0
write(1, "****** START MALLOC CALL ******\n"..., 33****** START MALLOC CALL ******
) = 33
brk(NULL)                               = 0x55857f668000
brk(0x55857f689000)                     = 0x55857f689000
mmap(NULL, 135168, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5312192000
write(1, "****** END MALLOC CALL ******\n\0", 31****** END MALLOC CALL ******
) = 31
munmap(0x7f5312192000, 135168)          = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```