  
  
פרויקט גמר בסייבר

EdenOS

שם התלמיד: עדן פרנקל

ת.ז.: 206964124

מנחים: מיכאל צ'רנובילסקי וגד רוזנטל

בית הספר: אורט גוטמן

כיתה: י"ב 8

**מאי 2015**

**תוכן עניינים**

[תקציר כולל ורציונל הפרויקט 4](#_Toc451551274)

[מבוא ורקע כללי 5](#_Toc451551275)

[תיאור המוצר המוגמר 9](#_Toc451551276)

[שפת התכנות וסביבת העבודה 11](#_Toc451551277)

[ניתוח וניסוח הבעיה האלגוריתמית 12](#_Toc451551278)

[מימוש Bootloader 12](#_Toc451551279)

[ניהול זיכרון 12](#_Toc451551280)

[פסיקות 13](#_Toc451551281)

[ריבוי משימות 13](#_Toc451551282)

[תקשורת עם אמצעי אחסון זיכרון חיצוניים 13](#_Toc451551283)

[מערכת קבצים 13](#_Toc451551284)

[תיאור אלגוריתמים קיימים 14](#_Toc451551285)

[מימוש Bootloader 14](#_Toc451551286)

[ניהול זיכרון 14](#_Toc451551287)

[פסיקות 15](#_Toc451551288)

[ריבוי משימות 15](#_Toc451551289)

[תקשורת עם אמצעי אחסון זיכרון חיצוניים 16](#_Toc451551290)

[מערכת קבצים 16](#_Toc451551291)

[הפתרון הנבחר 18](#_Toc451551292)

[מימוש Bootloader 18](#_Toc451551293)

[ניהול זיכרון 18](#_Toc451551294)

[פסיקות 18](#_Toc451551295)

[ריבוי משימות 19](#_Toc451551296)

[תקשורת עם אמצעי אחסון חיצוניים 20](#_Toc451551297)

[מערכת קבצים 20](#_Toc451551298)

[תיאור המודולים המרכיבים את המוצר 22](#_Toc451551299)

[פירוט המודולים של מערכת ההפעלה 23](#_Toc451551300)

[פירוט מודולי ה-Installer 25](#_Toc451551301)

[פירוט סוגי מבני הנתונים (structs) 26](#_Toc451551302)

[פירוט מבני הנתונים 29](#_Toc451551303)

[פירוט טבלאות לניהול המערכת 30](#_Toc451551304)

[Video Memory 30](#_Toc451551305)

[BIOS Memory Map 30](#_Toc451551306)

[מיפוי הדפים בזיכרון 30](#_Toc451551307)

[טבלת ה-IDT (Interrupt Descriptor Table) 30](#_Toc451551308)

[טבלאות ה-Page Tables 31](#_Toc451551309)

[טבלת ה-Page Directory 31](#_Toc451551310)

[PCI (Peripheral Component Interconnect) 31](#_Toc451551311)

[Frame List (UHCI) 31](#_Toc451551312)

[השוואת העבודה עם פתרונות ויישומים קודמים 32](#_Toc451551313)

[MS-DOS 32](#_Toc451551314)

[UNIX 32](#_Toc451551315)

[Windows (NT) 33](#_Toc451551316)

[Linux 33](#_Toc451551317)

[מדריך התקנה למשתמש (Linux) 34](#_Toc451551318)

[תיאור הממשק למשתמש 37](#_Toc451551319)

[מבט אישי על העבודה ותהליך הפיתוח 39](#_Toc451551320)

[ביבליוגרפיה 41](#_Toc451551321)

[קוד הפרויקט 42](#_Toc451551322)

# תקציר כולל ורציונל הפרויקט

מערכת הפעלה היא מערכת המנהלת את משאבי החומרה והתוכנה של המחשב.

בעולם המודרני התפתחו מספר רב של מערכות הפעלה מסוגים שונים, בעלות מבנה פנימי שונה, ובעלות רציונליים שונים. רוב מערכות ההפעלה המשמשות את המשתמש הממוצע הן מערכות הפעלה גרפיות שמאפשרות נוחות למשתמש, אך גם מצמצמות את האפשרות של המשתמש לגשת ישירות לחומרה ולעסוק בה.

זו תוצאה של אבסטרקציה (Abstraction) — אין זה הגיוני לצפות מכל מתכנת להתעסק ישירות עם החומרה כשהוא רוצה לכתוב כלים פשוטים שמבצעים את אותה פעולה ללא תלות בה. בנוסף, כיום יש כל כך הרבה סוגים מגוונים של רכיבי חומרה שונים, ולסדר ידנית שתוכנה מסוימת תהיה מותאמת לכל סוגי החומרה יהיה מעיק ובזבזני. לכן, מיושמת אבסטרקציה — מערכת ההפעלה מטפלת בחומרה ומתווכת בינה לבין המשתמש. לדוגמה, על מנת להתעסק עם קבצים, לא מעניין את המשתמש אם הוא משתמש ב-USB Flash Drive או ב-SATA Hard Drive; אם המשתמש מעוניין להדפיס למסך, לא מעניינת אותו רזולוציית המסך; אם המשתמש רוצה להריץ תוכנה, זה לא מתפקידו להקצות לתוכנית שלו שטח בזיכרון ולאתחל אותה בעצמו. מערכת ההפעלה לוקחת מקרים פרטיים ומאפשרת למשתמש לפנות אליהם באופן כללי יותר (מכאן — הפשטה).

המטרה שלי בעבודה זו היא ליצור מערכת עובדת שתוכל לנהל ברמה בסיסית את משאבי החומרה והתוכנה הבסיסיים של המחשב, ותוכל לספק למשתמש ממשק טקסטואלי שבו הוא יוכל להזין פקודות ולעשות שימוש בפעולות המערכת.

מערכת ההפעלה שלי תסייע למשתמש להשתמש במשאבי המחשב שלו, עם דגש על גישה מוחלטת, ולעבוד בסביבה וירטואלית שתספק לו שירותים בסיסיים באמצעות פקודות פשוטות.

# מבוא ורקע כללי

העבודה עוסקת בתחום מערכות ההפעלה, ובמימוש של מערכת הפעלה בסיסית על חומרה פיזית. מערכות ההפעלה פותחו על מנת לנהל את משאבי החומרה והתוכנה של המחשב, ועל מנת ליצור אבסטרקציה של החומרה הפיזית על מנת שמשתמש פשוט לא יצטרך להתמודד עם גישה ישירה לחומרה ויוכל ליצור תוכנות כלליות שירוצו על מערכות בעל רכיבי חומרה שונים.

אני, באופן אישי, רציתי להבין יותר לעומק כיצד פועלות מערכות ההפעלה ומבצעות את פעולותיהן השונות, וכיצד עובד המחשב ברמה הבסיסית ביותר. כבר כמה שנים נחשפתי לעולם התכנות ולמדתי שפות Scripting (Python), שפות עיליות (C#, Java) ואף שפות סף (Assembly), אבל בכל אלו השתמשתי בפעולות מוכנות של מערכת ההפעלה שהרחיקו ממני את החומרה ויצרו בשבילי מן אשליה של סביבה וירטואלית שהייתי אמור לקחת כדבר מובן מאליו. כתוצאה מכך הטרידה אותי העובדה שלא ידעתי כיצד זה באמת פועל, ביסוד.

המערכת הבסיסית שאני מתכנן ליצור תנהל באופן בסיסי את משאבי החומרה של המערכת (מסך, מקלדת, זיכרון ועוד) וכן את משאבי התוכנה של המערכת (תהליכים, מערכת קבצים ועוד). היא תציג למשתמש ממשק טקסטואלי בו הוא יוכל להזין פקודות ולהשתמש במערכת ההפעלה הבסיסית לביצוע פעולות שונות.

הפרויקט שלי בעצם מהווה מימוש של מערכת הפעלה השואבת מאפיינים מ-UNIX ו-MS-DOS, אך שונה מכל אחד מהם באופנים רבים.

למערכת מספר רכיבים שונים הפועלים זה עם זה על מנת לממש את המערכת במלואה. שני החלקים העיקריים הם ה-Bootloader וה-Kernel, וה-Kernel עצמה מחולקת גם היא לתתי רכיבים רבים.

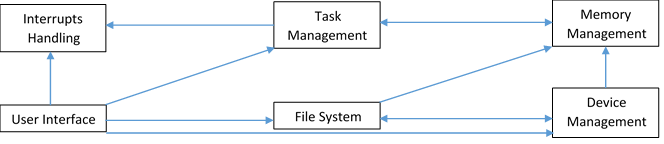
ה-Bootloader הוא הקוד שנטען לזיכרון המחשב ע"י ה-BIOS בעת ה-Boot, מכין את המערכת לריצת ה-Kernel (נכנס ל-Protected Mode (32bit), מאפשר את השימוש בזיכרון מורחב) ומעתיק אותה לזיכרון.

ה-Kernel היא בעצם הקוד הראשי של מערכת ההפעלה ובה נמצאות הפעולות שלה. ה-Kernel מורכבת ממודולים רבים היוצרים מערכת אחת מורכבת. מודולי ה-Kernel יפורטו בהמשך.

בין ה-Bootloader ל-Kernel מחבר קוד קצר הנקרא Bootstrap, המעביר את השליטה מה-Bootloader לפונקציה הראשית של מערכת ההפעלה.

תיאור פשוט של תהליך ה-Boot ושל טעינת ה-Kernel:

תיאור כללי של התקשורת בין חלקי המערכת העיקריים (שיתוף הפעולה ביניהם):



* **Memory Management —** רכיב זה של המערכת מנהל את הזיכרון ומקצה לתהליכים שונים שטח בזיכרון. רכיב זה אינו נעזר בשום רכיב אחר של המערכת, אך רכיבים רבים נעזרים בו על מנת לתפקד כראוי.
* **Task Management —** רכיב זה של המערכת ינהל את התהליכים השונים ויממש Scheduler. רכיב זה נעזר ברכיבי המערכת הבאים:
  + Memory Management — על מנת להקצות ולשחרר שטחים בזיכרון לתהליכים השונים.
  + Interrupts Handling — על מנת לקבל התראות על הזמן שעובר (ממומש באמצעות ה-RTC) למימוש ה-Scheduler.
* **Interrupts Handling —** רכיב זה של המערכת אחראי לקבל ולטפל בפסיקות חומרה ותוכנה של המערכת. רכיב זה אינו נעזר בשום רכיב אחר של המערכת, אך רכיבים רבים נעזרים בו על מנת לתפקד כראוי.
* **Device Management —** רכיב זה של המערכת ינהל את ההתקנים השונים (החיצוניים) שיחוברו למערכת. רכיב זה נעזר ברכיבי המערכת הבאים:
  + File System — על מנת לנתח מערכות קבצים הקיימות על התקנים ניידים ולחברן למערכת הקבצים הראשית של המערכת (Mounting).
  + Memory Management — על מנת להקצות זיכרון בשביל מימוש הממשק לשליטה בהתקנים.
* **File System —** רכיב זה של המערכת ינהל את מערכת הקבצים של המערכת, ויאפשר ביצוע פעולות עליה. רכיב זה נעזר ברכיבי המערכת הבאים:
  + Memory Management — על מנת להקצות מקום בזיכרון ולטעון לשם את התוכן הדרוש.
  + Device Management — על מנת לתקשר עם התקנים ניידים באמצעות אבסטרקציה ולקבל את תוכנם (אם הם מכילים מערכת קבצים מתאימה).
* **User Interface —** רכיב זה של המערכת ינהל את כל התקשורת עם המשתמש באמצעות CLI. רכיב זה נעזר ברכיבי המערכת הבאים:
  + Task Management — על מנת לאפשר למשתמש לנהל את התהליכים הרצים במערכת (לדוגמה, הפסקת ריצה של תהליך).
  + Interrupts Handling — על מנת לאפשר את התקשורת עם המשתמש יש לקבל קלט מהמקלדת שמתקבל באמצעות פסיקות חומרה. בנוסף באמצעות רכיב זה ניתן להציג למשתמש נתונים נוספים, כמו שעה או הזמן שחולף.
  + Device Management — על מנת לאפשר למשתמש לנהל ולראות את ההתקנים המחוברים למערכת.
  + File System — על מנת לאפשר למשתמש לנהל את מערכת הקבצים ולבצע פעולות עליה (יצירת קובץ, מחיקת קובץ ועוד).

# תיאור המוצר המוגמר

שם המוצר: EdenOS

המוצר אמור לנהל ברמה בסיסית את משאבי החומרה והתוכנה הבסיסיים של המחשב, ולספק למשתמש ממשק טקסטואלי שבו הוא יוכל להזין פקודות ולעשות שימוש בפעולות המערכת. פעולות המערכת יכללו ניהול תהליכים והליכים ועבודה עם מערכת קבצים.

המטרה המרכזית של המערכת היא מתן אפשרות נוחה למשתמש לנצל את מערכת המחשב שלו באופן בסיסי (תהליכים, מערכת קבצים ועוד), ומתן גישה מוחלטת לחומרה, בלי חסימות או הרשאות.

השאלות שהמערכת עונה עליהן:

1. **כיצד מאתחלת את עצמה ואת החומרה מערכת ההפעלה במהלך ה-Boot?** המערכת מאתחלת את המערכת את עצמה במהלך ה-Boot: ה-Bootloader נכנס ל-Protected Mode (32Bit), מכין את הסביבה וטוען את ה-Kernel לזיכרון.
2. **כיצד לנהל את הזיכרון במערכת?** המערכת משתמשת ב-bitmap על מנת להקצות pages של זיכרון, ובאפשרויות ה-Paging של המעבד על מנת לממש זיכרון ווירטואלי.
3. **כיצד לנהל את ה-Interrupts, לקבל פסיקות ולנהל אותן כראוי?** המערכת ממפה את מספרי ה-Interrupts, מממשת מבנה נתונים של Interrupt Descriptor Table בזיכרון (טבלה המראה מה יש לבצע במקרה של כל פסיקה), ומממשת ממשק לניהול הפסיקות ולטיפול בהן.
4. **כיצד לממש מערכת עם ריבוי תהליכים על מעבד בודד?** המערכת מממשת Multi-Tasking באמצעות Software Context Switch. היא מאפשרת איתחול, הפעלה ועצירה של ריצת תהליכים על מעבד בודד.
5. **כיצד מקבלים מידע על רכיבים והתקנים במערכת?** המערכת משתמשת ב-PCI (Peripheral Component Interconnect) על מנת לקבל מידע ולנהל את התקני המערכת.
6. **כיצד לגשת לזיכרון של USB Flash Drive?** המערכת מממשת תקשורת עם Universal Serial Buses ופרוטוקולים שונים כגון UHCI (Universal Host Controller Interface) המאפשרים תקשורת עם ההתקנים.
7. **כיצד לנהל מערכת קבצים ולבצע עליה פעולות?** המערכת מממשת ניהול מערכת קבצים, ומאפשרת למשתמש לבצע פעולות עליה.

# שפת התכנות וסביבת העבודה

מערכת ההפעלה נכתבה ב-C וב-Assembly. ה-Bootloader (החלק שנטען ב-Boot ומכין את סביבת העבודה) וה-Bootstrap (החלק המקשר בין ה-Bootloader ל-Kernel) כתובים ב-Assembly, וה-Kernel (מערכת ההפעלה הראשית) כתובה ב-C.

קוד ה-Assembly נכתב ב-Notepad++, ועבר קומפילציה באמצעות GNU Assembler לקובץ מסוג ELF32 Object File.

קוד ה-C נכתב גם הוא ב-Notepad++, ועבר קומפילציה באמצעות GNU Compiler Collection לקובץ מסוג ELF32 Object File.

באמצעות GNU Linker מאחדים את כל קבצי ה-Object ומקשרים ביניהם, והוא פולט את הקובץ הבינארי שייכתב על ה-USB Flash Drive.

על מנת לבנות את הכלים הללו ולהשתמש בהם, השתמשתי ב-Cygwin — שורת פקודה המדמה Terminal של Linux על Windows.

בתחילת העבודה, היה לי צורך בכלי שיכתוב את הקובץ הבינארי ישירות על ה-USB Flash Drive, ולמטרה זו השתמשתי ב-Rufus, כלי חינמי ליצירת Bootable USB Flash Drives.

ה-Installer נכתב ב-Python באמצעות ה-IDE PyCharm בסביבה של Linux (Ubuntu). סביבת ה-Linux התאפשרה באמצעות מכונה וירטואלית אשר רצה בסביבת VMware Player על Windows. ה-Server נכתב (ומורץ) בסביבת Windows, גם כן באמצעות PyCharm.

# ניתוח וניסוח הבעיה האלגוריתמית

במהלך פיתוח המוצר נתקלתי בבעיות אלגוריתמיות רבות:

## מימוש Bootloader

הבעיה הראשונה שנתקלתי בה בפיתוח המוצר היא כתיבת ה-Bootloader. ה-Bootloader צריך להכין את המערכת על מנת שמערכת ההפעלה תוכל לרוץ כראוי, להעתיק אותה לזיכרון, ולעבור אל שלב ה-Bootstrap. בתוך בעיה אלגוריתמית זו ישנן מספר תתי בעיות:

**הכנת המערכת:** כשטוען ה-BIOS את ה-Bootloader, המערכת רצה במצב של 16bit (Real Mode). מערכת ההפעלה שאני רציתי ליצור הייתה מיועד לרוץ במצב של 32bit (Protected Mode), המאפשר גישה לזיכרון מורחב הרבה יותר ואפשרויות תכנותיות מתקדמות יותר.

**הכנת הזיכרון:** כשטוען ה-BIOS את ה-Bootloader, מתאפשרת גישה מוגבלת מאוד לזיכרון, ל-MiB הראשון בלבד. על מנת לאפשר גישה לשאר הזיכרון, יש לאפשר את ה-A20 Gate. דבר זה מאפשר לגשת לזיכרון בכתובות בעלות יותר מ-20 ביטים, דבר שלא מתאפשר באופן אוטומטי.

**Segmentation:** במעבר בין Real Mode (16bit) ל-Protected Mode (32bit), משתנה שיטת הסגמנטציה מכתובות לינאריות ל-Descriptor Table.

**העתקת ה-Kernel לזיכרון:** הייתי צריך להעתיק את קוד מערכת ההפעלה הראשית לזיכרון.

## ניהול זיכרון

בניהול הזיכרון נתקלתי במספר תתי בעיות:

**הקצאת ושחרור דפים (pages) בזיכרון:** הבעיה הבסיסית ביותר שנתקלתי בה בניהול הזיכרון היא כיצד לשמור את הנתונים לגבי הקצאות של דפים בזיכרון.

**זיכרון דינאמי:** בעיה אלגוריתמית נוספת שנתקלתי בה היא כיצד לאפשר למערכת ההפעלה להקצות זיכרון דינאמי (זיכרון הקטן בגודלו מדף, וגודלו אינו קבוע).

**Paging:** בעיה נוספת שהייתה לי האם, ואם כן — כיצד, ליצור מיפוי וירטואלי של הדפים בזיכרון, באמצעות אפשרות של המעבד הנקראת Paging.

## פסיקות

בכל מערכת כיום קיימות פסיקות חומרה ותוכנה המאפשרות את פעילותה התקנית של המערכת. הדילמה שלי הייתה האם כדאי לממש טיפול בפסיקות, וכיצד.

## ריבוי משימות

בעיה אלגוריתמית נוספת היא מימוש ריבוי משימות על מעבד בודד. באמצעות פסיקת ה-RTC (Real-Time Clock) הייתי מסוגל לעקוב אחרי כמה זמן עובר, ובכל פרק זמן מסוים להחליף בין הליכים במערכת, ולהקצות לכל אחד מהם את הזמן הראוי לו, לפי מצבו. השאלה היא, כיצד לשמור את מצבם של ההליכים, כיצד להחליף ביניהם, ובכלל — כיצד צריך להיות בנוי מבנה הנתונים שלהם?

## תקשורת עם אמצעי אחסון זיכרון חיצוניים

הייתי מעוניין שהמערכת שלי תוכל לתקשר עם אמצעי אחסון חיצוניים. הבעיה האלגוריתמית היא אילו דרכים יש לתקשר עם אמצעי אחסון שונים.

## מערכת קבצים

בעיה אלגוריתמית נוספת היא מימוש מערכת הקבצים, על מנת לנתח את הזיכרון השמור על אמצעי האחסון.

# תיאור אלגוריתמים קיימים

## מימוש Bootloader

**הכנת המערכת:** האלגוריתם הקיים למעבר בין מצבים של המעבד הוא שינוי ערכים בחלק מאוגרי המעבד ומעבר לקוד של 32bit.

**הכנת הזיכרון:** ישנן שיטות שונות לבצע זאת, ולא כל חומרה תומכת בכל אחת מהן:

* **Keyboard Controller —** הדרך המסורתית לאפשור ה-A20 Gate היא באמצעות גישה ישירה ל-Keyboard Controller הקיים ברוב המערכות הקיימות, ומאפשר פעולה זו באמצעות תקשורת עם ה-ports שלו.
* **Fast A20 Gate —** ברוב המחשבים החדשים, יש ל-chipset אפשרות בשם FAST A20 המאפשרת אפשור מהיר של ה-A20 Gate.
* **BIOS —** ל-BIOS ישנה פונקציה קיימת לאפשור ה-A20 Gate (INT 15, AX = 0x2401), אך פונקציה זו לא נתמכת בכל המערכות ובחלקן מממשת באופן לא אמין.

**Segmentation:**

האלגוריתם הקיים לשינוי שיטת הסגמנטציה הוא בניית מבנה הנתונים Global Descriptor Table ואת המבנה של המצביע למבנה זה בזיכרון ולשמור מצביע אל המבנה המצביע באחד מאוגרי המעבד.

**העתקת ה-Kernel לזיכרון:**

* **BIOS —** משום שה-Bootloader רץ ב-Real Mode, ניתן לנצל את האפשרות של ה-BIOS להעתיק את הקוד מה-USB Flash Drive לזיכרון ה-RAM.
* **כתיבת Driver ל-USB —** ניתן היה לממש driver בשביל שה-Bootloader יהיה מסוגל להעתיק את מערכת ההפעלה לזיכרון.

## ניהול זיכרון

**הקצאת ושחרור דפים (pages) בזיכרון:**

* bitmap, שהיתרונות שלה הם פשטות מימוש וביטול הקצאה פשוט (סיבוכיות O(1)), אך החיסרון היה מציאת דף פנוי (סיבוכיות O(n)).
* מבני נתונים מקושרים בזיכרון שישמרו את הנתונים על מקומות פנויים, שהיתרון שלה הוא מציאת דף פנוי (סיבוכיות O(1)), אך החסרונות היו ביטול הקצאה (סיבוכיות O(n)) ומורכבות המימוש (בעיקר אם רוצים יעילות גבוהה יותר).

**זיכרון דינאמי:**

* bitmap, בדומה לסעיף הקודם.
* מבני נתונים מקושרים בזיכרון, בדומה לסעיף הקודם.

**Paging:** טבלאות בזיכרון, שאותן מסוגל המעבד לפענח (פורמט מסוים). על מנת לנצל אפשרות זו, יש ליצור את הטבלאות בזיכרון, לקשר ביניהן, לאחסן את המצביע לטבלה הראשית באוגר של המעבד, לאפשר את Paging על ידי שינוי ערכים באוגר מסוים של המעבד, ולספק ממשק מסוים שינהל טבלאות אלו ויוכל לשנות בהן ערכים.

## פסיקות

האלגוריתם הקיים ב-Protected Mode (32bit), בו עבדתי, הוא יצירת מבנה נתונים הנקרא IDT (Interrupt Descriptor Table). מבנה נתונים זה ממפה את הפסיקות השונות ומורה למעבד היכן נמצא את הקוד אותו יש להריץ במקרה של כל פסיקה. המעבד מוצא את ה-entry המתאים לפסיקה שהתקיימה, וקופץ בהתאם למקום המתאים בזיכרון.  
דבר זה הוביל אותי לעוד פיצול באפשרויות מבחינה אלגוריתמית: האם כדאי ש—

* כל entry למקום שונה בזיכרון, שם תטופל הפסיקה. או ש—
* כל entry למקום שונה בזיכרון, שם ישמר מספר הפסיקה, ומשם יקפצו כל הפסיקות לאותה פונקציה שתטפל בכל פסיקה בהתאם.

## ריבוי משימות

הטכניקה להחלפה בין משימות נקראת Context Switching, ויש לה שני מימושים אפשריים — חומרתי ותוכנתי.

* החומרתי מנצל אפשרות קיימת במעבד של שמירת מצב המעבד במבנה נתונים שהמעבד יודע להתנהל איתו באופן אוטומטי — TSS (Task State Segment). יש לממש מבנה נתונים זה לכל הליך, לאחסן Descriptors שלו ב-Descriptor Table, ולהחליף ביניהם במעבר בין הליכים.
* לעומת זאת, לתוכנתי אין קשר לחומרה, והוא ממומש על ידי דחיפת הנתונים של ההליך למחסנית שלו, ושליפתם בפעם הבאה שיגיע תורו.

לגבי המבנה, ישנן מספר אפשרויות:

* תהליכים שלכל אחד מהם יש הליך אחד בלבד. כלומר, אין Threads — ישנה רק רמה אחת של Multitasking.
* לכל תהליך יש מספר הליכים שיכולים לרוץ במקביל.

## תקשורת עם אמצעי אחסון זיכרון חיצוניים

היו לי שתי אפשרויות עיקריות באשר לסוג אמצעי האחסון איתם יכולתי לתקשר:

* דיסקים קשיחים המחוברים ישירות ללוח האם. איתם ניתן לתקשר באמצעות אחרת מווריאציות ממשק ה-ATA (AT Attachment).
* USB Flash Drive, המוכר בשם דיסק-און-קי. תקשורת עם אמצעי אחסון זה היא מורכבת ומסובכת יותר, ובנויה משכבות תקשורת שונות (בדומה לתקשורת ברשת). הסיבה לכך היא שה-Ports של חיבור ה-USB (Universal Serial Ports) נועדו לאפשר תמיכה לסוגים מגוונים של מכשירים (מקלדות, עכברים, התקני אחסון, התקני שמע ועוד), ובעקבות הגיוון הזה, הממשק של ה-Ports האלו הוא כללי ביותר, ויש לממש גם את התקשורת איתו, ודרכו לתקשר עם המכשיר הספציפי.

## מערכת קבצים

ישנן מספר מערכות קבצים אפשריות:

* FAT (File Allocation Table) — מערכת קבצים שפותחה ע"י Microsoft, ונחשבת לפשוטה אך חזקה. בבסיסה עומדת טבלה המכילה entry עבור כל cluster (שטח מסוים של זיכרון), ומכאן מגיע שמה.
* NTFS (New Technology File System) — מערכת קבצים שפותחה ע"י Microsoft והחליפה את מערכת FAT. NTFS אומנם מסובכת יותר למימוש, אך היא מציעה שיפור בביצועים, באמינות, בניצול שטח דיסק ועוד. המערכת משתמש בעץ B+ מקושר.
* יצירת מערכת קבצים משל עצמי. היתרונות הברורים באפשרות זו הם העובדה שיהיה פשוט ומובן לי יותר לממש מערכת קבצים שאני המצאתי, ושאוכל להתאים אותה לצרכים של מערכת ההפעלה שלי. החיסרון הוא העובדה שמערכת ההפעלה שלי ככל הנראה תהיה פחות יעילה ואמינה, לעומת מערכות קבצים קיימות בשוק.

# הפתרון הנבחר

## מימוש Bootloader

**הכנת המערכת:** האלגוריתם הקיים למעבר בין מצבים של המעבד הוא שינוי ערכים בחלק מאוגרי המעבד ומעבר לקוד של 32bit.

**הכנת הזיכרון:** בחרתי באפשרות המשתמשת ב-Keyboard Controller, משום שהיא פשוטה למימוש ועבדה על החומרה שלי (כמו על רוב החומרות כיום).

**Segmentation:** האלגוריתם הקיים לשינוי שיטת הסגמנטציה הוא בניית מבנה הנתונים Global Descriptor Table ואת המבנה של המצביע למבנה זה בזיכרון ולשמור מצביע אל המבנה המצביע באחד מאוגרי המעבד.

**העתקת ה-Kernel לזיכרון:** בחרתי באפשרות של ה-BIOS, משום שמימוש driver ב-Bootloader יהיה מסובך מידי, וכנראה גם כבד מידי, ונעשה בשלב מאוחר יותר ב-Kernel עצמה, כשלא ניתן להשתמש ב-BIOS. ה-BIOS מאפשר פתרון פשוט להעתקה ב-Bootloader.

## ניהול זיכרון

בניהול הזיכרון נתקלתי במספר תתי בעיות:

**הקצאת ושחרור דפים (pages) בזיכרון:** בחרתי ב-bitmap, משום שהמימוש שלה פשוט הרבה יותר ונוח לשימוש, והיעילות סבירה בהחלט.

**זיכרון דינאמי:** בחרתי ב-bitmap גם במקרה הזה, מאותן הסיבות.

**Paging:** בחרתי לנצל אפשרות זו.

## פסיקות

נבחר מימוש ה-IDT. בנוסף, נבחר שיהיה Handler אחד שיטפל בכל הפסיקות, באופן המתאים. הוא בעצם מרכז וקורא לכל הפונקציות המתאימות לטיפול לפסיקות השונות. כאשר מגיע פסיקה, המעבד עובר לקוד שדוחף את מספר הפסיקה (ה-IRQ), ואז קופץ ל-Handler הכללי. הוא אז בודק את מספר הפסיקה וקורא לפונקציה המתאימה.

## ריבוי משימות

לבסוף, בחרתי במימוש התוכנתי, משום שחשבתי שהוא יהיה לי פשוט יותר למימוש, ומשום שזהו המימוש שבו משתמשות רוב מערכות ההפעלה המקצועיות שקיימות היום.

בחרתי לממש ריבוי הליכים בכל תהליך, על מנת שכל תהליך יוכל לממש Multitasking בתוך עצמו.

מבנה הנתונים שבחרתי הוא בעצם רשימה מקושרת מעגלית של כל התהליכים הפעילים, שלכל אחד מהם ישנה רשימה מקושרת סופית של כל ההליכים שלו. ה-Scheduler עובר בין תהליכים, ובתוך כל תהליך נותן לכל הליך שלו את זמן המעבד המגיע לו על פי מצבו, ועובר לתהליך הבא כשנגמר זמנו של ההליך האחרון. לדוגמא:

THREAD1

THREAD2

THREAD1

THREAD2

THREAD1

THREAD2

THREAD1

THREAD2

## תקשורת עם אמצעי אחסון חיצוניים

אף על פי שהמימוש של התקשורת עם הדיסק-און-קי התברר להיות מאוד מסובך, בחרתי לממש אותו. זאת משום שרציתי שמערכת ההפעלה שלי תהיה "ניידת", ותשתמש במשאבי ההתקן ממנו היא מבצעת את ה-Boot.

התקשורת עם ההתקן בנויה בשכבות, כאשר כל אחת נתמכת בקודמת לה:  
(ממוין מנמוכה לגבוהה)

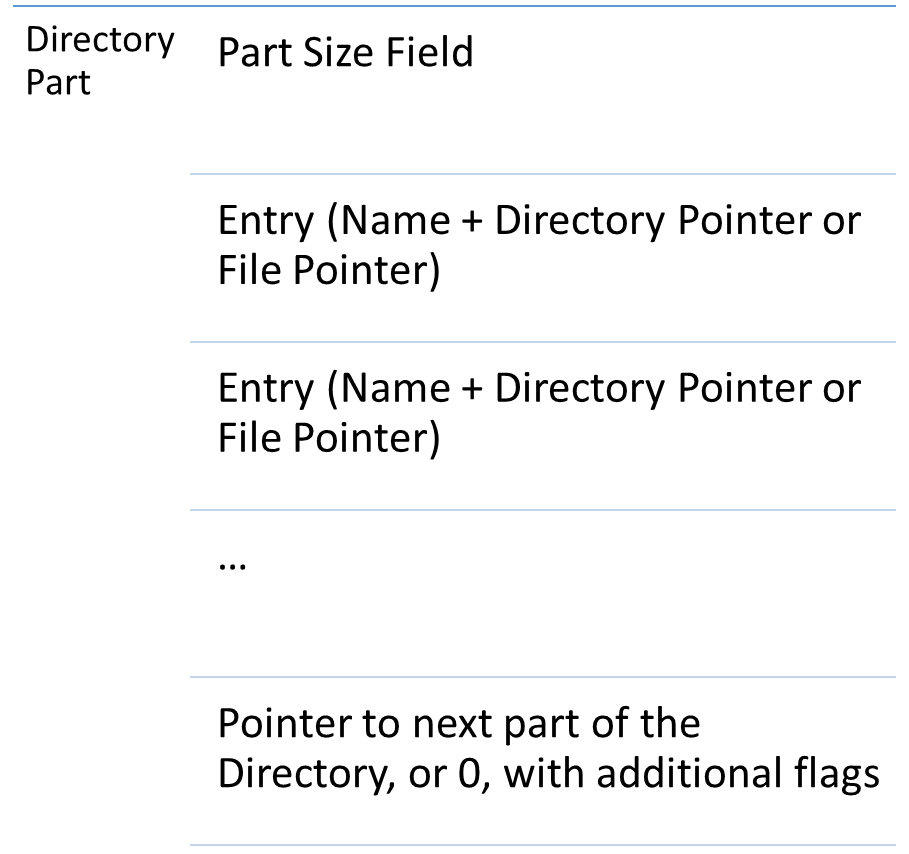
1. **USB Host Controller** — התקשורת עם ה-Host של ה-Serial Port.
2. **USB Mass Storage Device — Bulk Only Transport** — התקשורת עם ההתקן הספציפי והממשק בו הוא תומך.
3. **SCSI** — התקשורת עם ה-Chipset הנמצא בתוך ההתקן ומבצע את הפעולות על הזיכרון.

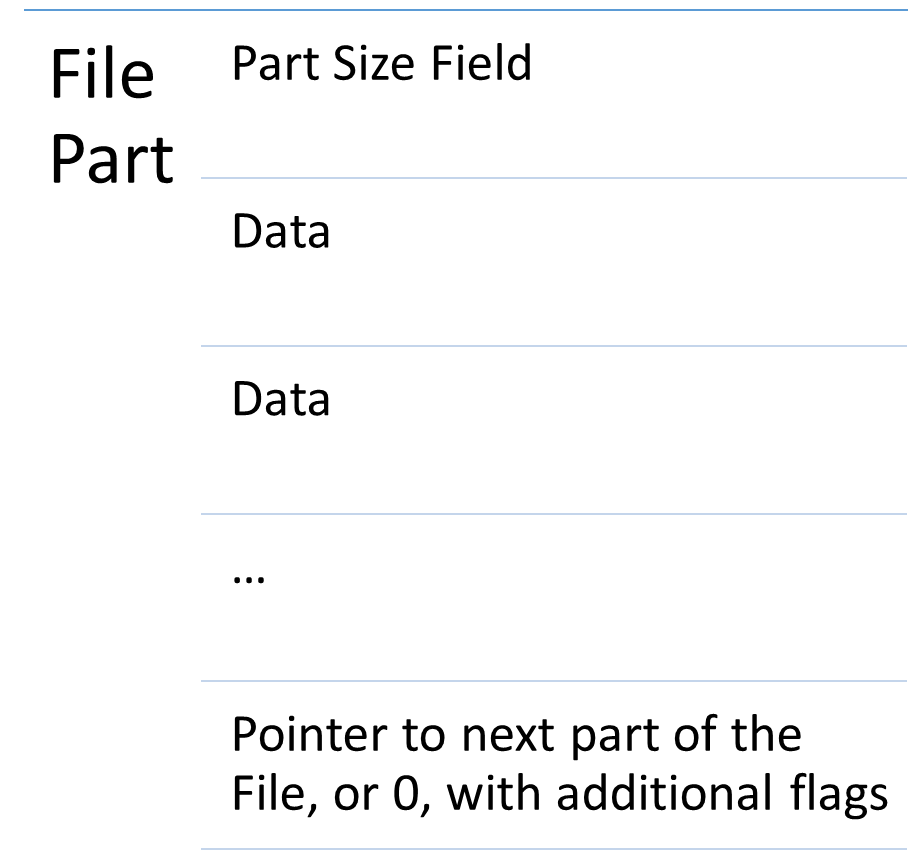
## מערכת קבצים

בחרתי ליצור מערכת קבצי משל עצמי, על מנת שהיא תתאים לצרכים של מערכת ההפעלה שלי.

במערכת קבצים זו, תיקיות יהיו רשימה מקושרת של טבלאות המכילות entries לכל אובייקט בתיקיה, וקבצים יהיו רשימה מקושרת של בלוקי מידע. בבסיס המערכת תהיה תיקיית root.

סרטוט המתאר את מבנה כל אחד מהחלקים במערכת הקבצים:





שמירת המידע על בלוקים פנויים בזיכרון תעשה באמצעות bitmaps. על מנת לשפר את יעילות המערכת, יהיו מספר רמות של bitmaps, שכל אחת תציין האם בחלק מסוים מהרמה הבאה יש בלוק פנוי.

# תיאור המודולים המרכיבים את המוצר

## פירוט המודולים של מערכת ההפעלה

* **Console (console.c) —** מודול זה אחראי לניהול הפלט המוצג על המסך. לאחר ה-boot הוא מאתחל את המסך. הוא מספק פעולות של הדפסת תו או מחרוזת, ותוחם את הטקסט לשטח מוגדר.
* **String (string.c) —** מודול זה מספק פעולות שונות ושימושיות לפעולות על מחרוזות או קטעי זיכרון.
* **Keyboard (keyboard.c) —** מודול זה אחראי על התקשורת עם המקלדת, והוא מספק פעולות לקבלת קלט ממנה. הוא מספק פעולות של קבלת תו או מחרוזת.
* **Interrupts (interrupts32.c) —** מודול זה אחראי לטיפול בפסיקות. הוא מכיל Handler כללי שאליו מופנות הפסיקות, והוא קורא בהתאם לפסיקה לפעולה הנחוצה.
* **IDT (idt32.c) —** מודול זה אחראי על אתחול וניהול טבלת ה-IDT, המכילה descriptors לכל פסיקה, המורים את כתובתו של הקוד שיש להריץ במקרה שהיא מתקיימת. יש בו פעולת אתחול ופעולה המוסיפה entry ל-IDT.
* **Time (time.c) —** מודול זה אחראי למדידת זמן, ובנוסף אחראי על פנייה למודול ה-Processing כל פעם שעבר זמנו של הליך מסוים לרוץ. מודול זה מכיל את ה-Handler לטיפול בפסיקה של ה-RTC (Real-Time Clock), ובאמצעותה הוא יכול למדוד את הזמן שעבר. המודול מודד את הזמן שעבר מאז אתחול המערכת, והוא גם מדפיס ערך זה למסך באופן קבוע.
* **Command Prompt (cmd.c) —** מודול זה מספק את ממשק שורת-הפקודה למשתמש, שבו הוא יכול להפעיל פעולות שבהן תומך המודול. המודול מכיל מבנה נתונים שבו נשמרות הפעולות. בנוסף, הוא מכיל פעולה להוספה של פעולות. המודול משתמש ב-Console Module על מנת לפלוט פלט למסך, וב-Keyboard Module על מנת לקלוט קלט.
* **Memory Management (memory.c) —** מודול זה אחראי על הקצאת ושחרור דפים (4KiB) בזיכרון. הוא מבצע זאת באמצעות bitmap הממוקם בשטח שהוקצה לכך מראש. הוא מספק פעולות להקצאת ושחרור דפים.
* **Dynamic Memory (dmemory.c) —** מודול זה אחראי על הקצאת ושחרור זיכרון דינאמי בזיכרון. הוא מקצה לו דפים בזיכרון, מקשר ביניהם, ובכל אחד מהם יוצר לעצמו bitmap שמורה לו אילו בתים פנויים ואילו לא. בנוסף, בעבור כל הקצאה, הוא שומר גם את מספר הבתים שהוקצו, על מנת להקל על השחרור. הוא מספק פעולות להקצאה ושחרור דינאמיים.
* **Paging (paging.c) —** מודול זה אחראי על יצירת וניהול מבנה הנתונים שיוצר את ההמרה בין דפי זיכרון פיזיים לבין דפי זיכרון וירטואליים. הוא יוצר מבנה זה, מאתחל אותו, ומאפשר ניהול של ה- entriesשלו.
* **Processing (processing.c) —** מודול זה אחראי על ניהול מערכת ה-multitasking של מערכת ההפעלה. הוא מאפשר את ההחלפה בין התהליכים וההליכים השונים, יצירת הליכים או תהליכים חדשים, וחיסול כאלו שכבר רצים. בנוסף, הוא מאפשר קבלת מידע על תהליכים והליכים שרצים.
* **PCI (pci.c) —** מודול זה אחראי על קבלת המידע אודות הרכיבים המחוברים למערכת מרכיב ה-PCI (Peripheral Component Interconnect).
* **USB (usb.c) —** מודול זה אחראי על האתחול, הניהול והתקשורת עם התקני ה-USB. הוא מסנן את המכשירים שתואמים את הסוג שבו הוא תומך (Mass Storage Bulk Only Transport), ומאתחל אותם. בנוסף, הוא מספק את השכבה הבסיסית היותר לתקשורת עם מכשירים אלו (מערכת העברת ה-packets).
* **BBB (Bulk-Only Trasport) —** מודול זה מספק את הממשק לביצוע פעולות על התקן מסוג BBB. הוא מספק פעולות לקריאת וכתיבת raw data מהתקן כזה, פעולה למציאת גודל שטח האכסון, ועוד.
* **SCSI (scsi.c) —** מודול זה מספק ממשק ליצירת SCSI Command Packets. במודול זה משתמש מודול ה-BBB על מנת ליצור תקשורת עם ה-SCSI Chipset הנמצא בהתקן.
* **EdenFS (edenfs.c) —** מודול זה אחראי על יצירת מערכת קבצים, וניהולה. הוא מספק פעולות לקריאת, כתיבת, יצירת ומחיקת קבצים, כמו כן יצירת ומחיקת תיקיות, "סיור" במערכת הקבצים. הוא מנהל את מערכת הקבצים ברמת ה-raw data.
* **Main Kernel (kernel\_c.c) —** מודול זה מכיר את הפונקציה הראשית של מערכת ההפעלה, שמורצת ישר לאחר ה-boot. מודול זה מאתחל את המערכת ואת כל רכיביה ע"י קריאה לפעולת האתחול של המודולים הנחוצים, והפעלת ממשק המשתמש.

## פירוט מודולי ה-Installer

|  |  |
| --- | --- |
| **מודול** | **תפקיד** |
| DeviceManager.py  *(צד לקוח)* | ניהול ההתקנים המחוברים למערכת והתקשורת איתם. |
| EdenFSManager.py  *(צד לקוח)* | ניהול מערכת הקבצים. |
| EdenOSClient.py  *(צד לקוח)* | ניהול התקשורת עם השרת (קבלת ה-Hash של מערכת ההפעלה או מערכת ההפעלה עצמה). |
| EdenOSManager.py  *(צד לקוח)* | המודול העיקרי שאותו מריצים בצד הלקוח. תפקידו הוא ניהול השימוש בכל המודולים ותקשורת עם המשתמש. |
| FSBrowser.py  *(צד לקוח)* | ניהול ממשק המשתמש בחלק של ממשק המשתמש לביצוע פעולות על מערכת הקבצים. הוא מספק את הממשק לקליטת וביצוע הפקודות. |
| UI.py  *(צד לקוח)* | סיפוק כלים ליצירת גרפיקה באופן טקסטואלי. ממשק זה מספק את ה-progress bar הטקסטואלי. |
| EdenOSServer.py  *(צד שרת)* | המודול העיקרי שאותו אנו מריצים בצד השרת. הוא מנהל את פעילות השרת: החיבורים ללקוחות ומסירת המידע ע"פ בקשות. |

# פירוט סוגי מבני הנתונים (structs)

|  |  |  |
| --- | --- | --- |
| **מודול** | **שם המבנה** | **תפקיד** |
| bbb.c | CBW (struct command\_block\_wrapper) | מעטפת ל-packet של פקודת SCSI |
| CSW (struct command\_status\_wrapper) | מעטפת ל-packet של סטטוס SCSI |
| cmd.c | struct command | מאחסן את המידע לגבי פקודה מסוימת |
| edenfs.c | DirEntry (struct dir\_entry) | Entry של אובייקט בטבלה של תיקיה |
| BootSect (struct boot\_sect) | מבנה ה-sector הראשון של הזיכרון במכשיר האחסון |
| LevelNode (struct level\_node) | תא ברשימה מקושרת המתארת את שכבות ה-bitmap של מערכת הקבצים |
| PathPart (struct path\_part) | חלק של כתובת במערכת הקבצים (חלק של רשימה מקושרת) |
| File (struct file) | מבנה המתאר קובץ פתוח |
| DirPart (struct dir\_part) | חלק של תיקיה, כפי שהוא מופיע בזיכרון |
| FilePart (struct file\_part) | חלק של קובץ, כפי שהוא מופיע בזיכרון |
| Idt32.c | struct idt\_entry | Entry של טבלת ה-IDT |
| struct idt\_ptr | מצביע לטבלת ה-IDT, כפי שהוא מאוכסן באוגר ה-IDTR |
| memory.c | struct bios\_mmap\_entry | Entry של טבלת מיפוי הזיכרון המתקבלת באמצעות ה-BIOS |
| paging.c | PTE (struct page\_table\_entry) | Entry של טבלת ה-Page Table |
| PDE (struct page\_dir\_entry) | Entry של טבלת ה-Page Directory |
| pci.c | ConfigAddr (struct cfg\_addr) | כתובת של רכיב במערכת, כפי שהיא מיוצגת באוגרים של רכיב ה-PCI |
| processing.c | Page (struct page\_node) | רכיב ברשימה מקושרת של דפים בזיכרון |
| ProcessNode (struct process\_node) | רכיב ברשימה מקושרת של תהליכים |
| ThreadNode (struct thread\_node) | רכיב ברשימה מקושרת של הליכים |
| PTE (struct page\_table\_entry) | Entry של טבלת ה-Page Table |
| PDE (struct page\_dir\_entry) | Entry של טבלת ה-Page Directory |
| scsi.c | Read12 (struct read12) | Read12 Packet |
| Write12 (struct write12) | Write12 Packet |
| TestUnitReady (struct test\_unit\_ready) | TestUnitReady Packet |
| Inquiry (struct inquiry\_packet) | Inquiry Packet |
| ReqSense (struct request\_sense) | RequestSense Packet |
| usb.c | UHCIDevice (struct uhci\_dev) | מבנה נתונים השומר מידע אודות התקן |
| FLP (struct frame\_list\_ptr) | מצביע ל-Frame List כפי שהוא נשמר באוגרים של התקן ה-UHCI |
| TD (struct transfer\_desc) | Transfer Descriptor כפי שמצוין ב-UHCI Specification |
| QH (struct queue\_head) | Queue Head כפי שמצוין ב-UHCI Specification |
| SetupReq (struct setup\_request) | Setup Request Packet |
| DevDesc (struct device\_desc) | Device Descriptor כפי שמתקבל מבקשת SETUP |
| ConfigDesc (struct configuration\_desc) | Configuration Descriptor כפי שמתקבל מבקשת SETUP |
| InterDesc (struct interface\_desc) | Interface Descriptor כפי שמתקבל מבקשת SETUP |
| EndpDesc (struct endpoint\_desc) | Endpoint Descriptor כפי שמתקבל מבקשת SETUP |

# פירוט מבני הנתונים

|  |  |  |  |
| --- | --- | --- | --- |
| **מודול** | **שם המבנה** | **סוג** | **תפקיד** |
| cmd.c | commands | struct command[] | לאחסן את המידע לגבי כל הפקודות |
| idt32.c | idt | struct idt\_entry[] | לשמש בתור טבלת ה-IDT |
| keyboard.c | scancode\_to\_  ascii | int[] | מערך המרה מ-scancode ל-ASCII |
| scancode\_to\_  ascii\_shift | int[] | מערך המרה מ-scancode ל-ASCII כאשר SHIFT לחוץ |
| scancode\_to\_  ascii\_caps | int[] | מערך המרה מ-scancode ל-ASCII כאשר CAPS מופעל |
| scancode\_to\_  ascii\_caps\_  shift | int[] | מערך המרה מ-scancode ל-ASCII כאשר CAPS מופעל ו-SHIFT לחוץ |
| memory.c | TYPES | const char \*[] | מערך הממיר מסוג (מספרי) לייצוג באמצעות מחרוזת. |
| processing.c | status\_ticks | unsigned int[] | מערך המכיל את מספר ה-ticks (בכל שנייה יש 1024 ticks) שלכל הליך יש, בהתאם למצבו |

# פירוט טבלאות לניהול המערכת

## Video Memory

בכתובת 0x000B8000 בזיכרון מתחיל ה-Video Memory, שבעצם מה שנמצא בו מוצג על המסך. המודול של ה-Console מנהל את אזור זה בזיכרון על מנת להציג על המסך את הטקסט המבוקש.

## BIOS Memory Map

טבלה זו המופקת ע"י ה-BIOS (בשלב ריצת ה-bootloader), ממפה את הזיכרון לזיכרון שבו ניתן להשתמש וזיכרון שבו לא ניתן להשתמש. מערכת ההפעלה עוברת על מאגר נתונים זה ומנהלת את הזיכרון בהתאם.

## מיפוי הדפים בזיכרון

מיפוי הדפים בזיכרון נעשה באמצעות bitmap שלה מוגדר כתובת קבועה בזיכרון (0x00100000). לכל דף בזיכרון מוקצה ביט אחד המציין אם הוא פנוי (0) או תפוס (1).

## טבלת ה-IDT (Interrupt Descriptor Table)

טבלת ה-IDT היא מאגר נתונים המכיל 256 entries. כל אחת מה-entries מייצגת את הכתובת אליה יש לעבור במקרה שפסיקה מסוימת מתקיימת. את ה-entry מוצא המעבד באמצעות מספר הפסיקה.

סרטוט המציג את המסלול שעוברת פסיקה מהרגע שהיא נקלטת עד שהפעולה המתאימה מבוצעת:

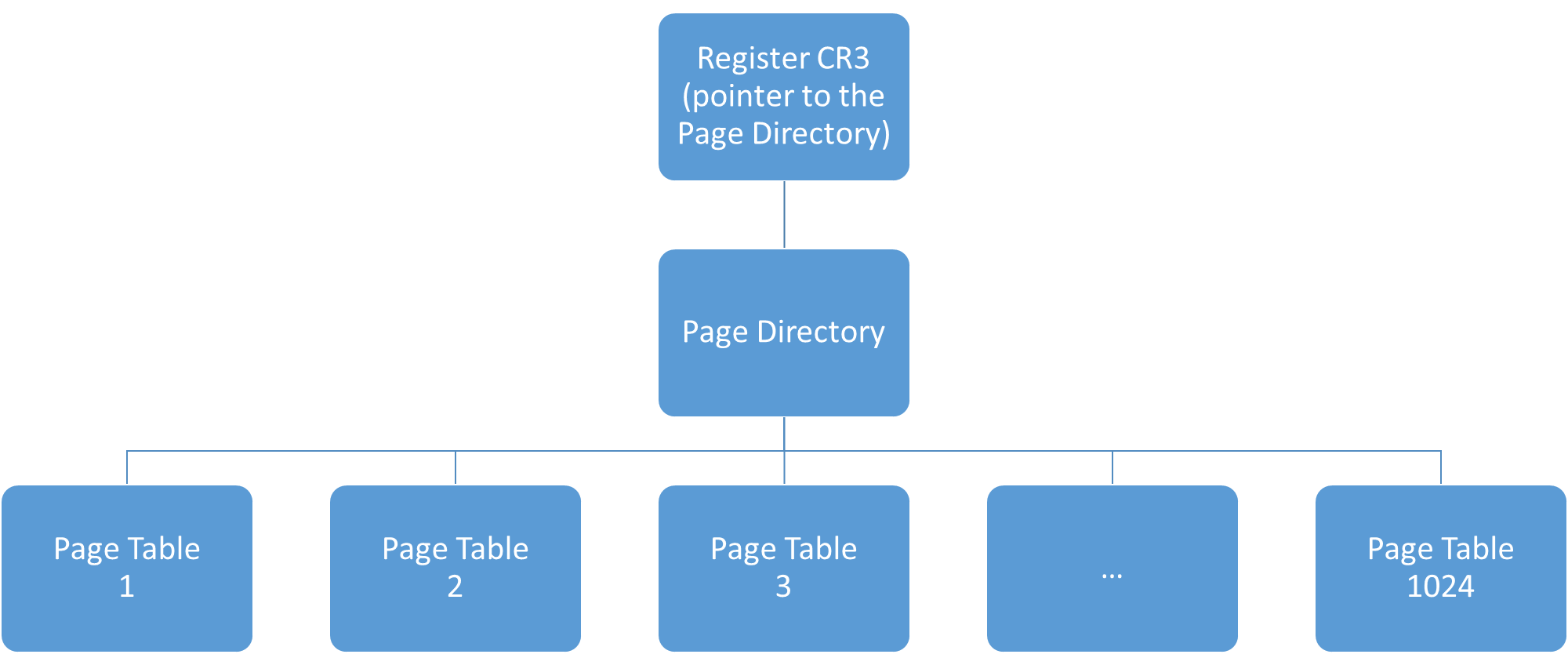
## טבלאות ה-Page Tables

טבלאות ה-Page Tables הן טבלאות המכילות (כל אחת) 1024 Page Entries. כל entry ממפה כתובת דף וירטואלית לפיזית, על פי מיקומו בטבלה, ומיקום הטבלה ב-Page Directory.

## טבלת ה-Page Directory

טבלת ה-Page Directory היא טבלה המכילת כתובות של 1024 Page Tables, ובעצם מהווה את ה-root של מערכת ה-Paging.

סרטוט המציג את מבנה טבלאות ה-Paging:



## PCI (Peripheral Component Interconnect)

רכיב זה של המערכת מכיל בעצם טבלה המכילה מידע לגבי כל אחד מהרכיבים המחוברים למערכת. מערכת ההפעלה שולחת שאילתות לממשק שמספק רכיב זה על מנת לקבל את המידע הנחוץ לה על כל רכיב.

## Frame List (UHCI)

ממשק ה-UHCI שבאמצעותו מתקשרים עם התקן ה-USB, מעביר את ה-packets שלו באמצעות מעין טבלה הנקראת Frame List. זהו בעצם מערך המכיל מספר Frames המצביעים ל-Transfer Descriptor או ל-Queue Head (כפי שמצוין ב-Specification של UHCI).

# השוואת העבודה עם פתרונות ויישומים קודמים

בעולמנו קיימות מערכות הפעלה רבות, כאשר חלק מהן מובילות את השוק בימינו. מערכת ההפעלה שאני בניתי שאבה השראה ממערכות הפעלה קיימות, אך גם יש בה הרבה דברים מקוריים.

נבחן מספר מערכות הפעלה קיימות ונציג את הדמיון והשוני ביניהן ובין EdenOS.

## MS-DOS

MS-DOS (Microsoft Disk Operating System) היא מערכת הפעלה המיועדת למעבדים מבוססי x86 אשר פותחה ע"י Microsoft. מערכת זו השתמשה ב-CLI (Command-Line Interface), שבה הזין המשתמש פקודה ומספר ארגומנטים, והמערכת הפעילה תהליך בהתאם, והעבירה אליו את הארגומנטים הנ"ל. מערכת זו תמכה במערכת הקבצים FAT.

במידה רבה ניתן לומר שנשאבה השראה מ-MS-DOS בעת כתיבת תכנון EdenOS. שתי מערכות ההפעלה מיועדות למעבדים מבוססי x86 ורצות ב-Protected Mode (32bit). בדומה ל-MS-DOS, גם EdenOS משתמש ב-CLI. גם כאן, מזין המשתמש פקודה ופרמטרים, ואלו מועברים לפעולה.

ההבדל אחד הוא שבעוד ש-MS-DOS משתמשת במערכת הקבצים FAT, EdenOS משתמשת ב-EdenFS. יש דמיון מסוים בין מערכות הקבצים, אך יש גם שוני רב.

## UNIX

UNIX היא מערכת הפעלה מרובת משימות ומרובת משתמשים, הנחשבת ליעילה ואמינה.

הדבר העיקרי הנלקח מ-UNIX היא ריבוי המשימות. טכניקת יצירת ה-Threads ב-EdenOS דומה מאוד לפעולת fork הממומשת ב-UNIX.

הבדל ניכר הוא ש-UNIX היא רבת משתמשים, בעוד שב-EdenOS אין שום מימוש לדבר.

## Windows (NT)

אחת ממערכות ההפעלה הנפוצות ביותר בימינו היא Windows של Microsoft. מערכת הפעלה זו היא גרפית. איתה הגיעה מערכת קבצים חדשה—NTFS, והיא ברירת המחדל בה (אף על פי שהיא תומכת גם ב-FAT).

שוני מובהק הוא שבעוד שממשק המשתמש ב-EdenOS הוא טקסטואלי, ב-Windows הוא גרפי. עם זאת, ניתן להשוות את EdenOS ל-Command Prompt של Windows, הפועל בצורה דומה ל-DOS.

אך גם ממערכת זו נשאבה השראה בתחום מסוים. במערכת זו קיים Scheduler מתוחכם המקצה לכל תהליך את זמן המעבד שמגיע לו, על פי הסטטוס שלו. שאר הזמן מועבר לתהליך Idle של המערכת. ב-EdenOS גם קיימים מספק מצבים, רק שכאן, במקום לכל תהליך, מוקצה זמן לכל הליך. בנוסף, לא קיים תהליך Idle של המערכת, והזמן שמקבל כל הליך תלוי רק במספר ההליכים הרצים במערכת, משום שה-Scheduler עובר ביניהם בלולאה אין-סופית.

## Linux

זוהי מערכת מבוססת Unix, אך גם היא, כמו Windows, בעלת ממשק משתמש גרפי.

הדבר ממנו נשאבה השראה ב-Linux, הוא שישנה אפשרות להתקרב לחומרה וקיימות פחות הגבלות על המשתמש מב-Windows.

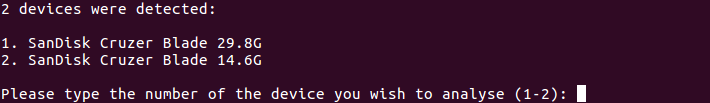
ב-EdenOS דבר זה נלקח צעד אחד קדימה: אין הפרדה בין ה-Kernel Mode ל-User Mode. אמנם המשתמש משתמש בכלים שמסופקים לו ע"י ה-Kernel, אך פעולותיו מבוצעות ברמת ה-Kernel—הרמה הנמוכה ביותר.

# מדריך התקנה למשתמש (Linux)

ההתקנה של מערכת ההפעלה מבוצעת באמצעות לקוח ב-Linux הצורב את מערכת ההפעלה על דיסק-און-קי (USB Flash Drive).

חבר את הדיסק-און-קי עליו אתה מעוניין לבצע את ההתקנה. הרץ את הקובץ EdenOSManager.py (ניתן לעשות זאת ע"י הפעלת ה-Terminal בתיקייה (מקש ימני 🡨 Open in Terminal) והזנת הפקודה *python EdenOSManager.py*).

כעת התוכנה תסרוק את המכשירים המחוברים למערכת ותציג את המכשירים עליהם ניתן להתקין את מערכת ההפעלה, באופן הבא:



הקש את המספר הסידורי של המכשיר המבוקש על מנת לבחור אותו.

כעת התוכנה תוודא באופן אוטומטי אם מותקנת מערכת ההפעלה העדכנית על המכשיר:



במידה ולא מותקנת מערכת ההפעלה העדכנית על המכשיר, התוכנה תתקין אותה באופן אוטומטי:



כעת תבדוק התוכנה אם המכשיר מפורמט למערכת ההפעלה EdenFS. אם הוא לא, תפרמט התוכנה את המכשיר למערכת הקבצים EdenFS. לפני ביצוע הפרמוט, תשאל התוכנה את המשתמש אם הוא מעוניין בבחירת תיקיית ROOT:



במידה ויבחר ב-'Y' ('כן') יוכל המשתמש להזין את הכתובת של התיקייה:



במידה ונבחרה תיקיית ROOT, הקבצים והתיקיות שבתיקייה זו יועתקו לתוך תיקיית ה-ROOT של מערכת הקבצים החדשה.

כעת תפרמט התוכנה את מערכת הקבצים במכשיר, ובמידת הצורך, תעתיק את הקבצים מהתיקייה שנבחרה אליה:

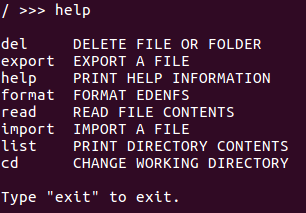


כעת יוצג בפני המשתמש שורת פקודה בה הוא יכול להזין פקודות על מנת לבצע פעולות על מערכת הקבצים שבמכשיר:



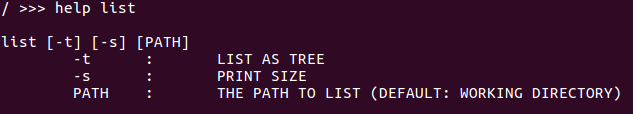
בתחילת השורה מופיע הכתובת של התיקייה בה כרגע מטפלת התוכנה, כאשר ‘/’ מסמל את תיקיית ה-ROOT. בין תיקיות ניתן לעבור באמצעות הפעולה ‘cd’ (Change Directory).

פעולות נוספות כוללות ייבוא וייצוא קבצים ממערכת הקבצים של המכשיר למערכת הקבצים המקומית של Linux, הצגת רשימה של תוכן תיקייה, ועוד. את מכלול הפקודות ומידע עליהן ניתן לקבל באצמעות פקודת ה-‘help’:



*\* זוהי דוגמה. בתוצר הסופי ייתכנו פקודות נוספות.*

על מנת לקבל מידע אודות פקודה ספציפית, ניתן להזין את הפקודה ‘help’ ואחריה את שם הפקודה המבוקשת. לדוגמה:

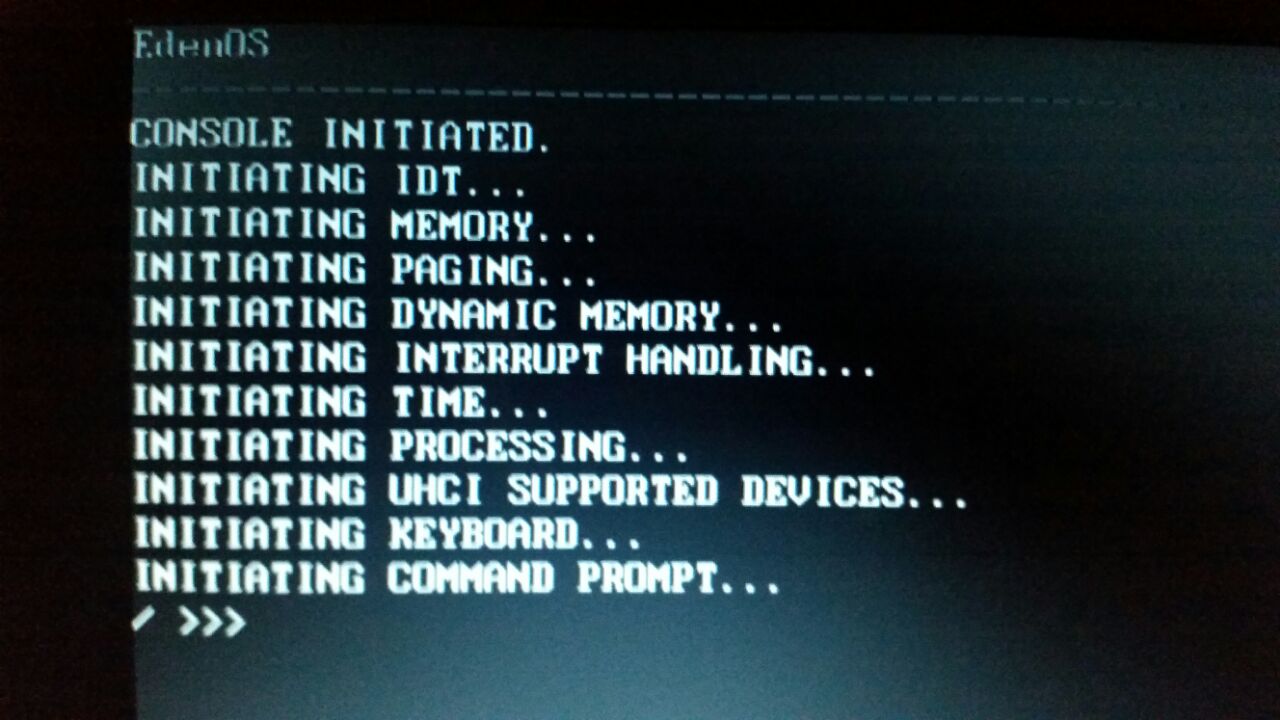


לאחר סיום ביצוע הפעולות הרצויות, יש להזין את הפקודה exit, לנתק את המכשיר, לחבר אותו למחשב אשר מריץ את המערכת ההפעלה, ולהפעילו כך שיריץ את מערכת ההפעלה הצרובה על המכשיר.

# תיאור הממשק למשתמש

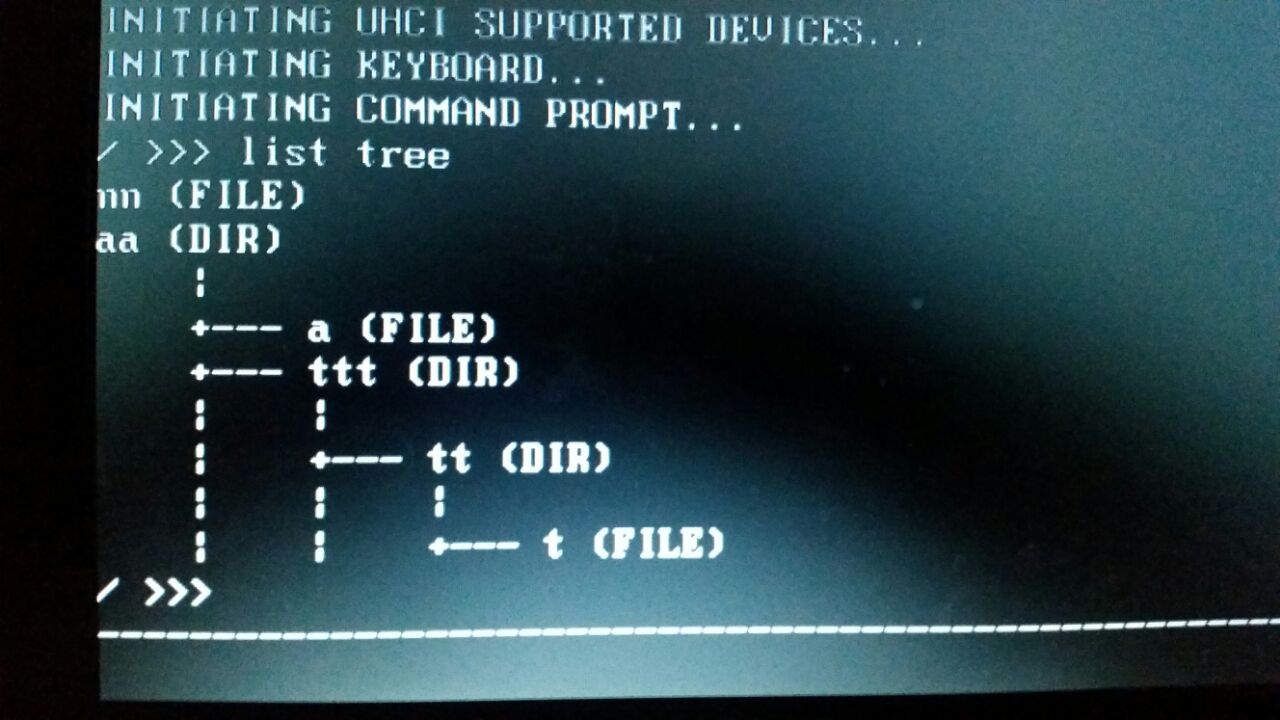
הממשק של המשתמש במערכת ההפעלה הוא CLI (Command Line Interface). המשתמש יכול להזין פקודות, להעביר אליהן פרמטרים, ולקבל פלט תואם. על מנת לקבל את רשימת הפקודות ותיאור של כל אחד, יש להזין את הפקודה ‘help’, ועל מנת לקבל מידע מורחב על כל פקודה יש להזין ‘help’ ובתור פרמטר את שם הפקודה.

לאחר ה-BOOT, המערכת תתחיל לפעול ותבצע פעולות ראשוניות של אתחול המערכת (בניית מבני נתונים וטבלאות, קונפיגורציה של התקנים ניידים ועוד), ולאחר הסיום, תופיע שורת הפקודה:



כעת יוכל המשתמש להתחיל להזין את הפקודות המבוקשות, ולתקשר עם המערכת.

פעולה לדוגמה היא ‘list’ הפולטת את תוכן התיקייה הנוכחית. במקרה זה הפקודה מבוצעת ב-ROOT ומועבר אליה הפרמטר ‘tree’, אשר גורם להצגת כל עץ הקבצים החל מתיקייה זו:



ניתן לראות שנפלט מבנה עץ הקבצים והתיקיות של מערכת הקבצים.

# מבט אישי על העבודה ותהליך הפיתוח

במהלך הכנת הפרויקט ביצעתי תהליך מעמיק של מחקר בתחומים שונים, החל מתחום החומרה ועד לתחום התוכנה. תחילה, הייתי צריך לחקור כיצד עובד תהליך ה-Boot במחשב על מנת שאוכל לכתוב קוד שיממש את התהליך הזה ויטען את מערכת ההפעלה שלי לזיכרון המחשב. בתהליך מחקר זה נעזרתי בעיקר באתר האינטרנט wiki.osdev.org המציע ערכים שונים לגבי תחומים מגוונים בפיתוח מערכת הפעלה, כולל ה-boot sequence. בנוסף, נעזרתי בפורומים שונים בהם דיווחו אנשים שונים על בעיות דומות שבהן אני נתקלתי בתהליך, ונעזרתי בתשובות שהם קיבלו מאנשים מנוסים יותר. את חלק זה של המערכת כתבתי ב-Assembly, שפה אשר למדתי בכיתה י', ולכן לא היה לי צורך לבצע מחקר מעמיק לגבי השימוש בשפה, אלא רק לגבי דברים ספציפיים שלא הכרתי.

לאחר מכן, הייתי צריך לפתח את מערכת ההפעלה עצמה בשפת C, שפה שמעולם לא התנסיתי. תחילה, ניגשתי לאתרים שונים המציעים ללמד את השפה. לאחר מכן, נעזרתי בספרים שונים על שפת C שנתקלתי בהם באינטרנט, כמו " The C Programming Language".

לפני שהתחלתי ממש לכתוב קוד, החלטתי לקרוא קצת לגבי המבנה של מערכות הפעלה, כדי שיהיה לי קצת רקע. מלבד אתרים שונים למתכנתי מערכות הפעלה שמציעים אלפי מאמרים לגבי תחומים שונים, נתקלתי גם בספרים שונים, לדוגמה, "Operating Systems Design and Implementation" שנכתב ע"י יוצר מערכת ההפעלה MINIX שעליה התבססה מערכת ההפעלה Linux. ספר זה נתן לי ידע ראשוני וידע בסיסי לגבי איך בנויה מערכת ההפעלה, וכיצד היא עובדת מאחורי הקלעים.

לאחר מכן, היה לי צורך ב-compiler שימיר את הקוד שלי לקוד בינארי שיוכל לרוץ על המעבד. חקרתי וקראתי כיצד "לבנות" compiler מותאם לצרכים שלי ולמעבד שעליו אני עובד.

מרבית המחקר שביצעתי לאחר מכן היה לגבי תחומים ספציפיים יותר במערכת ההפעלה, כמו ניהול זיכרון, טיפול בפסיקות ועוד. בשביל כל התחומים האלו נעזרתי בשני אתרים עיקריים המיועדים למתכנתי מערכות הפעלה (wiki.osdev.org ו-www.osdever.net) המכילים מאמרים רבים ואף מדריכים כיצד לממש תכונות שונות של מערכת ההפעלה, ופירוט על דרך הפעולה של מערכת ההפעלה וגישות שונות לגבי תכונות מסוימות שלה.

ראוי לציין שבמקרים מסוים פניתי (או הופניתי) ע"י מאמרים שונים ל-Manuals רשמיים של המעבד, שם קראתי בפירוט כיצד יש לנצל תכונות מסוימות שלו ולממש באמצעותו דברים מסוימים.

לאחר מכן החלטתי לממש מערכת קבצים משלי. זה היה תהליך מעניין במיוחד, משום שבעצם בניתי את מערכת הקבצים הזו מאפס, ותכננתי כל פרט בה. בנוסף, מימשתי אותה כך שתפעל באופן אופטימלי עם מערכת ההפעלה שלי. שאבתי השראה ממערכות הפעלה קיימות (בעיקר FAT), אך המערכת אינה דומה בשלמותה למערכות קיימות בהן נתקלתי, והיא פרי היצירה שלי.

אחד הדברים העיקריים שלמדתי מהפרויקט הזה, מעבר לכיצד לפתח מערכת הפעלה, הוא כיצד לבצע מחקר. התנסיתי בחיפוש אחר מקורות מידע בנושאים שהם לא כל כך נפוצים ויחסית מורכבים, ובהיעזרות אפילו ב-Manuals רשמיים במידת הצורך.

בנוסף, נתקלתי ב-community של מפתחי מערכות הפעלה חובבנים, שלא ידעתי שקיימת קודם. המידע העצום שהתפתח ב-community זו סייע לי מאוד ביצירת הפרויקט.

# ביבליוגרפיה

* OSDev—אתר wiki שנועד למפתחי מערכות הפעלה  
  <http://wiki.osdev.org/Main_Page>
* פורומים של OSDev  
  <http://forum.osdev.org/>
* אתר המכיל tutorial בתחום של פיתוח מערכות הפעלה  
  <http://www.osdever.net/>
* Universal Host Controller Interface (UHCI) Design Guide  
  <ftp://ftp.slackware.com/pub/netwinder/pub/misc/docs/29765002-usb-uhci%20design%20guide.pdf>
* Universal Serial Bus—Mass Storage Class—Specification Overview  
  <http://www.usb.org/developers/docs/devclass_docs/Mass_Storage_Specification_Overview_v1.4_2-19-2010.pdf>
* Universal Serial Bus—Mass Storage Class—Bulk-Only Transport  
  <http://www.usb.org/developers/docs/devclass_docs/usbmassbulk_10.pdf>
* SCSI Commands Reference Manual  
  <http://www.seagate.com/staticfiles/support/disc/manuals/Interface%20manuals/100293068c.pdf>
* Wikipedia  
  <https://www.wikipedia.org/>

# קוד הפרויקט

את קוד הפרויקט ניתן לראות בכתובת:

<https://github.com/eden060398/EdenOS>