File System

CSC 415.01

Group A

Youssef Hammoud | 921558141 |

Alexander del Rio | 920764010 |

Joshua Alfaro | 918551821 |

Aneida Blanco Palacio | 918466281 |

Github: ajdelrio

Github Link: https://github.com/CSC415-2022-Summer/csc415-filesystem-ajdelrio

Description of our file system:

This assignment is to write a File system. The first phase is the "formatting" of the volume. The second phase is the implementation of the directories-based functions. And the final phase is the implementation of file operations. Our filesystem is as follows :

Since we had the shell created for us, our team only had to focus on three phases for the file system project. In the first phase, we needed to initialize the VCB structure. The **VCB** (volume control block) is a structure that contains important information such as the number of blocks on the volume, the freemap logical block address, and the root directory logical block address. We will also use the VCB to store pointers to the freeMap, root directory, and current working directory for easy access while our filesystem runs. Secondly, we had to initialize our **free space** structure for the implementation.Our free space management uses a bitvector, bitV, to store an array of bytes (uint8_t), whose bits represent blocks on the disk. If a bit is 0, it means the block of the same index is free to use, if it is 1, then that block is in use and not available.
We found it helpful to write a few helper functions to take care of actions we'll commonly need to do:

setBit() takes a byte address and a bit and sets that bit on that byte to 1.

```
void setBit(uint8_t* byte, uint8_t bit) {
    bit = 1 << bit;     // set the bitmask
    *byte |= bit;       // OR byte with the bitmask
}
```

unsetBit() takes a byte address and a bit and sets that bit on that byte to 0.

```
void unsetBit(uint8_t* byte, uint8_t bit) {
    bit = 1 << bit;     // set the bitmask
    bit ^= 0xFF;        // XOR with 0xFF to invert the bitmask
    *byte &= bit;       // AND byte with the bitmask
}
```

isBitFree() takes a byte and a bit and returns 1 if that bit is set to free (0) or 0 if that bit is in use (1).

```c
int isBitFree(uint8_t byte, uint8_t bit) {
    bit = 1 << bit;             // set the bitmask
    if((byte & bit) == 0) {     // if byte & bit == 0, the bit is free
        return 1;
    }
    else {
        return 0;
    }
}
```

allocBlocks() takes a number, tries to find that many free contiguous blocks on the freeMap and if successful, marks those bits as in use on the freeMap and returns the starting LBA of those blocks, if it fails it returns 0, as we know we will not be overwriting the VCB on block 0. The third part of our first phase implementation is our **directory system**.The directory system creates the directories. It creates the root directory and initializes its attributes (root name, size, location, etc.). The date and time the file is created will be stored in dateTimeCr. Each time the file is modified, the date and time of such action will be stored in dateTimeMd. The LBA of the file's first block is stored in the location field. The size field will store the size of the file. The name field will store the name and extension of the file, delimited by "." if applicable. Finally, the attr field will store the attributes of the file.

So far we have 2 attributes we wish to use:

0b00001111 will mean that directory entry is for the root directory

0b00000001 will mean that directory entry is for a directory

For our B_io functions. Our approach is as follows:

Our B_open allows us to open a buffered file. We allocate the buffer using malloc. Then we set out the index to zero since we start at zero. Then we are going to decrement each time the read is called upon. The read will be tracked each time it is called. Then the LBAreads the first block and reads it to the buffer incrementing the position by one each time.

Our B_read allows us to have an interface to read a buffer. We first created a loop to check fd to see if its between 0 and (MAXFCBS-1). Then we create another loop to check to make sure that FCB is active. Then we start our read function with a loop statement, we load a new block if what we requested is larger than what is left in the buffer here we give the difference of count and the remaining buffers. Shortly after we updated the position using "fcbArray[fd].readpos++;". Then we store what's needed from the new memcpy and the block.

After we update the new parameters and we return the # of bytes. Once the filesize is exceeded it will end the program.
Our B_close allows us to free and allocate memory but also places the file into the control block back into the pool that was not used in the file control blocks.

Key functions of our File System:

1. fs_mkdir()
    a. Creates a new directory and updates directory parent into the disk.
2. fs_rmdir()
    a. Removes a directory if it has a valid path.
3. fs_getcwd()
    a. Mallocs a temporary working directory and tokenizes a copy of the directory by finding the root directory.
4. fs_setcwd()
    a. Checks if a given path and name of a directory is valid. If so, set the cwd location to the directory location.
5. fs_isFile()
    a. Checks if a given path is a file.
6. fs_isDir()
    a. Checks if a given path is a directory.
7. fs_delete()
    a. Checks if the path given is a file or directory by calling pathParse(). If so, release the free space of the directory and delete directory data.
8. fs_stat()
    a. Finds the index of a directory entry and fills in the values of an fs_stat structure.
9. fs_createDir()
    a. Inits directory entries in a directory.
10. pathParse()
    a. Checks if a given path is a directory, file, both, or nor.
11. Opendir()

a. It is used to asynchronously open a directory in the file system. It creates an fs.Dir object that is used to represent the directory. This object contains various methods that can be used to access the directory.

12. CloseDir()
    a. The function closes the directory stream indicated by dirp. It frees the buffer that readdir uses.
13. Readdir()
    a. This function is used to read the contents of a given directory. The callback of this method returns an array of all the file names in the directory.

Key structures of our File System:

1.

```
typedef struct {    // Directory Entry Structure
    time_t      dateTimeCr;     // Date and Time the file was created
    time_t      dateTimeMd;     // Date and Time the file was last modified
    uint64_t    location;       // LBA of this file's first block on disk
    uint32_t    size;           // If a directory, number of used entries, otherwise file size
    char        name[35];       // Name and extension (if applicable)
    uint8_t     attr;           // Attributes
} dirEnt_t;
```

2.

```
typedef struct
   {
   /*****TO DO:  Fill in this structure with what your open/read directory needs  *****/
   unsigned short  d_reclen;       /*length of this record */
```

```
   unsigned short  dirEntryPosition;   /*which directory entry position,
like file pos */
   uint64_t    directoryStartLocation;     /*Starting LBA of directory */
   dirEnt_t* dirPtr;
   struct fs_diriteminfo* diInfo;
   } fdDir;
```

3.

```
struct fs_stat
   {
   off_t     st_size;          /* total size, in bytes */
   blksize_t st_blksize;       /* blocksize for file system I/O */
   blkcnt_t  st_blocks;        /* number of 512B blocks allocated */
   time_t    st_accesstime;    /* time of last access */
   time_t    st_modtime;       /* time of last modification */
   time_t    st_createtime;    /* time of last status change */

   /* add additional attributes here for your file system */
   };
```

4.

```
typedef struct {    // VCB structure
   uint64_t    blockSize;      // Number of bytes per block
   uint64_t    numBlocks;      // Number of blocks on this volume
   uint64_t    freeMapAddr;    // LBA of free-space bitmap
   uint64_t    freeLen;        // Number of bytes in freeMap
   uint32_t    mapBlocks;      // Number of blocks the free-space map uses
   uint64_t    rootAddr;       // LBA of the root directory
   uint32_t    rootSize;       // Number of blocks the root directory
occupies
   uint64_t    ourSig;         // Signature for our filesystem
   int         dirLen;         // Number of directory entries per
directory

   /* These are pointers to be used during operation and must be
initialized on
    each run. The values stored on disk will not be valid */
   uint8_t*    freeMap;        // Pointer to the freeMap
   dirEnt_t*   root;           // Pointer to the root directory
```

```
   dirEnt_t*    cwd;            /* Pointer to the current working
directory, set
                                   to the root directory on startup */
   char*        cwdName;        // Pointer to the absolute path name
} vcb_t;
```

5.

```
struct pathInfo {
   enum fType { DNE, File, Dir } lastElem; /* Status of last element in
the path
                                           * DNE  - Does Not Exist
                                           * File - Found and is a file
                                           * Dir  - Found and is a
directory
                                           */
   dirEnt_t* parent; // Pointer to the parent of the last element
   uint64_t parentLoc; // Parent's LBA
   char absPath[MAX_PATH]; // Absolute path that was parsed
   int index; // Index of the last elem in its parent, -1 if DNE
};
```

Issues and Resolutions:

We had many issues with our file system design. Our team had many ideas and it was difficult due to the complications of sharing code and being able to contribute to branches. We attempted to divide our code up but ended up with some code faults such as: not being able to test some of our functions, mix up of who is doing what, and functionality errors.

We resolved these issues by communicating more on discord and testing one by one.

Details of how the driver program works:

First, open the terminal and clone the repository. Then, change the directory into the repository. To run the code, open the terminal either inside or outside of visual studio code and type “make clean”. After “make clean” is done executing, type “make”. Finally, type “make run”. Now, you are ready to prompt a command to the driver.

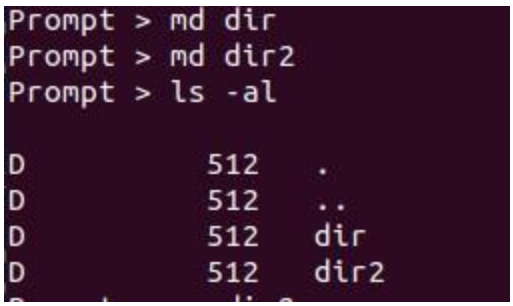
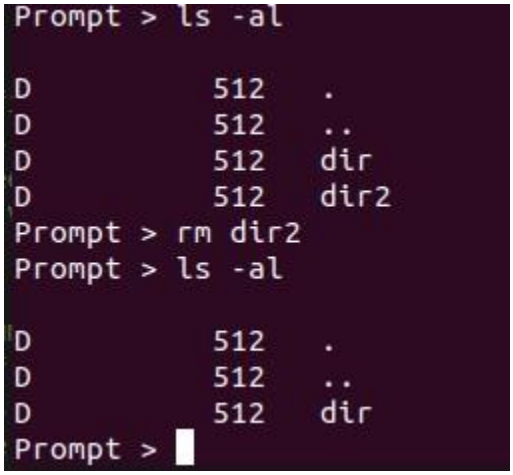
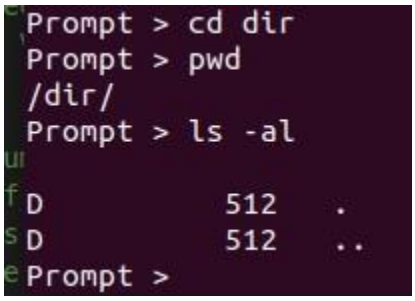
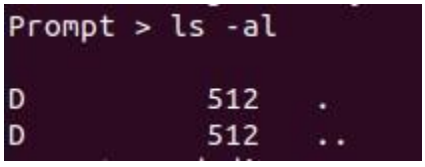
Commands of our driver program:

1. ls: prints a list of the files in a directory
2. cp: copies a file
3. mv: moves a file
4. md: makes a new directory
5. rm: removes a file or directory
6. cp21: copies a file from the test file system to the linux file system
7. cp2fs: copies a file from the linux file system to the test file system
8. cd: changes directory
9. pwd: prints the working directory
10. history: prints the history
11. help: prints out help

Our driver program is a file system that can perform various commands and handle files in a volume. The fsshell.c file has the main function that executes the file system and all command functions to take user inputs and output information and complete operations of the file system that is prompted by the user. Next, the main function calls the function initFileSystem() to reload the free space management if the volume has been formatted. Otherwise, if the volume has not been formatted, the driver program initializes the volume along with initializations of free space management and root directory and writes the volume into the disk. The driver program can then read the volume at the memory level and access data. After that, the driver program would prompt users to enter commands to output information of the file system or process the file and directory I/O Operations. The driver program can finally exit and terminate the shell by calling the functions exitFileSystem() and closePartitionSystem() in the main function.

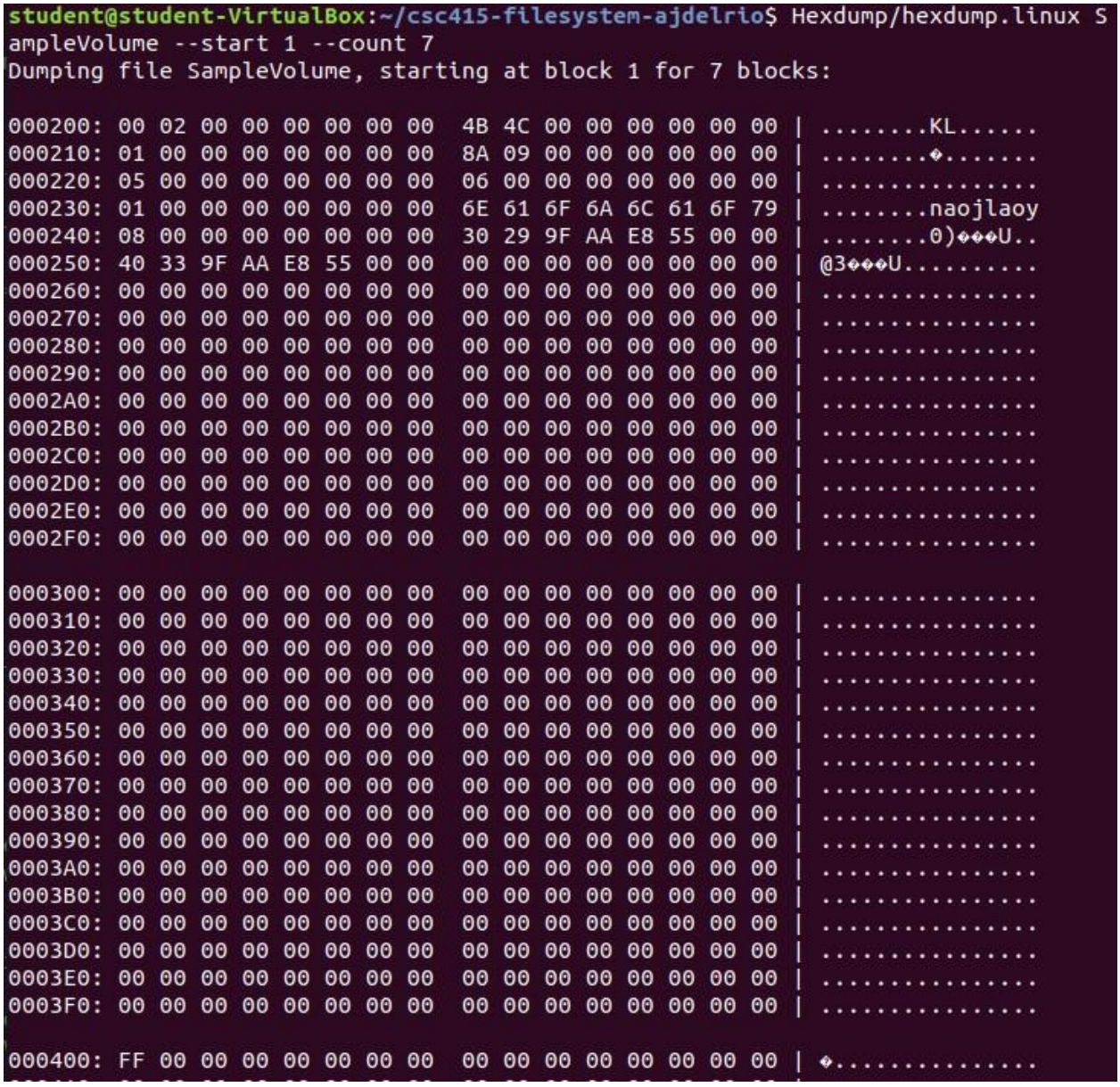
Screenshot of commands:

Cp	
Mv	

Md	
Rm	
Cp21	
Cp2fs	
cd + pwd	
Ls	

Hexdump:

VCB and Free Map (beginning)



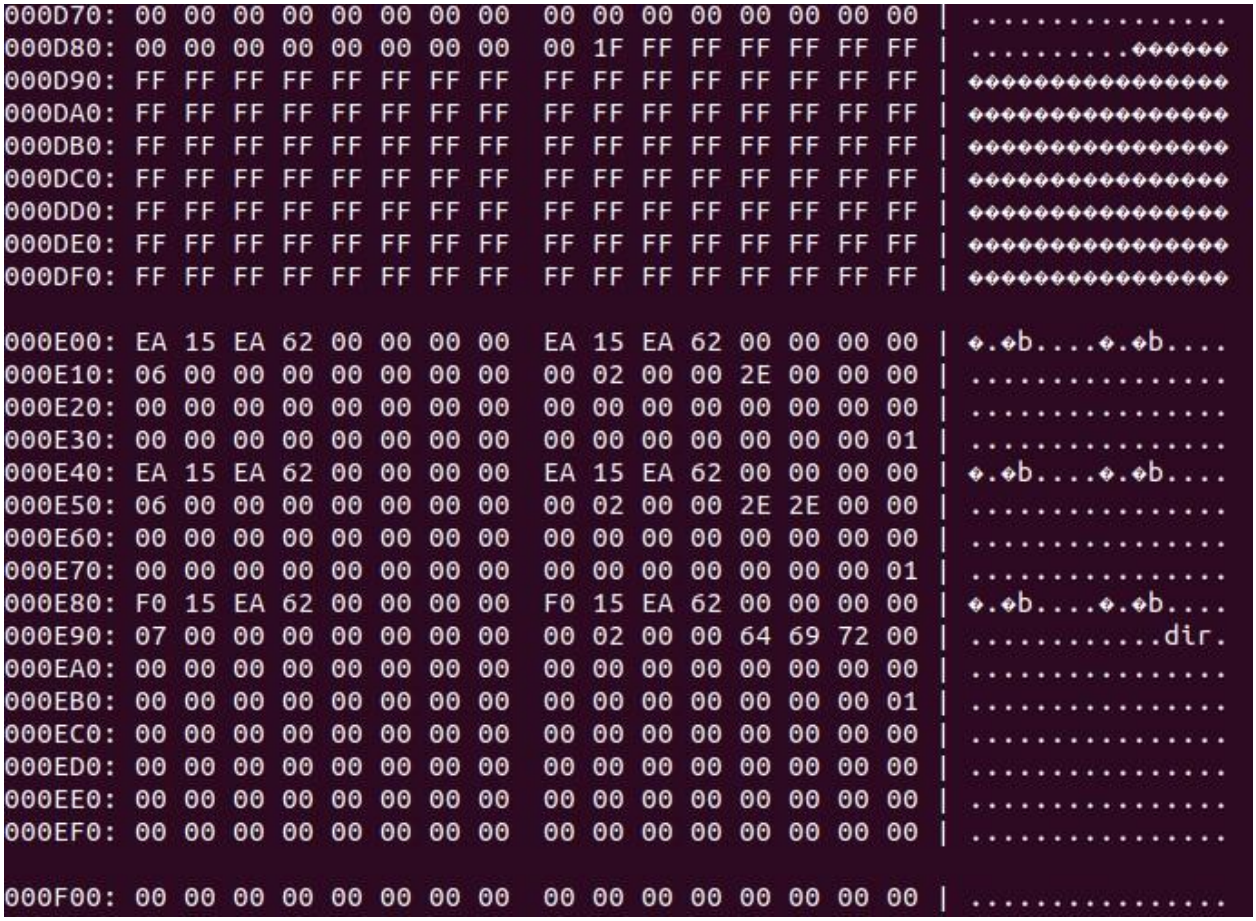
```
student@student-VirtualBox:~/csc415-filesystem-ajdelrio$ Hexdump/hexdump.linux S
ampleVolume --start 1 --count 7
Dumping file SampleVolume, starting at block 1 for 7 blocks:

000200: 00 02 00 00 00 00 00 00  4B 4C 00 00 00 00 00 00 | ........KL......
000210: 01 00 00 00 00 00 00 00  8A 09 00 00 00 00 00 00 | ........◆.......
000220: 05 00 00 00 00 00 00 00  06 00 00 00 00 00 00 00 | ................
000230: 01 00 00 00 00 00 00 00  6E 61 6F 6A 6C 61 6F 79 | ........naojlaoy
000240: 08 00 00 00 00 00 00 00  30 29 9F AA E8 55 00 00 | ........0)◆◆◆U..
000250: 40 33 9F AA E8 55 00 00  00 00 00 00 00 00 00 00 | @3◆◆◆U..........
000260: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000270: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000280: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000290: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0002F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000300: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000310: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000320: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000330: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000340: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000350: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000360: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000370: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000380: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000390: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
0003F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000400: FF 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ◆...............
```

Free Map (end) and root directory



```
000D70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000D80: 00 00 00 00 00 00 00 00  00 1F FF FF FF FF FF FF | ..........◆◆◆◆◆◆
000D90: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000DA0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000DB0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000DC0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000DD0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000DE0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆
000DF0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆

000E00: EA 15 EA 62 00 00 00 00  EA 15 EA 62 00 00 00 00 | ◆.◆b....◆.◆b....
000E10: 06 00 00 00 00 00 00 00  00 02 00 00 2E 00 00 00 | ................
000E20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01 | ................
000E40: EA 15 EA 62 00 00 00 00  EA 15 EA 62 00 00 00 00 | ◆.◆b....◆.◆b....
000E50: 06 00 00 00 00 00 00 00  00 02 00 00 2E 2E 00 00 | ................
000E60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000E70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01 | ................
000E80: F0 15 EA 62 00 00 00 00  F0 15 EA 62 00 00 00 00 | ◆.◆b....◆.◆b....
000E90: 07 00 00 00 00 00 00 00  00 02 00 00 64 69 72 00 | ............dir.
000EA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 01 | ................
000EC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000ED0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
000EF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

000F00: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```