Practical Case Study E

Operating Systems Programming – COMP 3015
Operating Systems Programming (Advanced)– COMP 3016

1 Introduction

In this workshop you will be implementing a file system simulator, loosely based on the historic file system used by the CP/M system.

The file system will be have the following properties:

- it is a single level directory system.
- the directory entry has the following format:

  ```
  struct entry
  {
          int8_t user;
          char name[9];
          char extension[4];
          int16_t blockcount;
          int16_t block[24];
  };
  ```

  With the `name` and `extension` fields being C style strings. This structure is 64 bytes in size.
- The disk size is 851 kbyte. (This is the $5\frac{1}{4}''$ inch disk used by the Apple Lisa.)
- The smallest unit of allocation is 512 bytes.
- There are 16 files on the disk, so the directory takes up the first 2 blocks on the disk.
- No control information about it needs to be stored in the directory (i.e. no `.` entry).
- The only user is user 1
- User -1 is not a valid user, and could be used to mark free directory entries.
- Alongside the directory you also need a bitmap that is capable of representing all of the blocks available on the disk, this can be a free space bitmap or an allocation bitmap, this is your choice. This structure is not stored on the disk but would be computed by the operating system when the disk was inserted. Your bitmap will need to track the directory blocks, so they are not allocated to another file.

You are not supposed to implement the actual storage, only the control structures of the file system. When implementing the free bitmap you must use a bitmap, i.e. it should be an array, but each element of the array should represent several blocks.

2 Programming Tasks

When your program starts, it will assume that the disk is unformatted, you should provide a menu that implements the following options:

Initialise Disk initialise disk control structures, setting the blocks allocated for the bitmap to used, and marking all directory entries as being available.

List Files in the Directory List the names, extensions and block counts of all the valid files in the directory.

Display the Free Bitmap print the value of each of the *bits* in the bitmap. This need not be pretty, just a long list of 1's and 0's is sufficient

Open/Create File scans the directory and if the name provided doesn't exist then adds that file to the directory. This file will must be used in all subsequent operations until a new file is open/created or it is deleted.

Read File list the blocks occupied by the currently open file (not the content of these blocks as you don't store this information)

Write File allocate another block to the currently open file. You should not preallocate blocks for the file, you should allocate the first available block, by scanning the bitmap for the first block that is available. Each write shall add another block to the file until there are no more slots to allocate blocks to, or the disk runs out of blocks. (There are only 24 slots available for each file.)

Delete File deallocate all blocks for the current file in the bitmap, and marks as free the directory entry for that file

You need to pay close attention to multiple boundary conditions, which exist in this file system, including the total size of the disk, maximum size of a file, maximum number of files etc. Your program should also allow the disk to be reinitailsed.

3 File: `fs.h`

```c
/* fs.h
* Various definitions for OSP Practical Case Study E
*
* Dr Evan Crawford (e.crawford@westernsydney.edu.au)
* COMP 30015 Operating Systems Programming
* This is the sample file.
*/
#ifndef FS_H
#define FS_H
/* Prevent multiple inclusion with preprocessor guard */
#include<stdint.h>

/* The bitmap */
#define DISKSIZE (851*1024) /* The disk size in bytes */
#define BLOCKSIZE (512) /* The block size */
#define NUMBLOCKS (DISKSIZE/BLOCKSIZE) /* number of blocks on disk */
#define DIR_ENTRIES 16
extern uint8_t bitmap[NUMBLOCKS/8];

/* The directory entry */
struct entry
{
        int8_t user;
        char name[9];
        char extension[4];
        int16_t blockcount;
        int16_t block[24];
};

/* The Directory */
extern struct entry directory[DIR_ENTRIES];
/* extern here means the variable is defined in another
* file, prevents multiple definition errors
*/
int toggle_bit(int block);
/* Toggles the value of the bit 'block', in  the external array 'bitmap'.
* returns the current value of the bit
*
*  Does NOT validate 'block'!!!
*/
int block_status(int block);
/* Returns the status of 'block', in the external array 'bitmap'
* returns 0 if bitmap bit is 0, not 0 if bitmap bit is 1
*
* Does NOT validate block!!!
*/
#endif
```

4 File: `fs.c`

```c
/* fs.c
* Some useful functions for OSP Practical Case Study E
*
* Dr Evan Crawford (e.crawford@westernsydney.edu.au)
* COMP 30015 Operating Systems Programming
* This is the sample file.
*/
#include"fs.h"


uint8_t bitmap[NUMBLOCKS/8];
struct entry directory[DIR_ENTRIES];

int toggle_bit(int block)
{
        int elem = block / 8;
        int pos  = block % 8;
        int mask = 1 << pos;

        bitmap[elem] ^= mask;

        return bitmap[elem] & mask;
}


int block_status(int block)
{
        int elem = block / 8;
        int pos = block % 8;
        int mask = 1 << pos;

        return bitmap[elem] & mask;
}
```

5 File: **main.c**

```
/* main.c
 * Driver for OSP Practical Case Study E
 *
 * Dr Evan Crawford (e.crawford@westernsydney.edu.au)
 * COMP 30015 Operating Systems Programming
 * This is the sample file.
 */
#include<stdio.h>
/* stdio.h will be found in the system path */
#include"fs.h"
/* fs.h will be found in the local path */


int main(int ac, char**av)
{

        printf("Please␣make␣me␣useful\n");

        return 0;

}
```

6 File: `makefile`

```
# makefile -- rules to build OSP workshop C
# Dr Evan Crawford (e.crawford@westernsydney.edu.au)
# COMP 30015 Operating Systems Programming
# Practical Case Study C
# This is the sample file
#
# to use simply type "make"
# this will build the main executable caseE
#
# note, this is a configuration file for the MAKE utility
# do not try to run it directly
# if typing up the file, the indented lines need to be indented
# with TABS not spaces.


all: caseE

caseE: main.o fs.o
	$(CC) -o $@ $^
```