מכללה אקדמית הדסה
החוג למדעי המחשב
תרגיל 9 בקורס מ.ה. ב'
הגשה: 10.06.2011
בנושא: סימולציה של מערכות קבצים

בתרגיל זה נבנה מערכת קבצים בעלת מבנה הדומה לזו של System V (s5fs). מערכת הקבצים שנבנה תשתמש בקובץ רגיל כדיסק לוגי. את הקובץ ניתן יהיה לפרמט למערכת ההפעלה שלנו (פעולה שתאתחל את מבני הנתונים הדרושים בקובץ/זיכרון ובכך גם תרוקן את מ.ק. במידה והיו בה כבר קבצים) ואז ליצור קבצים שונים, לכתוב מידע אליהם, לקרוא אותם ולסגור אותם.

- הדיסק שלנו יהיה למעשה קובץ! (כמו מרחב השחלוף של הזיכרון בתרגיל 7 שהיה קובץ בדיסק).
- במערכת קבצים זו תהיה רק ספריה אחת (root) וכל הקבצים יווצרו תחת ספריה זו. כלומר לא נאפשר לייצר תת מדריכים של root.

Data structures

- הstruct fs מכיל את כל מבני הנתונים הדרושים לניהול מערכת הקבצים. כלומר הוא השומר את המידע הגלובלי אודות מערכת הקבצים :

```
struct fs{
    // tell us which blocks are occupied
    int Bitmap[NR_BLOCKS];

    // "map" the inodes on the disk
    struct FSInode inodeList[NR_INDOES];

    // pointer to the first Inode (the first directory)
    struct FSInode* pRootInode;

    // holds the open file desvriptors.
    struct FileDescriptor fileTable[NR_INDOES] ;

    //=0 if initializes, 1 otherwise
    int fsInitialized ;

    // fd of our virtual HDD
    FILE* fd  ;
};
typedef struct fs fs_t;
```

מספר דגשים:

- למה משמש Int fsInitialized ?
  - כאשר עושים mount יש לאתחל ערך זה לאפס. ורק אחרי הformat הראשון יש לאתחל אותו לאחד.
- מה השימוש של הflag הזה ?
  - כדי שנדע שהמערכת עדיין לא אותחלה ולא נאפשר לבצע פקודות כמו read/write לפני שבוצע format.
- מה השימוש של file-descriptor[0] ?
  - בעזרתו וע"י שימוש בfsOpenFile ו fsReadFile תוכלו לקרוא בעזרתו את רשימת הקבצים במערכת.

מבנה נתונים (הוסברו בתרגול):

```
struct FSInode {
    int inUse;
    int fileSize;
    int directBlocks[DIRECT_ENTRIES];
    int singleIndirectBlocks[SINGLE_INDIRECT_ENTRIES];
    int doubleIndirectBlocks[DOUBLE_INDIRECT_ENTRIES];
};

struct FileDescriptor {
    int inUse;
    int fd;
    char filename[MAX_FILENAME];
    int inode;
    int fileOffset;
};

struct DirEntry {
    char filename[MAX_FILENAME];
    int inode;
};
```

<u>דגשים על מבנה הנתונים:</u>

:

א. הקובץ שיסמלץ את מ.ק. יקרא testFS.dat ויהיה בגודל 63456 בתים[1]. (על כן עליכם לצור במדריך שלכם קובץ כנ"ל. עשו זאת למשל בעזרת תכנית קטנה שתייצר את הקובץ, תיפנה בו להסטה הדרושה, ותכתוב עליה תו יחיד, וכך יצרתם קובץ עם 'חור עצום' שהינו בגודל המתאים.)

ב. בתרגיל הנוכחי גוש במערכת הקבצים יהיה בן 16 בתים (כמוגדר ע"י הקבוע BLOCK_SIZE), ועל כן כל פעולת קלט\פלט תקרא כמות כזאת של נתונים.

ג. מ.ק. שלנו תכיל את המרכיבים הבאים:

1. מפת סיביות המתארת אילו גושים במ.ק. פנויים\תפוסים.
2. מערך של inode, כל inode מכיל מידע אודות קובץ בודד במ.ק..
3. גושי נתונים בהם נשמרים נתונים הקבצים השמורים במ.ק. שלנו.

ד. הגושים המרכיבים כל קובץ וקובץ ישמרו במ.ק. (כלומר בקובץ שמסמלץ את מ.ק. אצלנו) בגושים אינם בהכרח רציפים, ונשתמש באינדקס, בדומה למה שתואר בכיתה, ועל-פי ההנחיות שמוצגות בהמשך, כדי לאתר

---

[1] חשבון מפורט של גודל הקובץ:
blockBitmaps - 3712 = 3712 bytes
listInodes - 128*28 = 3584 bytes
blocks - 16*3712 = 59392 bytes

את גושי הקובץ. האינדקס, אצלנו, כמו בעולם האמיתי יהיה חלק מה- inode שהינו מבנה הנתונים המרכזי המכיל את המידע אודות כל קובץ וקובץ.

ה. כאמור, במ.ק. שלנו נאפשר יצירה רק של מדריך שורש יחיד (ללא תת-מדריכים). על כן יהיה לנו גם מערך יחיד שיכיל את נתונים הקבצים הנשמרים במדריך היחיד. כזכור מהשיעור, כל כניסה במערך מכילה את שם הקובץ, ואת מספר ה- inode שלו במ.ק.

ו. קובץ הסימולציה ייפתח באמצעות FILE *fd כלומר באמצעות כלים של שפת סי, כלומר כל הקלט והפלט בתכנית, יבוצע באמצעות read()/write()

ז. מבנה הנתונים struct FSInode מכיל מידע אודות כל קובץ וקובץ במ.ק.. על כן נזדקק למערך של מבנים כנ"ל. אודות כל קובץ, בתרגיל שלנו, נחזיק את הנתונים: גודל הקובץ בבתים, כתובות של DIREST_ENTRIES גושים של הקובץ אליהם יש הפניה ישירות מה- inode, כתובות של SINGLE_INDIRECT_ENTRIES גושים, המכילים כ"א כתובות של גושים הנכללים בקובץ, ואליהם אנו פונים בגישה עקיפה מרמה אחת. ובדיוק באותו אופן גם לגבי DOUBLE_INDIRECT_ENTRIS

ח. מערך הbitmap - מערך בגודל **3712** כאשר כל תא בו מייצג בלוק. לדוגמא: כאשר int מספר 78 דלוקה אזי הblock 78 תפוס וכאשר הוא אפס אזי הוא פנוי. מערך זה ישמש אותנו כאשר נרצה לכתוב נתונים במערכת הקבצים לשלנו וכאשר נרצה לכתוב נתונים ונצטרך עוד בלוקים לכתיבה נעזר במערך זה ע"מ למצוא בלוקים פינוים אלו.

1. <u>שני מבני נתונים</u> הנ"ל נשמרים באופן בסיסי בדיסק. עת טוענים את מ.ק עותק שלהם נטען לזיכרון. נדבר על כך גם בהמשך אך אזכיר זאת גם כאן: הBitmap וכן מערך של הinodes הינם נתונים אשר: יכתבו בעזרת write() ובהתאמה יקראו בעזרת read() לתוך מצביע ל- struct. כאשר ראשית נקרא את הSuper Block ובעזרתו נוכל לחלץ את המיקומים של שאר הנתונים בקובץ

ט. DirEntry הינו מבנה נתונים אשר נשמר עבור כול קובץ הקיים במערכת. אנו פשוט רושמים אותו לתוך inode מספר 0. כלומר אנו משתמשים בקובץ בתוך המערכת שלנו כדי לשמור את רשימת הקבצים הקיימים.

י. מערך ה FileDescriptor <u>אינו נישמר בשום מקום</u> אלא נבנה כל פעם מחדש כאשר אנו מבצעים mount ונמחק כאשר אנו עושים unmount. מערך זה ניבנה מיתוך רשימת הinodes הכללית ורשימת הקבצים במערכת: שרשומים כאמור בinode מספר 0.

קבועים

(הוסברו בתרגול):

```
#define BLOCK_SIZE 16
#define BLOCK_ADDRESS_SIZE  4
#define DIRECT_ENTRIES 3
#define SINGLE_INDIRECT_ENTRIES 1
#define DOUBLE_INDIRECT_ENTRIES 1
#define ENTRIES_PER_BLOCK (BLOCK_SIZE/BLOCK_ADDRESS_SIZE)

#define NR_INDOES  128

#define NR_BLOCKS  3712
#define ROOT_DIRECTORY_HANDLE 0  // the root ('/') dir handle

// maximum blocks held by the single/double indirect entries
#define SINGLE_INDIRECT_BLOCKS ENTRIES_PER_BLOCK
#define DOUBLE_INDIRECT_BLOCKS (ENTRIES_PER_BLOCK*ENTRIES_PER_BLOCK)

// maximum blocks an inode can hold
#define BLOCKS_PER_INODE (DIRECT_ENTRIES \
                        + SINGLE_INDIRECT_ENTRIES*SINGLE_INDIRECT_BLOCKS \
                        + DOUBLE_INDIRECT_ENTRIES*DOUBLE_INDIRECT_BLOCKS)

// maximum file size in bytes
#define MAX_FILE_SIZE (BLOCKS_PER_INODE*BLOCK_SIZE)

// maximum file name
#define MAX_FILENAME 12

#define BLOCK_BITS (BLOCK_SIZE*sizeof(char))  //num of bits per block
```

פונקציות שיש לממש:

שם הפונקציה	פעולה
fsFormat(fs_t* fs, char* filename);	הפונקציה מאתחלת את מבנה הנתונים עבור מערכת ההפעלה בזיכרון ו בדיסק הנדרשים למימוש המערכת.
מאתחלת את מבנה הנתונים של המערכת. • filename יהיה משהוא כמו "c:\" והוא יהיה השם של file-descriptor[0].	
fs_t* fs fsMount(char* filename);	הפונקציה מאתחלת את מבני הנתונים של מע' הקבצים מהמידע השמור על הדיסק
fsUnMount(fs_t* fs);	הפונקציה שומרת שינויים/עדכונים שנעשו בזיכרון אל הדיסק (קובץ) ומנקה את מבני הנתונים בזיכרון. הכוונה משחררת.
• למערכת קבצים זו נוכל להתחבר ע"י Mount ולהתנתק ע"י unmount. o חיבור למערכת הקבצים כולל קריאת כל הנתונים על-מנת לאפשר אתחול של ה struct fs. ▪ כאשר ה struct fs כולל את: superBlock, blocksBitmap וה inodeList והם נקראים ישירות As Is מהקובץ. כלומר הנתונים נקראים מהקובץ שמדמה את הדיסק כגוש באמצעות פעולת read() לתוך מצביע ל- struct. ▪ שם לב: אין צורך לאתחל כאן את הfile-table o umount: שמירת הstruct fs לתוך הקובץ.	
fsPrintRootDir(fs_t* fs);	הפונקציה מדפיסה את תוכן הספרייה הראשית (והיחידה הקיימת בתרגיל שלנו). יש להדפיס את שמות הקבצים הנכללים בספרייה.
• ניגשת לקובץ הראשון בדיסק (**שימו לב: שמדובר על קובץ וירטואלי בדיסק הסימולציה**). קובץ זה, <u>ביתר דיוק ספריה</u> זו מכילה: Structים של DirEntry (DirEntry מכיל: שם קובץ, inode שלו) אחד אחרי השני! כך תוכל המערכת למפות בין שם הקובץ לinode שלו. • פונקציה זו עוברת אחד אחד על structים אלו ומדפיסה את שמות הקבצים.	
	פונקציות הפועלות על קובץ
int fsCreateFile(fs_t* fs, char* fileName);	הפונקציה יוצרת קובץ חדש בשם נתון (מחזירה int שהוא הfd)
o אתחול מבנה הנתונים עבור קובץ חדש במערך הinode ובמערך הfile-table. o רישום שם הקובץ החדש ב rootDir, כלומר שומר עבורו DirEntry חדש בinode[0]. (שימו לב שאתם מבינים את הדרישה האחרונה: המשפט יכול קצת לבלבל.) o מאתחלת תא חדש במערך ה fileTable ומחזירה file-descriptor ▪ ראה הסבר מפורט על עבודה עם FileTable בסעיף (fsPrintRootDir).	
int fsOpenFile(fs_t* fs, char* fileName);	הפונקציה פותחת קובץ נתון לקריאה וכתיבה (מחזירה int שהוא

	הfd).
o דומה ל Create אך אינו יוצר מקום חדש במבנה הנתונים אלא מחפש האם הקובץ קיים ב[0]inode ומקשר אותו למקום הפנוי הבא ברשימת הinodes o מאתחלת תא חדש במערך ה fileTable ומחזירה file-descriptor • **מה** ההבדל בין open ו create ? o : בגדול, בcreate מאתחלים inode ובopen מfileDiscriptor.	
fsCloseFile(fs_t* fs, int fileHandle);	הפונקציה סוגרת קובץ. שומרת עדכונים שנעשו למבני הנתונים הרלוונטיים לדיסק.
fsWriteFile(fs_t* fs, int fd, const char* buffer, const int size);	הפונקציה כותבת מידע בגודל נתון לקובץ.
fsReadFile(fs_t* fs, int fd, char* buffer, int* readSize);	הפונקציה קוראת מידע מהקובץ ל-buffer. קריאה נוספת תמשיך מהנקודה בה הפסקנו.
o ע"פ fd שמסופק לפונקציה נשלפים הנתונים הרלוונטיים: מתוך הfileTable והinodeList . o מתוך: (1) גודל הקובץ הקיים (2) size אותו התבקשנו לקרוא\לכתוב, נמצא בעזרת הinode את הblockים המכילים את המידע הרלוונטי עבורנו ומהם נוכל לקורא\לכתוב o במקרה של כתיבה בהחלט יתכן שתצטרכו להקצות blockים חדשים אותם נימצא בעזרת מערך הbitMap	

דוגמא לmain

```
#include "YOUR_H_FILE...."
 define FILENAME "testFS.dat"

int main(int argc, char* argv[])
{
 fs_t *fs;
 char data[BLOCK_SIZE];
 int h1,h2;
 int size;

 //# mount !
 fs = fsMount(FILENAME);

 //# first FORMAT
 res = fsFormat(fs, FILENAME,size);
 printf("Filesystem size=%d\n",size);

 //# create two test files
 h1 = fsCreateFile(fs, "test1.txt");
 h2 = fsCreateFile(fs, "test2.txt");
 char blockSizeData[BLOCK_SIZE] = "I am 001 file  "; // 16 bytes
 size = strlen(blockSizeData)+1;

 //# wite
 for (int t=0;t<BLOCKS_PER_INODE;t++)
  {
   sprintf(blockSizeData+size-3,"%2d",t);
   blockSizeData[size-1]=0;
   res = fsWriteFile(fs, h1,blockSizeData,size);
  }

  //# read
 char zeros[110];
 memset(zeros,0xFF,sizeof(zeros));
 res = fsReadFile(fs, h2,zeros,100,size);

 //# umount
 res = fsUnMount(fs);
}
```

את התרגיל יש להגיש עם הmain הנ"ל ולהוסיף לו את האלמנטים הבאים עם סיום התכנית הצגת פלט:

- מצב מפת הסיביות
- תוכנו של המדריך, וגודלו של כל קובץ, ורשימת הגושים המוקצים לו.

בהצלחה!!