עבודת סיום קורס מערכות הפעלה 2022

סימולציה של **Disk File System** - 'מערכת לניהול הדיסק'

תאריך הגשה 4.8.2022

תזכורת - עבודה עצמאית, מצאתי פתרון ברשת או במקום אחר או אצל סטודנט אחר או שנה אחרת – זה לא עבודה עצמאית, וגם לא משהו דומה...אנחנו גם נבדוק את זה. (**בבקשה, בבקשה, בבקשה!** זה רק בשבילכם, לא תעבדו עצמאית לא תלמדו...אין קיצורי דרך)

תיאור המערכת

סימולציה זוהי סביבת הדמיה בתוכנה לאירועים ופעולות הקורים במערכת אמיתית (חומרה או תוכנה).

מבוא ורקע :
מערכת ניהול הדיסק במערכת ההפעלה (Disk File System)- היא הדרך שבה שמות קבצים מיקומם ותוכנם מאורגנים על גבי הדיסק הקשיח. ללא מערכת הקבצים, המידע מאוחסן לא יהיה מאורגן לקבצים בודדים ויהיה בלתי אפשרי לאתר ולגשת לתוכנם.

המשתמש "הפשוט" אשר משתמש בתוכנת הוורד למשל, רואה לפניו קובץ וורד כאשר מיקום הקובץ הפיזי על-גבי הדיסק, אינו מעניינו של המשמש. וזו בדיוק תפקידה של 'מערכת לניהול הדיסק' במערכת ההפעלה, למפות את כל חלקי הקובץ אשר מאוחסנים על-גבי הדיסק. חלקי הקובץ שמורים ביחידות קטנות אשר נקראות 'בלוקים' ומאורגנים לכדי יחידה לוגית אחת. היא הקובץ. חלקי הקובץ אינם נמצאים באופן ישיר ורציף על הדיסק אלא מפוזרים על פני הדיסק. המיפוי של חלקי הבלוקים האלה לכדי קובץ שלם על ידי מערכת הקבצים בדומה לתרגיל מספר 6 אשר ביצעתם בסוף הסמסטר -- בתרגיל זה אנו נדרשים לסמלץ מערכת קבצים במערכת מחשב קטנה עם דיסק קטן ותיקייה בודדת ונממש את כל הפעולות אשר מערכת הפעלה עושה על הדיסק

נקודות מרכזיות בתרגיל:

- הדיסק שלנו יהיה למעשה קובץ! (בדומה למרחב השחלוף של הזיכרון בתרגיל 5). בגודל של 256 תווים בשם **"DISK_SIM_FILE.txt"**.
- מערכת קבצים זו תכיל רק תיקייה אחת וכל הקבצים יווצרו תחת תיקייה זו. (לא נממש תת תיקיות)
- מערכת ניהול הדיסק אותה נסמלץ היא **Indexed allocation**. בקצרה: לכול קובץ יהיה בלוק שמכיל מצעים לבלוקים של נתונם (ניתן דוגמא, וניתן לחוזר על החומר בשיעור)
- בתרגיל חובה לממש שלוש מחלקות/מבני נתונים **FsFile, FileDescriptor , fsDisk** כמו כן, נגדיר 8 פונקציות חובה למימוש במחלקה **fsDisk**.
  - את 8 הפעולות הבאות: FsFormat, CreateFile, OpenFile, CloseFile, DeleteFile, WriteToFile, ReadFromFile, listAll.
- ה-**main** של התרגיל נתון לכם וכן קוד ראשוני של המחלקות, וכן הפלט הנדרש..

בתרגיל זה אנו נממש יש לנהל שלושה מבני נתונים (מחלקות):

1. מידע על קובץ בודד ברמה "הנמוכה" ישמר במבנה הנתונים -- **FsFile**
   - תפקידו לשמור מידע בסיסי על הקובץ (גודל קובץ, באיזה בלוקים נישמר הקובץ.. ע"י מצביע ל indexBlock וכו')
2. קישור בין שם הקובץ לFsFile שלו, ישמר ב **FileDescriptor**
   - מבנה הנתונים הפשוט הנ"ל שומר את שם קובץ ומצביע ל-FsFile של הקובץ.
   (תיקייה במערכת היא למעשה מערך של FileDescriptor)
3. הדיסק עצמו ופעולות שניתן לעשות על הדיסק ישמור ב **fsDisk**
   - הדיסק עצמו, שומר את כל נתוני הדיסק.

<u>הרחבה על כל אחד ממבנה הנתונים/מחלקות</u>

כעת, נסביר בהרחבה על כל אחד ממבני הנתונים, אחר-כך נסביר על על אחת מ-8 הפעולות וכן נתאר את הממשק

1. **fsFile**

כאמור ה-fsFile הוא מבנה נתונים אשר שומר מידע בסיסי ברמה "הנמוכה" על הקובץ בשיטת index allocation

במחלקה fsFile אתם נדרשים להגדיר שלוש שדות חובה block_size, index_block, file_size, -, נסביר אותם:

- block_size– מספר התווים שאפשר לשמור בכל בלוק
- index_block – שדה זה שומר את מיקום ה index_block – כלומר את מספר הבלוק אשר מחזיק את המצביעים לבלוקים של נתוני הקובץ.
- file_size – גודל הקובץ (בתווים)

אופציונאלי - block_in_use - מספר הבלוקים בשימוש

```
class FsFile {
    int file_size;
    int block_in_use;
    int index_block;
    int block_size;


    public:
    FsFile(int _block_size) {
        file_size = 0;
        block_in_use = 0;
        block_size = _block_size;
        index_block = -1;
    }

    int getfile_size(){
        return file_size;
    }
```

ה-constructor מאתחל את גודל הקובץ (fileSize) ומספר הבלוקים לשימוש (block_in_used) לאפס. כמו כן מקצה ומאתחל מספר הindex_block ל 1-. (כלומר, טרם הוקצה).

מעבר לזה אתם רשאים/נדרשים להוסיף עוד פונקציות למחלקה זו.

מספר הבלוקים המקסימלי של קובץ במערכת שלנו יוכל להיות הם:

block_size

מעבר לזה - במערכת שלנו, לא נוכל לשמור יותר בלוקים עבור קובץ בודד!

ובהתאם, גודל קובץ (מספר התווים אשר ניתן לשמור בקובץ יהיה

block_size * block_size

2. **FileDescriptor**

המחלקה בפשטות שומרת מידע מצומד של שם קובץ ומצביע ל-FsFile של הקובץ. בנוסף, שומרת המחלקה משתנה בוליאני inUse שערכו שווה ל true כאשר פותחים את הקובץ ושווה ל-false כאשר סוגרים אותו. (שימו לב, סגירת קובץ זה לא מחיקת קובץ).

נשים לב שהתיקייה (MainDir) במערכת היא למעשה מערך של FileDescriptorים. במערכת שלנו תהיה תיקייה יחידה, [עוד בנושא זה - במבנה הנתונים הבא....]

<u>חתימה של תחילת המחלקה וה *constructor*:</u>

```
// ==============================================================
class FileDescriptor {
    string file_name;
    FsFile* fs_file;
    bool inUse;

    public:
    FileDescriptor(string FileName, FsFile* fsi) {
        file_name = FileName;
        fs_file = fsi;
        inUse = true;
    }

    string getFileName() {
        return file_name;
    }
```

אין לשנות מבנה נתונים זה.

3. fsDisk

```
class fsDisk {
    FILE *sim_disk_fd;
    bool is_formated;

    // BitVector - "bit" (int) vector, indicate which block in the disk is free
    //              or not.  (i.e. if BitVector[0] == 1 , means that the
    //             first block is occupied.
    int BitVectorSize;
    int *BitVector;

    // (5) MainDir --
    // Structure that links the file name to its FsFile

    // (6) OpenFileDescriptors --
    //  when you open a file,
    // the operating system creates an entry to represent that file
    // This entry number is the file descriptor.
```

זאת המנה העיקרית.... מבנה הנתונים האחרון שחובה להגדיר בתרגיל, הוא הדיסק עצמו. מחלקה בשם ה-fsDisk.

במחלקה זו, ישנם 6 שדות חובה הבאים: sim_disk_fd, is_formated, BitVectorSize, BitVector, MainDir, OpenFileDescriptors

כאשר הטיפוס 4 שדות נתון לכם ו 2 נוספים (MainDir, OpenFileDescriptors) לבחירתכם, בהתאם למימוש שתבחרו.

- המשתנה הראשון sim_disk_fd הוא מצביע על הקובץ מסוג של FILE* שאותו נפתח בעזרת פקודה fopen (ראו constructor) זה *ה*דיסק שישמש אותנו לסימולציה.
- המשתנה הבא, הוא is_formated - משתנה בוליאני, אשר מציין האם הדיסק הוא כבר עבור פורמט או לא (מדליקים אותו ל-true בסיום הפונקציה fsFormat (ראו בהמשך).
- משתנה נוסף הוא מערך בשם BitVector מסוג int, כל תא i במערך מציין האם הבלוק מספר i תפוס/בשימוש, כן או לא.
- משתנה נוסף הוא MainDir, זו "מערך" אשר מקשרת בין שם הקובץ ל-FsFile שלו. כאשר המימוש הספציפי נתון לשיקול דעתכם.
- כמו כן, יהיה לנו וקטור בשם OpenFileDescriptors, זה למעשה זה מערך של FileDescriptor שהם כל הקבצים הפתוחים, כל הקבצים שפתחנו בסימולציה. גם כאן לא הגדנו לכם באיזה מבנה נתונים להשתמש וזה לשיקול דעתכם.

שאלה: למה בכלל מעניין אותנו מבנה נתונים זה והאם אין כאן כפילות של MainDir ? תשובה: בעזרת מבנה זה נוכל לדעת איזה file descriptors פתוחים \ אילו סגורים ובמקרה של פתיחת קובץ חדש נוכל לדעת איזה מספר file descriptors חדש יש להקצות. כאשר MainDir מחזיק את כל הקבצים גם הסגורים...

```
// ------------------------------------------------------------------
fsDisk() {
    sim_disk_fd = fopen(DISK_SIM_FILE , "r+");
    assert(sim_disk_fd);

    for (int i=0; i < DISK_SIZE ; i++) {
        int ret_val = fseek ( sim_disk_fd , i , SEEK_SET );
        ret_val = fwrite( "\0" ,  1 , 1, sim_disk_fd);
        assert(ret_val == 1);
    }

    fflush(sim_disk_fd);
    is_formated = false;
}
```

כמו כן נתנו לכם בסיס לקונסטרקטור שבו יש להשתמש - כל תפקידו בשלב זה הוא לפתוח את קובץ הסימולציה של הדיסק ולאפס את תוכנו ואת שאר המשתנים..

אתם רשאים להוסיף עוד שדות ופונקציות לפי הבנתכם.

כעת, סיימנו להגדיר את מבני הנתונים העיקריים, נפנה להגדרת ה-main והפונקציות שחובה שיש לממש

```
int main() {
    int blockSize;
    string fileName;
    char str_to_write[DISK_SIZE];
    char str_to_read[DISK_SIZE];
    int size_to_read;
    int _fd;
    int writed;

    fsDisk *fs = new fsDisk();
    int cmd_;
    while(1) {
        cin >> cmd_;
        switch (cmd_)
        {
            case 0:   // exit
                delete fs;
                exit(0);
                break;

            case 1:  // list-file
                fs->listAll();
                break;
            case 2:    // format
                cin >> blockSize;
                fs->fsFormat(blockSize);
                break;

            case 3:    // creat-file
                cin >> fileName;
                _fd = fs->CreateFile(fileName);
                cout << "CreateFile: " << fileName << " with File Descriptor #: " << _fd << endl;
                break;

            case 4:  // open-file
                cin >> fileName;
                _fd = fs->OpenFile(fileName);
                cout << "OpenFile: " << fileName << " with File Descriptor #: " << _fd << endl;
                break;

            case 5:  // close-file
                cin >> _fd;
                fileName = fs->CloseFile(_fd);
                cout << "CloseFile: " << fileName << " with File Descriptor #: " << _fd << endl;
                break;

            case 6:   // write-file
                cin >> _fd;
                cin >> str_to_write;
                writed = fs->WriteToFile( _fd , str_to_write , strlen(str_to_write) );
                cout << "Writed: " << writed << " Char's into File Descriptor #: " << _fd << endl;
                break;

            case 7:    // read-file
                cin >> _fd;
                cin >> size_to_read ;
                fs->ReadFromFile( _fd , str_to_read , size_to_read );
                cout << "ReadFromFile: " << str_to_read << endl;
                break;

            case 8:   // delete file
                cin >> fileName;
                _fd = fs->DelFile(fileName);
                cout << "DeletedFile: " << fileName << " with File Descriptor #: " << _fd << endl;
                break;
            default:
                break;
        }
    }
```

)

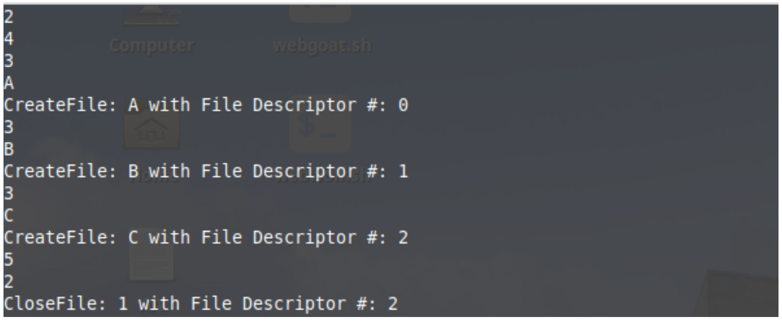
בתרגיל זה, ממשק ה-main נתון ובנוי על לולאה אשר כל פעם קולטת פקודה (מספר) מהמשתמש שהוא מספר בין אפס לשמונה.

- אופציה מספר אפס: מחיקת כל הדיסק ויציאה.
- אופציה מספר אחת: יש להדפיס את כל הקבצים שקיימים בדיסק void listAll(). הפונקציה הזאת נתונה לכם. הפונקציה מדפיסה את רשימת הקבצים שנוצרו בדיסק וכן את תוכן הדיסק[1] .
- אופציה מספר שתיים: פירמוט הדיסק - זימון הפונקציה fsFormat לפרמוט הדיסק. לצורך זה יש לקלוט מהמשתמש את גודל הבלוק בדיסק.
- אופציה מספר שלוש: יצירת קובץ ופתיחת קובץ - זימון הפונקציה .createFile, לצורך זה יש לקלוט מהמשתמש "שם קובץ", הפונקציה CreateFile מייצרת קובץ חדש במערכת. (רמז למימוש: פונקציה זו תהיה אחראית ליצירת fsFile וכן לעדכן את מבני הנתונים MainDir ו OpenFileDescriptors). הפונקציה תחזיר את file descriptor של הקובץ שנפתח (רמז: זה למעשה מיקומו בווקטור OpenFileDescriptors). הפונקציה גם תבדוק שהדיסק כבר אותחל ואם לא - תחזיר 1-. (מינוס 1)
- אופציות מספר ארבע וחמש הן פתיחה וסגירה של קובץ. אופציה מספר ארבע: פתיחת קובץ OpenFile מחזירה את ה file descriptor , יש לוודא שהקובץ קיים ולא פתוח כבר... (כאמור אופציה מספר <u>שלוש</u> יוצרת את הקובץ וגם פותחת אותו). אופציה מספר חמש נותנת לנו אפשרות לסגור את הקובץ CloseFile בהינתן file descriptor . כמובן שהפונקציה צריכה לוודא שיש קובץ כנ"ל ושהוא פתוח .בכל מקרה של שגיאה הפונקציה תחזיר 1- כ- string כאשר CloseFile או כ int מ OpenFile ....
- אופציות מספר שש ושבע הן, כתיבה וקריאה מקובץ: כאשר אנחנו רוצים לכתוב לקובץ ראשית לקלוט מהמשתמש מספר file descriptor של קובץ שאליו יש לכתוב, וסטרינג שאותו רוצים לרשום לתוך הקובץ. בהינתן שני אלו, מזמנים את הפונקציה WriteToFile שתכתוב נתונים לקובץ. <u>חלק</u> מהבדיקות שכמובן יש לוודא בפונקציה זו הם: (1) שיש מספיק מקום בדיסק, (2) בקובץ, (3) שהקובץ נפתח (4) ושהדיסק אותחל ואם לא תחזיר -1. (רמז למימוש WriteToFile: הפונקציה צריכה למצוא בלוקים פנויים בדיסק כדי לרשום לתוכם את הנתונים. אם כבר כתבו לקובץ זה בעבר, אולי נשאר מקום בבלוק האחרון שכבר הוקצה לקובץ זה ואז יש להשתמש בו..ובכל מקרה - בכל פעם יש להקצות מספר מינימלי של בלוקים הדרושים כדי לספק את הכתיבה הדרושה) אופציה מספר שבע לקריאה מקובץ ReadFromFile - ראשית שוב יש לקלוט מהמשתמש file descriptor של קובץ שממנו יש לקרוא וכן את מספר התווים שיש לקרוא. הפונקציה תיגש לקובץ המתאים, משם לבלוקים המתאימים (את הבלוקים נמצא דרך מבנה הנתונים FsFile של הקובץ) הפונקציה תחזיר את התווים שביקשנו ותדפיס אותם (כבר ממומש לכם ההדפסה :-) ...דברים נוספים שהפונקציה צריכה לבדוק? (תחשבו).
- אופציה מספר שמונה היא מחיקת קובץ. תקבל את שם הקובץ ותמחק את כל הנתונים שקשורים אליו. יש למחוק גם את ה-File שלו מתוך המאגר.

---

[1] תוכלו להעזר בפונקציה זו כדי ללמוד גם איך לגשת לדיסק לסרוק אותו ולקרוא ממנו נתונים,

דוגמת הרצה 1:



```
2
4
3
A
CreateFile: A with File Descriptor #: 0
3
B
CreateFile: B with File Descriptor #: 1
3
C
CreateFile: C with File Descriptor #: 2
5
2
CloseFile: 1 with File Descriptor #: 2
```

מה רואים בדוגמא ?

פרמוט דיסק עם בלוק בגודל 4, יצירת שלושה קבצים בשם A ו-B ו C. ומיד סגירת קובץ C.

דוגמת הרצה 2:

פרמוט דיסק עם בלוק בגודל 4,

הדפסת הדיסק. (אופציה מספר 1) כאשר – לצורך הנוחות הדפסנו כל תו מוקף בסוגריים. אנחנו רואים שבהתחלה הדיסק ריק.

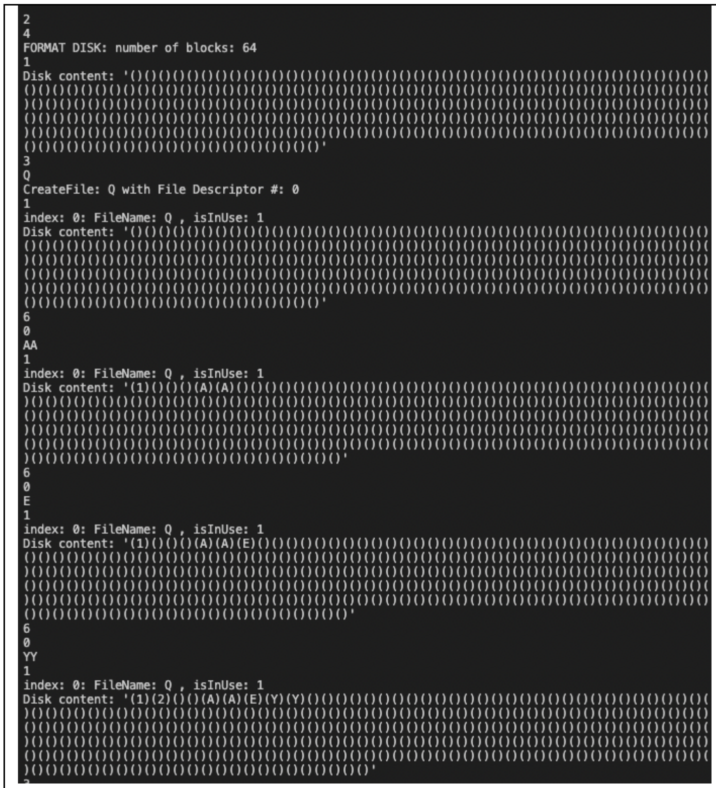
הדוגמא ממשיכה, ביצירת קובץ (אופציה מספר 3) בשם Q. הדפסת תוכן הדיסק (אופציה מספר 1).

הדוגמא ממשיכה, בכתבת שני תווים AA לקובץ Q. (כתיבה זה אופציה מספר 6) כתיבת לפייל דיסקריפטור מספר 0, ושוב הדפסת תוכן הדיסק. כאן, אנחנו כבר צוללים לתוך המימוש... ניתן לראות שנכתבו נתונים לשני הבלוקים הראשונים של הדיסק. בבלוק הראשון בדיסק הוא בלוק אינדקס נרשם ״1״ זה למעשה מצביע לבלוק הנתונים של הקובץ ובהתאם בבלוק אחד (השני בדיסק) הוא בלוק הנתונים של קובץ Q. – נרשמו התוויםAA .

הדוגמא ממשיכה, בכתבת תו E לקובץ Q.

הדוגמא ממשיכה, בכתבת שני תווים YY לקובץ Q.

אנחנו רואים מההדפסות שכאשר רושמים את E לקובץ עדיין יש מקום בבלוק הראשון של הקובץ. וכאשר רושמים את YY כבר יש צורך בבלוק נוסף.



```
2
4
FORMAT DISK: number of blocks: 64
1
Disk content: '()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()'
3
Q
CreateFile: Q with File Descriptor #: 0
1
index: 0: FileName: Q , isInUse: 1
Disk content: '()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()'
6
0
AA
1
index: 0: FileName: Q , isInUse: 1
Disk content: '(1)()()()(A)(A)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()'
6
0
E
1
index: 0: FileName: Q , isInUse: 1
Disk content: '(1)()()()(A)(A)(E)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()'
6
0
YY
1
index: 0: FileName: Q , isInUse: 1
Disk content: '(1)(2)()()(A)(A)(E)(Y)(Y)()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()'
```

```
3
B
CreateFile: B with File Descriptor #: 1
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
Disk content: '(1)(2)()()(A)(A)(E)(Y)(Y)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()(
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
6
1
DD
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
Disk content: '(1)(2)()()(A)(A)(E)(Y)(Y)()()()(4)()()()(D)(D)()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
6
0
KKKKK
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
Disk content: '(1)(2)(5)()(A)(A)(E)(Y)(Y)(K)(K)(K)(4)()()()(D)(D)()()(K)(K)()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
3
P
CreateFile: P with File Descriptor #: 2
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: P , isInUse: 1
Disk content: '(1)(2)(5)()(A)(A)(E)(Y)(Y)(K)(K)(K)(4)()()()(D)(D)()()(K)(K)()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
6
2
KLOPJSA
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: P , isInUse: 1
Disk content: '(1)(2)(5)()(A)(A)(E)(Y)(Y)(K)(K)(K)(4)()()()(D)(D)()()(K)(K)()()(7)(8)()()(K)(L)(O
)(P)(J)(S)(A)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
6
```

הדוגמא ממשיכה, ביצירת קובץ בשם B. הדפסת תוכן הדיסק.

הדוגמא ממשיכה, בכתבת שני תווים DD לקובץ B.

מתוך ההדפסה אנחנו רואים שהקובץ B משתמש בשני בלוקים בדיסק, הבלוק הרביעי והחמישי, בלוק ארבע, כבלוק אינקס, המצביע על הבלוק החמישי בקובץ (שימו לב, רשום שם 4, כי אנחנו סופרים את הבלוקים מאפס)

הדוגמא ממשיכה, בכתבת חמש תווים KKKKK לקובץ Q. לפי ההדפסה אנחנו רואים ששלושה תווים הלכו לבלוק קיים ושני תווים נוספים הלכו לבלוק חדש. אשר ממוקם בבלוק השישי בדיסק (כבר אחרי קובץ B), בהתאמה אנו רואים בהדפסה במקום השלישי בבלוק האינדקס של קובץ Q (בלוק אפס בדיסק) את המספר 5.

הדוגמא ממשיכה, בפתיחת קובץ נוסף בשם P וכיתבת מספר תווים לקובץ, אנחנו רואים שהקובץ החדש יש לו בלוק אינדס אחד ושני בלוקים של נתונים (7 ו 8 – כאשר סופרים מאפס)...

```
6
1
ASDAAA
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: P , isInUse: 1
Disk content: '(1)(2)(5)()(A)(A)(E)(Y)(Y)(K)(K)(K)(4)(9)()()(D)(D)(A)(S)(K)(K)()()(7)(8)()()(K)(L
)(O)(P)(J)(S)(A)()(D)(A)(A)(A)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
6
2
MMMMMM
1
index: 0: FileName: Q , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: P , isInUse: 1
Disk content: '(1)(2)(5)()(A)(A)(E)(Y)(Y)(K)(K)(K)(4)(9)()()(D)(D)(A)(S)(K)(K)()()(7)(8)(:)(;)(K)
(L)(O)(P)(J)(S)(A)(M)(D)(A)(A)(A)(M)(M)(M)(M)(M)()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
)()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()()
```

אני מניח שמכאן תצליחו לפרש את הדוגמא בעצמכם...

פונקציות עזר נוספות.

פונקציית ההדפסה:

שימו לב שפונקציית ההדפסה נתונה לכם (וזאת כדאי שההדפסה תצאה אחידה לכולם בבדיקה), אך יחד עם זאת יש כמה שורות קוד להשלים... וזאת בהתאם למימוש שלכם לשמירת ה file descripts ... הדפיסו את השם הקובץ והאם הוא בשימוש כן \ לא .

```
// ----------------------------------------------------------------------
void listAll() {
    int i = 0;

    for (.... ALL FILE DESCRIPTORS .... ) {
        cout << "index: " << i << ": FileName: " << .. FILE NAME ...  <<  " , isInUse: " << ... IS IN USED ... << endl;
        i++;
    }

    char bufy;
    cout << "Disk content: '";
    for (i = 0; i < DISK_SIZE; i++)
    {
        cout << "(";
        int ret_val = fseek(sim_disk_fd, i, SEEK_SET);
        ret_val = fread(&bufy, 1, 1, sim_disk_fd);
        cout << bufy;
        cout << ")";
    }
    cout << "'" << endl;
}
```

פונקציה <u>מאוד חשובה</u> decToBinary - מתי פונקציה זו שימושית? <u>כאשר רוצים לשמור את מספרי הבלוקים של ה- לדיסק</u>. יש להמיר את מספר הבלוק שמור במשתנה n לצורתו הבינארית כ-char שיישמר בתו c.

```
// ==========================================================================
void decToBinary(int n, char &c)
{
    // array to store binary number
    int binaryNum[8];

    // counter for binary array
    int i = 0;
    while (n > 0)
    {
        // storing remainder in binary array
        binaryNum[i] = n % 2;
        n = n / 2;
        i++;
    }

    // printing binary array in reverse order
    for (int j = i - 1; j >= 0; j--)
    {
        if (binaryNum[j] == 1)
            c = c | 1u << j;
    }
}
```

זהו ?

הגשה כרגיל… כולל README

בהצלחה!