עבודת סיום קורס מערכות הפעלה 2020

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

<u>תיאור המערכת</u>

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 לוג יושלם על ידי מערכת הקבצים. בנוסף 'מערכת לניהול הדיסק' היא זו שמנהלת את היררכיית התיקיות. בדומה לתרגיל מספר ארבע אשר ביצעתם במהלך 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 4 שהיה קובץ בדיסק).
- הדיסק יהיה בגודל 256 תווים ותו לא.:)
- מערכת קבצים זו תכיל רק תיקייה אחת וכל הקבצים יווצרו תחת תיקייה זו. (לא נממש ייצור של תת תיקיות)
- מערכת ניהול הדיסק אותה נסמלץ היא . **unix-fs**. עם שלושה **direct block** ו-**single in direct** בלוק אחד. כאשר ה-**direct block** נשמרים בתוך ה-**inode** ו-**single in direct** מצביע לבלוק בדיסק...
- בתרגיל חובה לממש שלוש מחלקות/מבני נתונים **fsInode, FileDescriptor , fsDisk** כמו כן, נגדיר 8 פונקציות חובה למימוש במחלקה **fsDisk**. אה.. למעשה 7, כי אחת אני נותן לכם :). גם ה-**constructor** הראשוני של כל מחלקה, יהיה נתון.
- ה-**main** של התרגיל גם הוא נתון לכם, וכן הפלט הנדרש.

אז מה נשאר? אז כאמור, בעבודה זו נממש סימולציה של גישות לדיסק. בתרגיל זה אנו נממש את 8 הפעולות הבאות:.FsFormat, CreateFile, OpenFile, CloseFile, DeleteFile, WriteToFile, ReadFromFile, listAll

יש לנהל שלושה מבני נתונים (מחלקות):

1. FsInode - תפקידו לשמור את מספרי הבלוקים בהם מאוחסן הקובץ.
2. FileDescriptor - שומר את שם קובץ ומצביע ל-inode של הקובץ.
   i. (הערה: תיקייה במערכת היא למעשה מערך של FileDescriptor)
3. fsDisk - הדיסק עצמו, שומר את כל נתוני הדיסק.

כעת, נסביר בהרחבה על כל אחד ממבני הנתונים, אחר-כך נסביר על על אחת מ-8 הפעולות וכן נתאר את הממשק (ה**main-**) אשר כאמור נתון.

אז נתחיל...

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

1. **fsInode**

ה-fsInode הוא מבנה נתונים אשר שומר את מספרי הבלוקים בהם מאוחסן הקובץ, כפי שהזכרנו בשיעור (ה-inode, - ראו הסבר בשיעור 11) במערכת ניהול הדיסק בשיטת inode - הבלוקים נשמרים במבנה נתונים דינאמי, כבר נסביר....

במחלקה זו, ישנם 4 שדות חובה: directBlock, singleInDirect, num_of_direct_blocks, block_size - נסביר אותם:

ה-num_of_direct_blocks בלוקים הראשונים בהם שוכן הקובץ נשמרים ישירות בfsInode, בתוך מערך בשם directBlock. גודלו של המערך יקבע להיות num_of_direct_blocks והוא יוקצה דינאמית ב-constructor של מחלקה זו (fsInode). במקרה ויהיה צורך ביותר מ-num_of_direct_blocks בלוקים, נשתמש ב-singleInDirect., .singleInDirect הוא משתנה מסוג int אשר שומר מספר של בלוק בדיסק, בלוק זה יוכל לשמור עד עוד block_size בלוקים בהם יישמר הקובץ.

כלומר, מספר הבלוקים המקסימלי של קובץ במערכת שלנו יוכל להיות הם:

num_of_direct_blocks + block_size

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

ובהתאם, גודל קובץ המקסימלי יהיה

(num_of_direct_blocks + block_size) * block_size

אתם רשאים להוסיף עוד שדות ופונקציות לפי הבנתכם. שדות נוספים שמומלץ להוסיף הם:
block_in_use - מספר המציין כמה בלוקים תכלס בשימוש עכשיו, fileSize - גודל הקובץ עד כה (מספר התווים שנרשמו לקובץ)

חתימה של תחילת המחלקה וה-*constructor*:.

ה-constructor מאתחל את גודל הקובץ (fileSize) ומספר הבלוקים לשימוש (block_in_used) לאפס.
כמו כן מקצה ומאתחל את מערך של ה-directBlocks וכן ה-.singleIndirect

```
class fsInode {
    int fileSize;
    int block_in_use;

    int *directBlocks;
    int singleInDirect;
    int num_of_direct_blocks;
    int block_size;


    public:
    fsInode(int _block_size, int _num_of_direct_blocks) {
        fileSize = 0;
        block_in_use = 0;
        block_size = _block_size;
        num_of_direct_blocks = _num_of_direct_blocks;
        directBlocks = new int[num_of_direct_blocks];
        assert(directBlocks)
        for (int i = 0 ; i < num_of_direct_blocks; i++) {
            directBlocks[i] = -1;
        }
        singleInDirect = -1;

    }
```

2. **FileDescriptor**

המחלקה אשר שומרת צמד (pair) של שם קובץ ומצביע ל-inode 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 שהתיקיה (MainDir) במערכת היא למעשה מערך של FileDescriptorים. במערכת שלנו תהיה תיקייה יחידה, [עוד בנושא זה - במבנה הנתונים הבא....]

חתימה של תחילת המחלקה וה *constructor*:

```
class FileDescriptor {
    pair<string, fsInode*> file;
    bool inUse;

    public:
    FileDescriptor(string FileName, fsInode* fsi) {
        file.first = FileName;
        file.second = fsi;
        inUse = true;

    }
```

3. fsDisk

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

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 טבלת ערבול (מערך אסוציאטיבי) אשר מקשרת בין שם הקובץ ל-inode שלו. כמו כן יהיה לנו וקטור בשם OpenFileDescriptors, למעשה זה מערך של FileDescriptor שהם כל הקבצים הפתוחים, כל הקבצים שפתחנו בסימולציה.

עד כאן השדות שהם חובה. שדות אופציונליים - לא חובה אך יכולים לעזור.. - הם direct_Enteris ו-direct_enteris. block_size מציין את מספר הבלוקים הראשונים בהם שוכן הקובץ ונשמרים ישירות ב-fsInode, ו-block_size הוא גודל הבלוק של הדיסק.

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

אתם רשאים להוסיף עוד שדות ופונקציות לפי הבנתכם, אך נוספים שמומלץ להוסיף הם:

```
#define DISK_SIM_FILE "DISK_SIM_FILE.txt"
// ==========================================================================
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;

    // Unix directories are lists of association structures,
    // each of which contains one filename and one inode number.
    map<string, fsInode*>  MainDir ;

    // OpenFileDescriptors --  when you open a file,
    // the operating system creates an entry to represent that file
    // This entry number is the file descriptor.
    vector< FileDescriptor > OpenFileDescriptors;

    int direct_enteris;
    int block_size;

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

    }

    private:
```

כעת, סיימנו להגדיר את מבני הנתונים העיקריים, נפנה להגדרת ה-main והפונקציות שחובה שיש לממש (נא להביא זכוכית מגדלת.... סתם - הקוד מצורף לכם)

```
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;
            cin >> direct_entries;
            fs->fsFormat(blockSize, direct_entries);
            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;
            fs->WriteToFile( _fd , str_to_write , strlen(str_to_write) );
            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 מאפיינים על הדיסק שהם: גודל הבלוק ומספר הdirect_entries שהיו בשימוש במבנה נתונים fsInode.
- אופציה מספר שלוש: יצירת קובץ -- זימון הפונקציה .reateFile, לצורך זה יש לקלוט מהמשתמש "שם קובץ", הפונקציה CreateFile מייצרת קובץ חדש במערכת. (רמז למימוש: פונקציה זו תהיה אחראית ליצירת fsInode וכן לעדכן את מבני הנתונים MainDirו OpenFileDescriptors). הפונקציה תחזיר את פייל-דיסקריפטור 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 הפייל-דיסקריפטור, 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 פייל-דיסקריפטור. כמובן שהפונקציה צריכה לוודא שיש קובץ כנ"ל ושהוא פתוח .בכל מקרה של שגיאה הפונקציה תחזיר 1- כ- string כאשר CloseFile או כ- int מOpenFile ....
- אופציות מספר שש ושבע הן, כתיבה וקריאה מקובץ: כאשר אנחנו רוצים לכתוב לקובץ ראשית לקלוט מהמשתמש פייל-דיסקריפטור של קובץ שאליו יש לכתוב, וסטרינג שאותו רוצים לרשום לתוך הקובץ. בהינתן שני אלו, מזמנים את הפונקציה WriteToFile. חלק מהבדיקות שכמובן יש לוודא בפונקציה זו הם: שיש מספיק מקום בדיסק, בקובץ, שהקובץ נפתח ושהדיסק אותחל ואם לא תחזיר -1. (רמז למימוש: הפונקציה צריכה למצוא בלוקים פנויים בדיסק כדי לרשום לתוכם את הנתונים. אם כבר כתבו לקובץ זה בעבר, אולי נשאר מקום בבלוקים שכבר הוקצו לקובץ זה ובכל מקרה - בכל פעם יש להקצות מספר מינימלי של בלוקים הדרושים כדי לספק את הכתיבה הדרושה) אופציה מספר שבע לקריאה מקובץ ReadFromFile - ראשית לקלוט מהמשתמש פייל-דיסקריפטור של קובץ שממנו יש לקרוא וכמות תווים שיש לקרוא. הפונקציה תיגש לקובץ המתאים, משם לבלוקים המתאימים ותחזיר לנו את כמות הנתונים שביקשנו...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-inode שלו מתוך המאגר.

---

[1] תוכלו להhעזר בפונקציה זו כדי ללמוד גם איך לגשת לדיסק לסרוק אותו ולקרוא ממנו נתונים, -- ולשמור ? זה מאוד דומא רק עם fwrite.

דוגמאות הרצה:

<u>דוגמא 1:</u>

יצירת דיסק עם בלוק בגודל 4 ושלושה direct-entries , יצירת שני קבצים בשם A ו-B. לתוך קובץ A רשמנו ABCD ולתוך קובץ B רשמנו ABCDEFGH והדפסנו את תוכנם.

```
2
4
3
FORMAT DISK: number of blocks: 64
3
A
CreateFile: A with File Descriptor #: 0
3
B
CreateFile: B with File Descriptor #: 1
6
0
ABCD
7
0
4
ReadFromFile: ABCD
6
1
ABCDEFGH
7
1
8
ReadFromFile: ABCDEFGH
```

דוגמא 2:

יצירת דיסק עם בלוק בגודל 4 ושלושה direct-entries, יצירת שלושה קבצים בשם A , B ו-C. וסגירת קובץ C. לתוך קובץ B רשמנו ABCDEFG ולתוך קובץ A רשמנו QWERQWER. בין לבין זימנו את הפונקציה listall עם אופציה מספר 1.

```
2
4
3
FORMAT DISK: number of blocks: 64
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: C with File Descriptor #: 2
6
1
ABCDEFG
1
index: 0: FileName: A , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: C , isInUse: 0
Disk content: 'ABCDEFG'
6
0
QWERQWER
1
index: 0: FileName: A , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: C , isInUse: 0
Disk content: 'ABCDEFGQWERQWER'
```

המשך דוגמא 2:

פתיחה מחדש של קובץ C והדפסה של מספר תווים רב יחסית לתוך הקובץ. הפעם מספר התווים הוא רב, ולכן יש גם צורך בשימוש של singleInDirect. איך אנו רואים זאת? רואים שיש בלוק "רק" בהדפסת הדיסק, שם שומרים את מספרי הבלוקים של קובץ C שהם מעבר לשלושה של ה-direct.

```
4
C
OpenFile: C with File Descriptor #: 2
6
2
AZXCDFVBGHNMJK<IUYWEW
1
index: 0: FileName: A , isInUse: 1
index: 1: FileName: B , isInUse: 1
index: 2: FileName: C , isInUse: 1
Disk content: 'ABCDEFGQWERQWERAZXCDFVBGHNMJK<IUYWEW     '
```

פונקציות עזר נוספות.

פונקציית ההדפסה:

```
// ----------------------------------------------------------------------
void listAll() {
    int i = 0;
    for ( auto it = begin (OpenFileDescriptors); it != end (OpenFileDescriptors); ++it) {
        cout << "index: " << i << ": FileName: " << it->getFileName() <<  " , isInUse: " << it->isInUse() << endl;
        i++;
    }
    char bufy;
    cout << "Disk content: '" ;
    for (i=0; i < DISK_SIZE ; i++) {
        int ret_val = fseek ( sim_disk_fd , i , SEEK_SET );
        ret_val = fread(  &bufy , 1 , 1, sim_disk_fd );
         cout << bufy;
    }
    cout << "'" << endl;


}
```

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

```
char  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... קובץ אחד, לכווץ את תיקיית ה visual-studio-code, כולל README

בהצלחה!