**כמה נמוך אפשר לרדת?**

כמה פעמים כתבתם קוד שמבצע כתיבה או קריאה מקבצים? כנראה שיותר ממה שאתם זוכרים.

אבל כמה פעמים כתבתם קוד באסמבלי שעושה את אותו הדבר? או ביצעתם קריאת מערכת ישירות עבור הכתיבה? או גישה ישירה לדיסק? או דרייבר?

נכון שאנחנו בדרך כלל מעדיפים לעבוד כמה שיותר "גבוה", ובצדק.  
אבל לחקור קצת מה קורה מאחורי הקלעים זה תמיד נחמד 😊

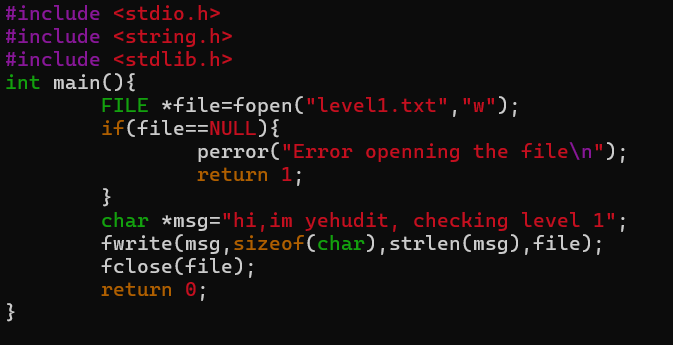
נתחיל בביצוע קוד פשוט, שמבצע שימוש בפונקציות מהספריה הסטנדרטית, לאחר מכן נבצע קוד שלא משתמש

בספריה הסטנדרטית, נמשיך בביצוע הקוד שקורה ישירות לקריאת המערכת המתאימה, באמצעות מספר הקריאה,

נממש קוד באסמבלי, נכתוב דרייבר וננסה לבדוק עד כמה נמוך אפשר לרדת.

**שלב 1: כתיבה לקובץ באמצעות fwrite )רמה גבוהה ביותר)**

בשלב זה, אנו משתמשים בפונקציה **fwrite** מהספרייה הסטנדרטית (libc).  
**fwrite** היא הפונקציה הפשוטה ביותר שמספקת לנו אפשרות לכתוב לקובץ. היא מתבצעת ברמת **השפה הגבוהה C**, כך שאין צורך לדאוג לקריאות מערכת, אוגרים או פרמטרים ברמה נמוכה.



fopen() פותחת את הקובץ level.txt.  
הפרמטw אומר לנו שהקובץ נפתח לכתיבה בלבד, אם הקובץ לא קיים הפונקציה fopen תיצור אותו, ואם הוא קיים- תוכנו יימחק.

הפונקציה מחזירה מצביע ל FILE שמקושר לקובץ הזה.

בפועל fwrite מפעילה את write כדי לכתוב את המחרוזת לקובץ.

בסיום הפונקציה נקרא לfclose() שסוגרת את הקובץ ומשחררת את המשאבים.

נקמפל ונריץ, ואכן נוצר לנו קובץ level1.txt שמכיל בתוכו את המחרוזת המבוקשת:



**שלב 2: שימוש בפונקציה write:**

כעת נרצה לוותר על השימוש בפונקציית **fwrite** ונרצה לגשת ישירות לפונקציית **write**,

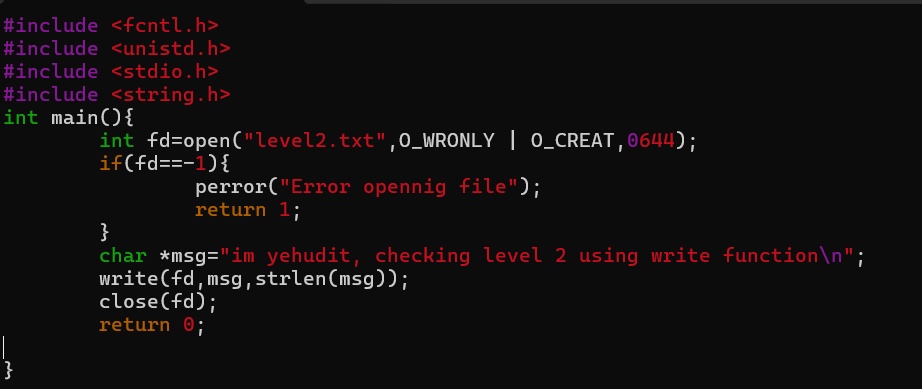
נשים לב שכעת אנחנו לא נייבא את הספריה הסטנדרטית (stdlbi.h) ובמקומה נייבא את הספריה **fcntl**.**h** שתאפשר לנו לגשת ישירות אל write, נשים לב שwrite משתמשת במערכת קבצים posix ומבצעת את קריאת המערכת ישירות.

נשים לב לארגומנטים שאנחנו מעבירים לפונקציה:

fd-file descriptors

O\_WRONLY | O\_CREAT -דגלי הפתיחה של הקובץ.

0644- הרשאות גישה לקובץ.



נקמפל ונריץ, ואכן בקובץ הפלט level2.txt, נקבל את הפלט הבא:

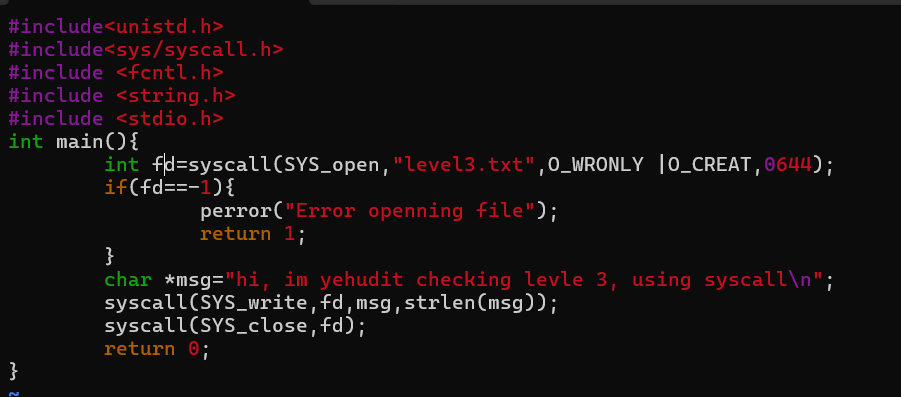


**שלב 3: שימוש בקריאות מערכת עם syscall :**

בשלב זה, אנו **עוקפים את הספרייה הסטנדרטית (libc)** ומתקשרים **ישירות עם הקרנל** בעזרת פונקציית **syscall()**.  
זהו שלב מתקדם יותר שבו איננו זקוקים לפונקציות כמו  **open(), write(), close()** מהספרייה **unistd.h**, אלא פועלים ישירות מול מערכת ההפעלה.

מה זה בעצם syscall:

**syscall** (System Call) היא פונקציה שמאפשרת לנו לבצע **מעבר ישיר מ-User Mode ל-Kernel Mode**.  
כדי לעשות זאת, אנו צריכים להעביר לקרנל כמה פרמטרים שמסבירים **איזו פעולה לבצע**, **על איזה קובץ לפעול**, ו**מה לכתוב או לקרוא**.



**SYS\_open, SYS\_write, SYS\_close** הן מספרי קריאות המערכת עבור הפונקציות open, write ו close.

נעביר גם כן את ההראשות והדגלים המתאימים לפתיחת הקובץ שלנו.

בעזרת syscall ובעזרת מספרי קיראות המערכת תתבצענה הקריאות המתאימות.

נריץ, ונקבל בקובץ הפלט את המחורזת המבוקשת שציפינו לקבל:



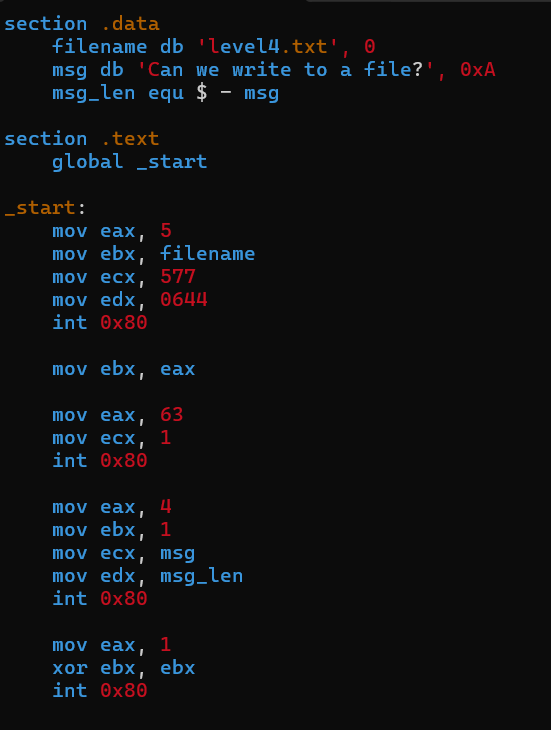
**שלב רביעי, הפעלת interrupt באמצעות int 0x80:**

קריאות מערכת ישירות עם **int 0x80**

הקריאה ל int 0x80 מפעילה **interrupt** שמעביר את השליטה ממצב משתמש למצב קרנל.

הקרנל משתמש באוגרים כדי לדעת בזמן המעבר איזה פעולה לבצע.

|  |  |
| --- | --- |
| מספר קריאת המערכת open.  שם הקובץ.  דגלי הפתיחה של הקובץ.  הרשאות גישה לקובץ.  קריאת המערכת. | mov eax,5  mov ebx, filename  mov acx, 577  mov edx, 0644  int 0x80 |
| הקרנל מבצע את הפעולה שביקשנו, במידה והפעולה הצליחה מספר ה fd מוחזר ברגיסטר aex, לאחר מכן נרצה להעתיק את הערך שנמצא ב eax ל ebx, כדי לעשות בו שימוש בהמשך לקריאה ל: dup2. | mov ebx, eax |
| מספר קריאת המערכת dup2.  STDOUT(FD 1)  המטרה כאן היא להעתיק את הfd של הקובץ, כך שstdout(1) יופנה לקובץ.  קריאת המערכת בפועל. | mov eax, 63  mov ecx, 1  int 0x80 |
| מספר קריאת המערכת write.  FD 1.  מצביע למחרוזת.  מספר הבתים לכתיבה.  קריאת המערכת בפועל. | mov eax, 4  mov ebx, 1  mov ecx, msg  mov edx, msg\_len  int 0x80 |
| מספר קריאת המערכת exit  קוד יציאה 0.  קריאת המערכת לביצוע הפקודה.  הקריאה הזו מעבירה חזרה את השליטה למערכת ההפעלה. | mov eax,1  xor ebx, ebx  int 0x80 |



נקמפל בעזרת הפקודות:



נריץ, ונקבל בקובץ level4.txt את הפלט המבוקש.

מגניב! הצלחנו לגשת ישירות ל int 0x80 ולהפעיל את קריאות המערכת הנדרשות!

**שלב חמישי- כתיבת דרייבר לקרנל (Kernel Module)**

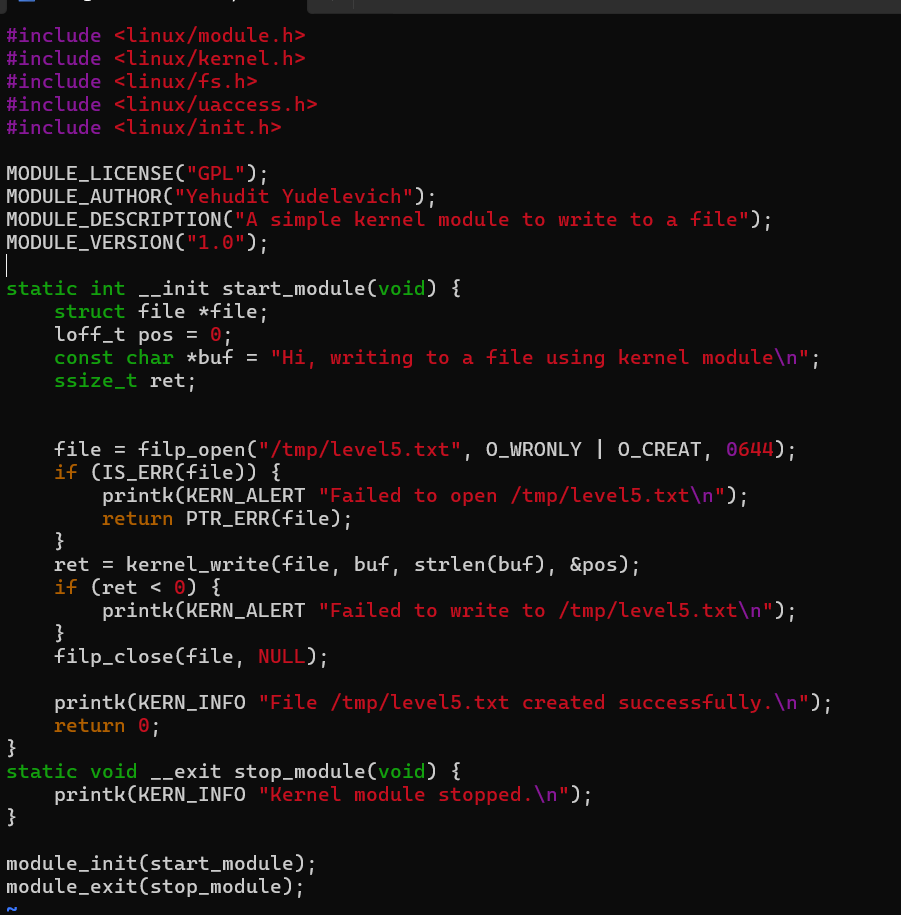
**דרייבר** הוא תוכנה שמקשרת בין הקרנל להתקני חומרה כמו דיסקים, USB וכו.

הדרייבר רץ ב kernel mode ויש לו גישה לחומרה ולזכרון.

**מודול קרנל** הוא קטע קוד שניתן להוסיף או להסיר מהקרנל **בזמן ריצה**, בלי לאתחל את המחשב.  
מודול קרנל מאפשר לנו להוסיף **פונקציונליות חדשה לקרנל**, כמו תמיכה בהתקן חדש, מערכת קבצים חדשה, או פרוטוקול רשת חדש.  
מודול קרנל הוא גם סוג של **דרייבר**, אבל הוא לא בהכרח קשור להתקני חומרה.

המודל קרנל שלנו מאד פשוט ומכיל שני פונקציות עיקריות:  
פונקציית init- קוד שרץ מיד כשהמודול נטען.

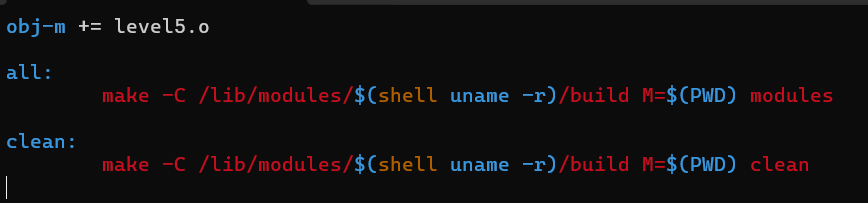
פונקציית exit- קוד שרץ כשהומדול מוסר מהקרנל.

**קוד המודול:**

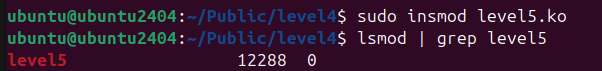
נכתוב קובץ מייקפייל:

נכתוב קובץ makefile ונקמפל.

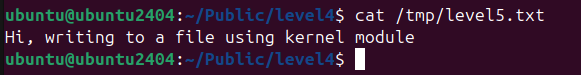
כדי לטעון את המודול פתחתי מכונה וירטואלית עם ubuntu. (לא ניתן באופן רגיל לטעון מודולים דרך windows או wsl)



ונטען את המודול:

****

מעולה, טענו את המודול כעת נותר לנו רק לפתוח את הקובץ שייצרנו ולקוות שאכן הוא נוצר בהצלחה.

****

והצלחנו גם הפעם!