**<Bug OS>**

מסמך אפיון

<0.0.1>

<1.11.2015>

אבידן בוריסוב

**הסטוריית גרסאות המסמך**

|  |  |  |  |
| --- | --- | --- | --- |
| **תאריך** | **עורך/ת** | **תיאור / תקציר השינויים** | **הערות נוספות** |
| 1.11.2015 | אבידן | גרסא ראשונה |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**1. הקדמה**

Bug היא מערכת הפעלה ב-32 bit למחשבים מבוססי ארכיטקטורת Intel x86.

1.1 הסבר כללי על המוצר

* *שם המוצר: Bug*
* *המוצר הינו מערכת הפעלה כוללת ומלאה למחשבים בעלי ארכיטקטורה תואמת ל-Intel i386*

*(ארכיטקטורות חדשות יותר בעלי תמיכה לאחור, כגון במעבדי אינטל מבוססי ,x86-64 נתמכות גם כן).*

* *המוצר אינו מבוסס על שום מערכת הפעלה מוכרת בשוק. כולו נכתב מאפס – מהרגע שהמחשב נדלק ועד שהוא נכבה, הקוד היחידי שירוץ יהיה קוד שנכתב עבור Bug. המוצר נכתב ב-x86 Assembly וב-C++, תוך דגש על עיצוב מודרני (בפרט נעשה שימוש ב-C++14), פשוט, קריא ותחזוקתי.*
* *מערכת ההפעלה ניתנת להפעלה מכל מחשב ביתי מצוי מתוך DOK בצורה חייה (Live-USB).*
* *מערכת ההפעלה לא משתמשת ב-bootloader חיצוני – Bug מכילה bootloader ייעודי שמצוי ב-boot sector של ה-USB שיודע למצוא ולהעלות את הליבה.*
* *מערכת ההפעלה יודעת להתממשק עם רכיבים חיצוניים:*
* *התקני אכסון*
* *שעון*
* *מסך*
* *מקלדת*
* *מערכת ההפעלה פועלת בצורה מאובטחת (היא רצה ב-Protected Mode עם Paging), לכל תהליך מובטחות הגנות זיכרון.*
* *שימוש במוצר:*
* *מערכת ההפעלה יודעת להריץ תהליכים אשר המשתמש יתקשר איתם.*
* *התהליכים רצים במקביל*
* *מערכת ההפעלה מספקת שירותים בסיסיים לתוכנות שרצות עליה (קריאות מערכת, ספריות).*
* *מערכת ההפעלה מספקת תוכנות עבור המשתמש.*
* *מערכת ההפעלה מכילה ממשק גראפי?*

1.2 הגדרות

IO = פעולות קריאה וכתיבה

סקטור = בלוק רציף של בתים שמהווה את היחידה הקטנה ביותר שניתן לקרוא/לכתוב מ/אל התקן חיצוני. לרוב מדובר ב-512 בתים.

דף = טווח זיכרון הקטן ביותר בגודל קבוע שניתן לעשות לו מיפוי מכתובת פיזית לכתובת וירטואלית. ב-x86 יש מספר אפשרויות לגודל דף, הגודל הנפוץ (שבו בין היתר משתמשת Bug) הוא 4096 בתים.

2. תיאור כללי

2.1 פונקציונליות

*מערכת ההפעלה אחראית לאתחל את כל מה שצריך לאתחל במחשב, ולאחר שסיימה לעשות זאת היא מפעילה את ה-Scheduler שמריץ תהליכים עבור המשתמש. בזמן ריצה מערכת ההפעלה תספק את השירותים אשר יאפשרו לתהליך הנוכחי לפעול.*

2.2 מאפייני משתמשים

**מערכת ההפעלה מיועדת לכולם.**

2.3 אילוצים עיקריים

*מערכת ההפעלה רצה רק על מחשבים מבוססי Intel i386 (או ארכיטקטורות חדשות יותר שתואמות ארכיטקטורה זו, כגון x86-64).*

*מערכת ההפעלה יכולה לרוץ מתוך כל התקן שמכיל לפחות MB1.44 שה-BIOS מאפשר לטעון מתוכו boot sector, למשל: DOK, CD, דיסקט. בפועל מערכת ההפעלה מיועדת לרוץ מתוך כל DOK סטנדרטי (תוך שימוש ב-Floppy emulation). סקריפט שמתקין את מערכת ההפעלה על DOK מסופקת כחלק מהפרויקט.*

*אין למערכת ההפעלה אילוצי זיכרון מיוחדים (לא ניתן למצוא מחשב אישי תואם למערכת ההפעלה בעל גודל זיכרון שלא יאפשר עבודה תקינה של המערכת).*

*קמפול מערכת ההפעלה יכול להיעשות רק בלינוקס. תהליך הבנייה דורש קרוס קומפיילר של GCC 5.1.0 ו-QMake (מערכת בנייה שמגיעה עם Qt Creator). ניתן להתקין את כלי הבנייה הנדרשים לבניית מערכת ההפעלה בהפצות לינוקס מבוססות Debian באמצעות סקריפט שמסופק כחלק מהפרויקט.*

2.4 הנחות ותלויות

*ההתקן עליו תותקן מערכת ההפעלה יכיל מערכת קבצים מסוג FAT12. ה-bootloader מצוי כולו בסקטור הראשון של ההתקן. הליבה והתוכנות שמופצות עם המערכת ימצאו על מערכת הקבצים.*

2.5 סיקור מצב השוק כיום (אופציונלי)

*יש מספר עצום של מערכות הפעלה למחשבים אישיים מבוססי אינטל. בין המוכרות: Windows, Linux, Mac OS. מערכות אלו מכילות מיליוני שורות קוד שנכתבו ע"י עשרות בני אדם במשך עשרות שנים.*

*Bug אינה מתחרה במערכות ההפעלה הקיימות בשוק מבחינת פונקציונליות. מטרתה לספק למשתמש את כל השירותים המצופים ממערכת הפעלה. כמו כן, ישנו דגש על הקריאות, הפשטות והעיצוב של מערכת ההפעלה.*

3. דרישות מפורטות

**3.1. דרישות פונקציונליות**

**Bootloader**

ה-bootloader חייב להיכנס כולו ל-sector אחד (512 בתים), אי לכך הוא חייב להיות מינימלי ככל הניתן. הוא צריך לעשות דבר אחד ולעשות אותו טוב: למצוא את הליבה ולטעון אותה לזיכרון.

1. ממשק IO בסיסי – נעשה בו שימוש רק במקרה של שגיאה. הוא מאפשר להציג טקסט על המסך, לחכות לקלט מהמשתמש ולעשות reboot.
2. ממשק דיסק – ממשק שמאפשר לקרוא סקטורים מההתקן ממנו עלה ה-bootloader אל הזיכרון.
3. ממשק מערכת הקבצים – ממשק שמאפשר למצוא קבצים בתוך מערכת הקבצים של ההתקן ולטעון אותם לזיכרון, תוך שימוש בממשק הדיסק.
4. הקוד העיקרי – מאתחל את המעבד למצב הרצוי וטוען את הליבה מההתקן אל הזיכרון תוך שימוש בממשק מערכת הקבצים. בסוף הטעינה תתבצע קפיצה לקוד הליבה שנטענה לזיכרון.

**Kernel (הליבה)**

זה החלק המרכזי והעיקרי במערכת ההפעלה. הליבה אחראית למעשה על כל מה שמתרחש במחשב.

הליבה אחראית לביצוע המשימות הבאות:

1. מעבר מהמצב בו היא נטענה מה-bootloader – כאשר המעבד קופץ לנקודת הכניסה בליבה, הוא עדיין נמצא במצב 16-bit Real Mode. המשימה הראשון של הליבה היא לאתחל את המחשב לפעול ב 32-bit Protected Mode. רק לאחר מכן מתבצעת הקריאה לשגרה הראשית של הליבה ומשם היא תמשיך בתהליך האתחול.
2. ניהול זיכרון – ניהול הזיכרון מורכב מהרבה חלקים שונים, והוא מתקשר כמעט לכל רכיב במערכת ההפעלה:
   1. זיהוי מפת הזיכרון – הליבה צריכה לזהות כמה זיכרון פנוי יש במחשב ואילו אזורים בזיכרון ניתנים/לא ניתנים לשימוש.
   2. מקצה זיכרון פיזי –ממשק להקצאה או שחרור של דפי זיכרון פיזי שייתחשב במפת הזיכרון הקיימת.
   3. Segmentation – ב-Protected Mode חייב להתבצע שימוש במנגנון שנועד להחליף את הסגמנטים של ה-Real Mode במנגון דינאמי שמאפשר הגנות זיכרון והרשאות שונות בין kernel mode ל-user mode (לא נעשה שימוש במנגנון זה על מנת לספק הגנות זיכרון, לטובת Paging).
   4. Paging – מנגון זה מאפשר את הקיום של זיכרון וירטואלי והגנות הזיכרון. מימוש של מנגון זה מספק לליבה את היכולת למפות כתובות וירטואליות לכתובות פיזיות בצורה דינמית.
   5. מקצה זיכרון וירטואלי – ממשק להקצאה או שחרור דפי זיכרון וירטואליים.
   6. Heap – ממשק שמאפשר למערכת ההפעלה להקצות או לשחרר בתים בודדים (בשונה מדפים שלמים).
3. ניהול פסיקות תוכנה ופסיקות מהתקנים חיצוניים –
   1. ממשק לניהול טבלת הפסיקות הגלובאלית (IDT)
   2. ניהול פסיקות תוכנה (ISR) – משמש לרישום שגרות לחריגות מעבד וקריאות מערכת.
   3. ניהול פסיקות חומרה (IRQ) – משתמש לרישום שגרות לסיגנלים מהתקנים חיצוניים, למשל המקלדת או ה-PIT (טיימר) שמשגר פסיקה כל פרק זמן קבוע ומאפשר למעשה את המימוש של ה-Scheduler.
4. תהליכים וה-Scheduler –
   1. מימוש של מבנה נתונים שמייצג תהליך:
      1. לכל תהליך מזהה ייחודי
      2. תהליכים פועלים ברמת הרשאה נמוכה מהליבה
      3. לכל תהליך מרחב זיכרון משלו, ולכל דף זיכרון ששייך לתהליך יש הרשאות בהתאם לסוג הסגמנט אותו הוא מייצג (קוד, מידע וכו').
   2. הScheduler יודע להריץ, להשהות, להמשיך ולהפסיק תהליכים בצורה הוגנת.
   3. תהליכים ניתנים לטעינה מתוך קבצי הרצה על התקן חיצוני.

מערכת ההפעלה מספקת לכל תהליך אוסף של קריאות מערכת שמשפיעות על פעילות ה-Scheduler:

1. לנהל את מצב התהליך הנוכחי: לשכפל את עצמו, להריץ תהליך חדש, לחכות פרק זמן, לסגור את התהליך.
2. פעולות IO – קריאה וכתיבה מקבצים, מהמסך, מהמקלדת.
3. ניהול מערכת הקבצים:
   1. ממשק "מערכת קבצים וירטואלי" –הקבצים אליהם ניתן לגשת יכולים להגיע ממקורות רבים, אך אין הבדל כלפי חוץ.
   2. מימוש דרייברים למערכות קבצים פופולאריות
   3. מימוש דרייברים לפעולות כתיבה וקריאה מההתקנים חיצוניים.
   4. קבצים מיוחדים – התוכן שלהם מיוצר בזמן ריצה ע"י הליבה.

**תוכנות (User space)**

1. Shell – איתו מתקשר המשתמש במצב טקסטואלי.
2. תוכנות מערכת בסיסיות: יצירת קבצים ותיקיות, עריכת קבצים, מידע על המערכת, כיבוי המחשב, וכו'

3.2 דרישות של ממשקים חיצוניים

מערכת ההפעלה תלויה רק בממשק חיצוני אחד, רק בעת תחילת עלייתה. ממשק זה הוא ה-BIOS, מערכת ההפעלה עושה שימוש בפסיקות מסוימות שה-BIOS אמור לספק, ביניהן:

1. פסיקות לקריאת מידע מהתקן חיצוני: קבלת מידע על ההתקן, קריאת סקטורים מההתקן.
2. פסיקות לקבלת מפת הזיכרון (גודל הזיכרון, אזורים שמורים, אזורים חופשיים).
3. כתיבה למסך (שה-bootloader עושה במידה והייתה שגיאה בתהליך טעינת ה-kernel).

3.3 דרישות לא פונקציונליות

1. מערכת ההפעלה תמיד תדאג להגיע לביצועים הכי גבוהים שניתן.
2. כל תהליך תחת מערכת ההפעלה מובטח לקבל:
   1. זמן עיבוד הוגן
   2. מרחב זיכרון וירטואלי: מבחינת התהליך, כל מרחב הזיכרון מעל סוף הליבה שייך לו. בפועל קיים מיפוי שונה לכל תהליך לכתובות פיזיות מאחורי הקלעים (Paging).
   3. אבטחה - מרחב זיכרון פרטי: כל תהליך יכול לגשת רק למרחב הזיכרון ששייך עבורו.
   4. שירותים – קריאות מערכת: מערכת ההפעלה תאפשר לתהליך לבצע קריאות מערכת עבור משימות שדורשות הרשאות גבוהות (למשל רוב סוגי ה-IO, למערכת הקבצים). באמצעות קריאות המערכת מתבצע המעבר בין קוד התהליך לקוד שרץ ב-kernel.

3.4 דרישות בסיס נתונים

נתונים ישמרו בזיכרון ובמערכת הקבצים.