CS342 Operating Systems - Fall 2017
Project 4: My File System (myfs)

Last document update: Dec 4, 2017

Assigned: December 4, 2017
Due date: December 23, 2017, 23:55

Objectives:

- *Learn details and internals of file systems.*
- *Have an experience in designing and implementing a file system.*
- *Practice probability and statistics knowledge.*

You will develop your project in Linux/C. The project can be done in groups of 2. You can do the project individually as well.

In this project you will implement your own simple file system called **myfs**. It will be implemented as a library (static library: **libmyfs.a**). An application linked with that library will be able to use it to create and use files in a virtual disk. The virtual disk will be a Linux file created in your default Linux file system. This Linux file will be acting as the storage that will be managed by myfs file system. In this project we will fix the disk size. Hence your file system design will be for a single disk size which will be 128 MB = 2^27 bytes. Block size will be 4 KB. Hence there will be 2^15 (i.e., 32768) blocks on the disk that will be managed by myfs.

You will design the internals of myfs. You can benefit from the ideas we learned in the class. You may, for example, use the FAT (file allocation table) method or another method that you learned or you design to keep track of the allocated and free blocks. Myfs will use a simple directory organization: there will be one directory (subfolder), namely a root directory, and no other directory.

Your myfs library will implement the following functions:

- ***int myfs_diskcreate (char * diskname).*** When the application calls this, a Linux file called *diskname* that will act as a disk will be created. The size of the Linux file will be 128 MB.
- ***int myfs_makefs(char *diskname).*** This will make a myfs file system on the disk *diskname*. If success, returns 0, otherwise -1. Some number of blocks will be used to store meta information (superblock, root directory, file allocation table, etc.). The rest of the blocks will be used as data blocks that can be allocated to the files. For simplicity you will use ¼ of the disk blocks to store metadata, the rest (¾ of the disk blocks) to store file data. In this way, you will have plenty of blocks to store metadata (more than enough).
- ***int myfs_mount(char *diskname).*** Mount the filesystem. That means prepare it to be used by subsequent file operations. Mounting may involve, for example, getting some of the metadata information from disk into memory, initializing an open file table, etc. Returns 0 if success; otherwise returns -1.
- ***int myfs_umount().*** Unmount the file system. Write back cached metada into disk, if any. After this operation, no file operation can be performed until the file system is mounted again.
- ***int myfs_create(char *filename).*** Create a file with name *filename*. This involves creating/allocating a directory entry and an FCB (file control block). If you wish, you can combine FCB info into the directory entry for the file.

- ***int myfs_open(char *filename).*** Open file *filename*. Returns a non-negative integer file descriptor (file handle) if success. Otherwise returns -1. The file descriptor returned will be used by subsequent operations on the file. It can simply be the index of the openfile table entry allocated for the file that is opened.
- ***int myfs_close(int fd).*** Close the file whose descriptor (handle) is *f*d. If success returns 0, otherwise returns -1.
- ***int myfs_delete(char *filename).*** Delete the file *filename*. If success returns 0, otherwise returns -1. File must be closed before deletion, otherwise deletion will fail. The data blocks used by the file must be deallocated (marked free).
- ***int myfs_read(int fd, void *buf, int n).*** Read n bytes from file *fd*. The read bytes are put into an application buffer (array) *buf* whose size is at least *n* characters (bytes). Space must have been allocated for *buf* by the application. The function will return the number of bytes read (which can be less than *n*), if success; otherwise it will return -1. File position pointer is advanced by the number of bytes read. The max value of *n* is 1024.
- ***int myfs_write(int fd, void *buf, int n*).** Write *n* bytes from application buffer *buf* into file *f*d. The file position pointer will be advanced by the number of bytes written. If success, returns the number of bytes written, otherwise returns -1. The max value of n is 1024. Note that the write operation can extend the file (filesize can incrase).
- ***int myfs_truncate(int fd, int size).*** Reduce the size of the file to size. Deallocate the respective data blocks, if necessary. If size of the file was smaller or equal to the size parameter, the call has no effect. Returns 0 if success, -1 if error. It is not possible to extend the file (increase file size) with truncate operation. Hence the size parameter must be less than or equal to the current size of the file.
- ***int myfs_seek(int fd, int offset).*** Set the file position pointer to *offset*. If *offset* is bigger than or equal to the file size, file position pointer is set to the file size.
- ***int myfs_filesize (int fd).*** Returns the current size of the file.
- ***void myfs_print_dir().*** Print the names of the files in the root directory. One filename per line.
- ***void myfs_print_blocks(filename).*** Print block numbers of the blocks allocated to file *filename*. Format of the output will be like the following (assume there are two files file1 and file2):

  file1: 100 90 300
  file2: 3560 98 2400 567

There can be at most 128 files that can be created on the disk. A process can open at most 64 files at a given time and the maximum filename size is 32 bytes. Use the following constants:

```
#define BLOCKSIZE          4096     // bytes
#define MAXFILECOUNT       128      // max number of files
#define DISKSIZE        (1<<27)     // 128 MB
#define MAXFILENAMESIZE  32         // characters
#define BLOCKCOUNT   (DISKSIZE / BLOCKSIZE)
#define MAXOPENFILES       64      // max number of open files
#define MAXREADWRITE      1024 // bytes; max read/write amount
```

Some initial source files will be provided in the following address so that you can start more easily.

`https://github.com/korpeoglu/cs342fall2017_p4`

Experiments and Report:

So some simple experiments like measuring the elapsed time to read some amount of data (100 bytes, 1000 bytes, 10000 bytes, 100000 bytes, etc.) using one or more read() operations from a file. Similarly you can measure the write() time, opening time, file creation time, etc. Plot some graphs. For a single experiment, do repetitions and take the average. Find out standard deviation of time to perform some operations like reading or writing.

Submission:

Use Moodle.

Clarifications:

- You can use fixed size directory entries.
- You will access the disk in blocks. We are providing you two functions in the library in github: *getblock* and *putblock*. You will use these to access (read or write) a block in the disk. You will not access the disk by other means. Just use these two functions.
- In github, we provide you a program called *createdisk* that can be used to create a virtual disk of the required size (i.e., a Linux file). You can implement the myfs_diskcreate() function by looking to this program.
- The *formatdisk* program in github will be used to format (i.e., make a filesystem) the disk. It is calling the myfs_makefs() function that you will implement.
- There is an *application* provided, app.c, which you can modify. It is a test program. You can develop such applications to test your file system. We will also write our test applications to test and stress your library.
- Do not change the content of myfs.h. This is the header file (interface) that we will include in our app programs. It is the interface of your library.

Provided Code Skeleton:

Makefile:

```
all:  libmyfs.a  app createdisk formatdisk

libmyfs.a:        myfs.c
      gcc -Wall -c myfs.c
      ar -cvq  libmyfs.a myfs.o
      ranlib libmyfs.a

app:  app.c
      gcc -Wall -o app app.c  -L. -lmyfs

createdisk: createdisk.c
      gcc -Wall -o createdisk createdisk.c
```

```
formatdisk: formatdisk.c
	gcc -Wall -o formatdisk formatdisk.c -L. -lmyfs

clean:
	rm -fr *.o *.a *~ a.out app createdisk formatdisk
```

formatdisk.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "myfs.h"

int main (int argc, char *argv[])
{
	char vdiskname [128];

	if (argc != 2) {
		printf ("usage: formatdisk <vdiskname>\n");
		exit (1);
	}

	strcpy (vdiskname, argv[1]);
	myfs_makefs (vdiskname);
	return (0);
}
```

createdisk.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <strings.h>
#include <string.h>

#include "myfs.h"

int main (int argc, char *argv[])
{
	int n, size ,ret, i;
	int fd;
	char vdiskname[128];
	char buf[BLOCKSIZE];
	int numblocks = 0;

	if (argc != 2) {
		printf ("usage: createdisk <vdiskname>");
		exit (1);
	}

	strcpy (vdiskname, argv[1]);
	  size = DISKSIZE;
	numblocks = DISKSIZE / BLOCKSIZE;

	printf ("diskname=%s size=%d blocks=%d\n",
		vdiskname, size, numblocks);
```

```
	ret = open (vdiskname,  O_CREAT | O_RDWR, 0666);
	if (ret == -1) {
		printf ("could not create disk\n");
		exit(1);
	}

	bzero ((void *)buf, BLOCKSIZE);
	fd = open (vdiskname, O_RDWR);
	for (i=0; i < (size / BLOCKSIZE); ++i) {
		printf ("block=%d\n", i);
		n = write (fd, buf, BLOCKSIZE);
		if (n != BLOCKSIZE) {
			printf ("write error\n");
			exit (1);
		}
	}
	close (fd);

	printf ("created a virtual disk=%s of size=%d\n", vdiskname,
size);
	return (0);
}
```

app.c:

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#include "myfs.h"

int main(int argc, char *argv[])
{
	char diskname[128];
	char filename[16][MAXFILENAMESIZE];
	int i, n;
	int fd0, fd1, fd2;	// file handles
	char buf[MAXREADWRITE];

	strcpy (filename[0], "file0");
	strcpy (filename[1], "file1");
	strcpy (filename[2], "file2");

	if (argc != 2) {
		printf ("usage: app <diskname>\n");
		exit (1);
	}

	strcpy (diskname, argv[1]);

	if (myfs_mount (diskname) != 0) {
		printf ("could not mound %s\n", diskname);
		exit (1);
	}
	else
		printf ("filesystem %s mounted\n", diskname);

	for (i=0; i<3; ++i) {
		if (myfs_create (filename[i]) != 0) {
			printf ("could not create file %s\n", filename[i]);
```

```
                exit (1);
        }
        else
                printf ("file %s created\n", filename[i]);
    }

    fd0 = myfs_open (filename[0]);
    if (fd0 == -1) {
        printf ("file open failed: %s\n", filename[0]);
        exit (1);
    }

    for (i=0; i<100; ++i) {
        n = myfs_write (fd0, buf, 500);
        if (n != 500) {
                printf ("vsfs_write failed\n");
                exit (1);
        }
    }

    myfs_close (fd0);

    fd0 = myfs_open (filename[0]);

    for (i=0; i<(100*500); ++i)
    {
        n = myfs_read (fd0, buf, 1);
        if (n != 1) {
                printf ("vsfs_read failed\n");
                exit(1);
        }
    }

    myfs_close (fd0);

    fd1 = myfs_open (filename[1]);
    fd2 = myfs_open (filename[2]);

    myfs_close (fd1);
    myfs_close (fd2);

    myfs_umount();

    return (0);
}
```

vsfs.h:

```
#ifndef MYFS_H
#define MYFS_H

#define BLOCKSIZE          4096     // bytes
#define MAXFILECOUNT       128      // files
#define DISKSIZE         (1<<27)  // 256 MB
#define MAXFILENAMESIZE    32       // characters - max that FS can
support
#define BLOCKCOUNT      (DISKSIZE / BLOCKSIZE)
#define MAXOPENFILES       64      // files
#define MAXREADWRITE      1024     // bytes; max read/write amount

// The following will be use to create and format a disk
```

```
int myfs_diskcreate(char *diskname);
int myfs_makefs (char *diskname);

// The following will be used by a program to work with files
int myfs_mount (char *vdisk);
int myfs_umount();formatdisk.c

int myfs_create(char *filename);
int myfs_open(char *filename);
int myfs_close(int fd);
int myfs_delete(char *filename);
int myfs_read(int fd, void *buf, int n);
int myfs_write(int fd, void *buf, int n);
int myfs_truncate(int fd, int size);
int myfs_seek(int fd, int offset);
int myfs_filesize(int fd);
void myfs_print_dir();
void myfs_print_blocks(char *filename);

#endif
```

vsfs.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#include "myfs.h"

// Global Variables
char disk_name[128];   // name of virtual disk file
int  disk_size;        // size in bytes - a power of 2
int  disk_fd;          // disk file handle
int  disk_blockcount;  // block count on disk


/*
   Reads block blocknum into buffer buf.
   You will not modify the getblock() function.
   Returns -1 if error. Should not happen.
*/
int getblock (int blocknum, void *buf)
{
      int offset, n;

      if (blocknum >= disk_blockcount)
            return (-1); //error

      offset = lseek (disk_fd, blocknum * BLOCKSIZE, SEEK_SET);
      if (offset == -1) {
            printf ("lseek error\n");
            exit(0);

      }

      n = read (disk_fd, buf, BLOCKSIZE);
      if (n != BLOCKSIZE)
```

```
            return (-1);

      return (0);
}


/*
    Puts buffer buf into block blocknum.
    You will not modify the putblock() function
    Returns -1 if error. Should not happen.
*/
int putblock (int blocknum, void *buf)
{
      int offset, n;

      if (blocknum >= disk_blockcount)
            return (-1); //error

      offset = lseek (disk_fd, blocknum * BLOCKSIZE, SEEK_SET);
      if (offset == -1) {
            printf ("lseek error\n");
            exit (1);
      }

      n = write (disk_fd, buf, BLOCKSIZE);
      if (n != BLOCKSIZE)
            return (-1);

      return (0);
}


/*
   IMPLEMENT THE FUNCTIONS BELOW - You can implement additional
   internal functions.
 */


int myfs_diskcreate (char *vdisk)
{
  return(0);
}


/* format disk of size dsize */
int myfs_makefs(char *vdisk)
{
      strcpy (disk_name, vdisk);
      disk_size = DISKSIZE;
      disk_blockcount = disk_size / BLOCKSIZE;

      disk_fd = open (disk_name, O_RDWR);
      if (disk_fd == -1) {
            printf ("disk open error %s\n", vdisk);
            exit(1);
      }

      // perform your format operations here.
      printf ("formatting disk=%s, size=%d\n", vdisk, disk_size);

      fsync (disk_fd);
      close (disk_fd);
```

```
      return (0);
}

/*
   Mount disk and its file system. This is not the same mount
   operation we use for real file systems:  in that the new
filesystem
   is attached to a mount point in the default file system. Here we
do
   not do that. We just prepare the file system in the disk to be
used
   by the application. For example, we get FAT into memory,
initialize
   an open file table, get superblock into into memory, etc.
*/

int myfs_mount (char *vdisk)
{
      struct stat finfo;

      strcpy (disk_name, vdisk);
      disk_fd = open (disk_name, O_RDWR);
      if (disk_fd == -1) {
            printf ("myfs_mount: disk open error %s\n", disk_name);
            exit(1);
      }

      fstat (disk_fd, &finfo);

      printf ("myfs_mount: mounting %s, size=%d\n", disk_name,
            (int) finfo.st_size);
      disk_size = (int) finfo.st_size;
      disk_blockcount = disk_size / BLOCKSIZE;

      // perform your mount operations here

      // write your code

      /* you can place these returns wherever you want. Below
         we put them at the end of functions so that compiler will
not
         complain.
        */
      return (0);
}


int myfs_umount()
{
      // perform your unmount operations here

      // write your code

      fsync (disk_fd);
      close (disk_fd);
      return (0);
}


/* create a file with name filename */
int myfs_create(char *filename)
{
```

```
	// write your code

	return (0);
}


/* open file filename */
int myfs_open(char *filename)
{
	int index = -1;

	// write your code

	return (index);
}

/* close file filename */
int myfs_close(int fd)
{

	// write your code

	return (0);
}

int myfs_delete(char *filename)
{

	// write your code

	return (0);
}

int myfs_read(int fd, void *buf, int n)
{
	int bytes_read = -1;

	// write your code

	return (bytes_read);

}

int myfs_write(int fd, void *buf, int n)
{
	int bytes_written = -1;

	// write your code

	return (bytes_written);
}

int myfs_truncate(int fd, int size)
{

	// write your code

	return (0);
}


int myfs_seek(int fd, int offset)
{
```

```
	int position = -1;

	// write your code

	return (position);
}

int myfs_filesize (int fd)
{
	int size = -1;

	// write your code

	return (size);
}


void myfs_print_dir ()
{

	// write your code

}


void myfs_print_blocks (char *  filename)
{

	// write your code

}
```