Group Name: French Fry Academy
Rohit Devdhar (SFSU ID: 920487201)
Hugh Nguyen (SFSU ID: 918700515)
Japheth Wun (SFSU ID: 921555593)
Rahima Kakar (SFSU ID: 916192269)

Github link for project:
https://github.com/CSC415-2022-Spring/csc415-filesystem-hughnguyen22

File System Project

Description of our file system:

Our file system is designed around a superblock. This superblock includes a magic number to help us figure out whether we are looking at our unique file system or not. In addition to that our superblock includes a place for keeping file nodes (fnodes), file names (fnames), and a bitmap that keeps track of which blocks are used and which blocks are empty.

In addition to the superblock, our system also includes a linked list of free blocks. This linked list is used to quickly find X number of free blocks (getFsbs() routine).

At file system initialization the initialization code clears all of the entries in the bitmap. After this, the initialization code sets only those bitmap entries that will be occupied by the bitmap and the fnodes and the fnames. This ensures that those blocks belonging to the bitmap, fnodes, and fnames are not used by programs as their datablock.

After this, the initialization code initializes the root directory and creates a directory name entry for it.

Once this is done, the initialization code writes the entire superblock onto the file system. After this, it writes the directory entry for the root directory at the first open block. Then, it increments the fnode and the fname counters by 1. These counters are used by our system to keep track of next fnode and fname entries. In case of a symbolic link (where two file names are pointing to the same physical file), the fname counter will be incremented, however, the fnode counter will remain the same.

Just before the initialization code completes, we run the routine (makeFreeSpaceLL()) to make the linked list of free blocks again. That completes the initialization step.

We also have a routine to get the tokens available in a file path name. This routine helps us get all of the tokens out starting from the top level directory to the bottom most leaf node. We are using this routine to help us make directories and create files.

All of the base routines and structures to implement the file system are mostly in place at this point. However, due to time constraints and other factors, we could not implement all of the routines.

Details of how we would have implemented our driver program

Theoretically, **<u>for a creation of a file,</u>**

- we will need to call the path tokenizer
- then for each entry in the fnames table
    - look up their corresponding fnode
    - look up their datablocks
    - compare each entry with what we need to have in the token and when found, break
- When we find the last directory where we need to place the file in, we use the reclen to place our file name at the right slot in the directory entry.
- We also get the next fnode and fname entry
    - In the fname entry, we write the name of the file and it's permissions as well as other details.
    - In the fname entry, we also write the fnode of this file
    - In the fnode, we write all of the details about this file
- After this, we get the next available FCB (file control block) to the caller.

- In the FCB index field, we write the fnode of this file and we return the FCB to the caller program.

Similarly, **<u>to write a file,</u>** (we have coded the block level write function, however, the fs_write() routine has not been written)

- We will have the FCB in hand as a handle
- Using this handle, we know what the fnode corresponding to this file is.
- First we calculate the number of blocks needed to write the buffer that is passed as part of the write() command.
- We allocate those free buffers using our linked list (getFsbs() routine).
- Then, we use the single block buffer in the FCB to transfer all the data from the file handle to the data blocks on the disk, one block at a time (we will use the written up block I/O routine for this).
- setBit() of all the new blocks written so that they are not used again
- Refresh the free space block linked list
- Then we return the number of bytes written.
- If there is an error, we will return -1

<u>To read a file,</u>

- Using the fd, find the fnode of the file.
- Using the fnode, find the datablocks
- Then use the datablocks and b_io_read function, to read the number of blocks required one at time into the read buffer.
- Return number of bytes read.
- Update file buffer index in the fd so that the next read will start from the next block onwards.

<u>To close a file,</u>

- Closing a file means clearing up the fd(file destructor) from the FCB.
- This releases that entry for some other file
- Return to the caller

<u>To remove a file,</u>

- Use the tokenizer to get each component and then go one by one till the leaf node.
- Get the fnode of the file and then get the numbers of the datablocks.
- Then clear the bitmap entries for those blocks
- Clear the fnode entry

- Clear the fname entry slot
- Then go to the directory entry and remove the file name record from the directory record
- Return to the caller

<u>To copy a file,</u>

- This is nothing but a combination of read and write routines written above.

<u>To symbolically link a file,</u>

- Get the fnode of the file that needs to be linked
- Create an fname entry for the link file
- Write the fnode of the physical file into the fnode field of the fname entry of the symbolic file
- Return to caller

<u>To move a file,</u>

- Get the fnode of the file that is to be moved
- Go to the fname entry of the file and change the name of the file to the new file
- If there is a directory change, also update the appropriate directory entry and insert the name of this new file.
- Remove the old file name from the old directory entry.

To make a directory,

- Use the tokenizer to get all of the entries in the directory path.
- Follow the entries one by one through the fname and fnode tables
- Do this till you get to the leaf node
- Then update the upper level directory records to include the name of the new directory to be created
- Increment rec len field in the directory entry by 1
- Get a new fnode for this directory and a new fname entry
- Enter the name of the directory into the fname field
- Enter the fnode of the directory into the fnode field of fname
- Allot one datablock to this directory entry
- Set bit for the datablock
- Return to caller

To remove a directory,

- Use the tokenizer to find the tokens in the path
- Go one by one until you reach the leaf node
- Find the directory name in the fnames array
- Get its fnode

- In the fnode's entry find the datablocks.
- Clear bits of all the data blocks
- Clear the fnode entry of this directory
- Clear the fname entry of this directory
- Remove this directory's name from the parent directory's directory structure
- Return to caller

Issues we had

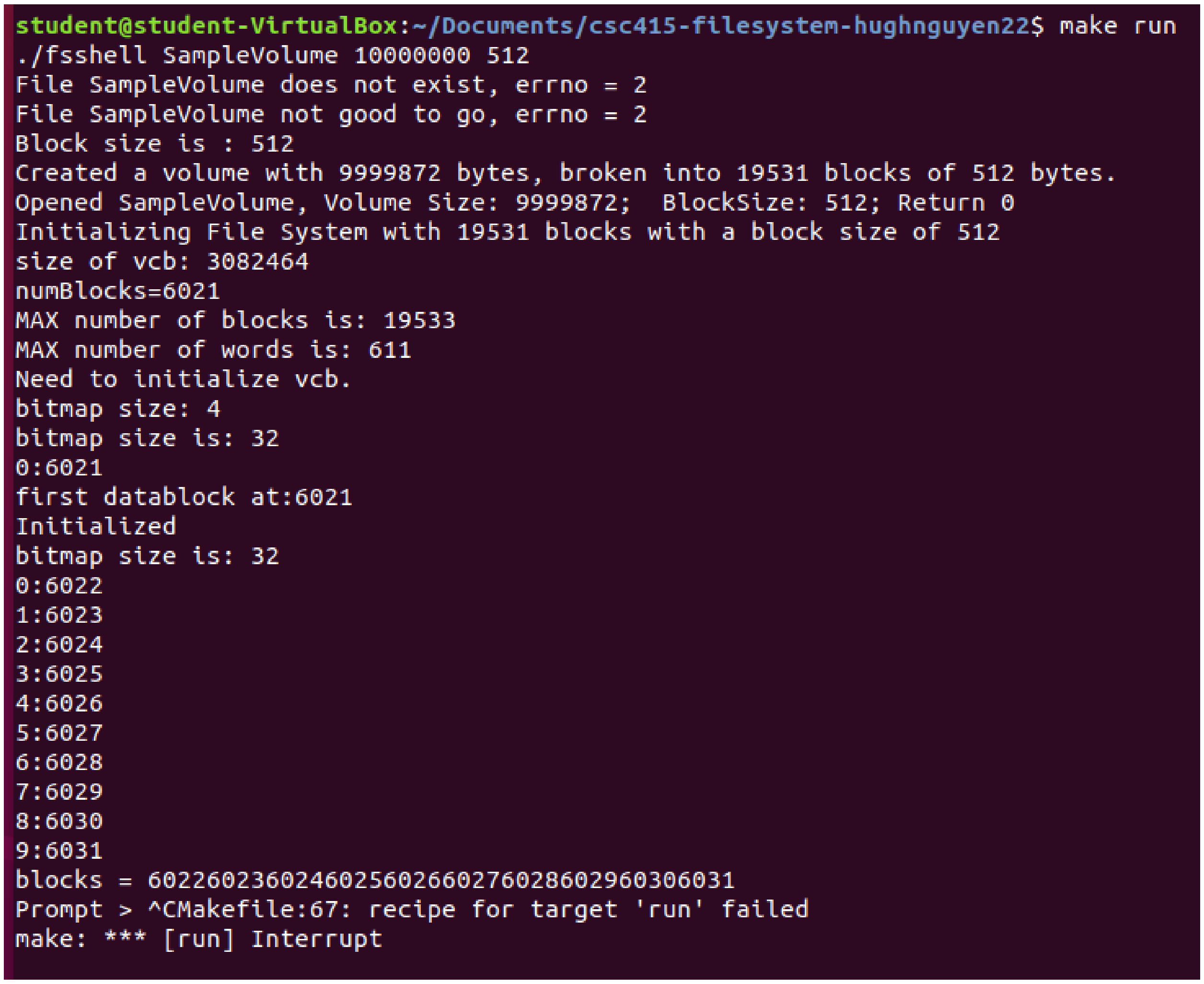
Some issues we had:

- We did not manage our time properly
- It took a little while to see how the SampleVolume and LBAread and LBAwrite were working.
- It was difficult to assess which routines needed change within the block I/O routines.
- It was difficult to see how the block I/O structures were used.
- Much time was spent in deciding whether to implement a bitmap or a linked list in order to manage free space.
- Initially, we were reading only one block but trying to write many

- Because our fnode and fname structures were heavy, they were occupying a significant area of the SampleVolume(nearly 30%)

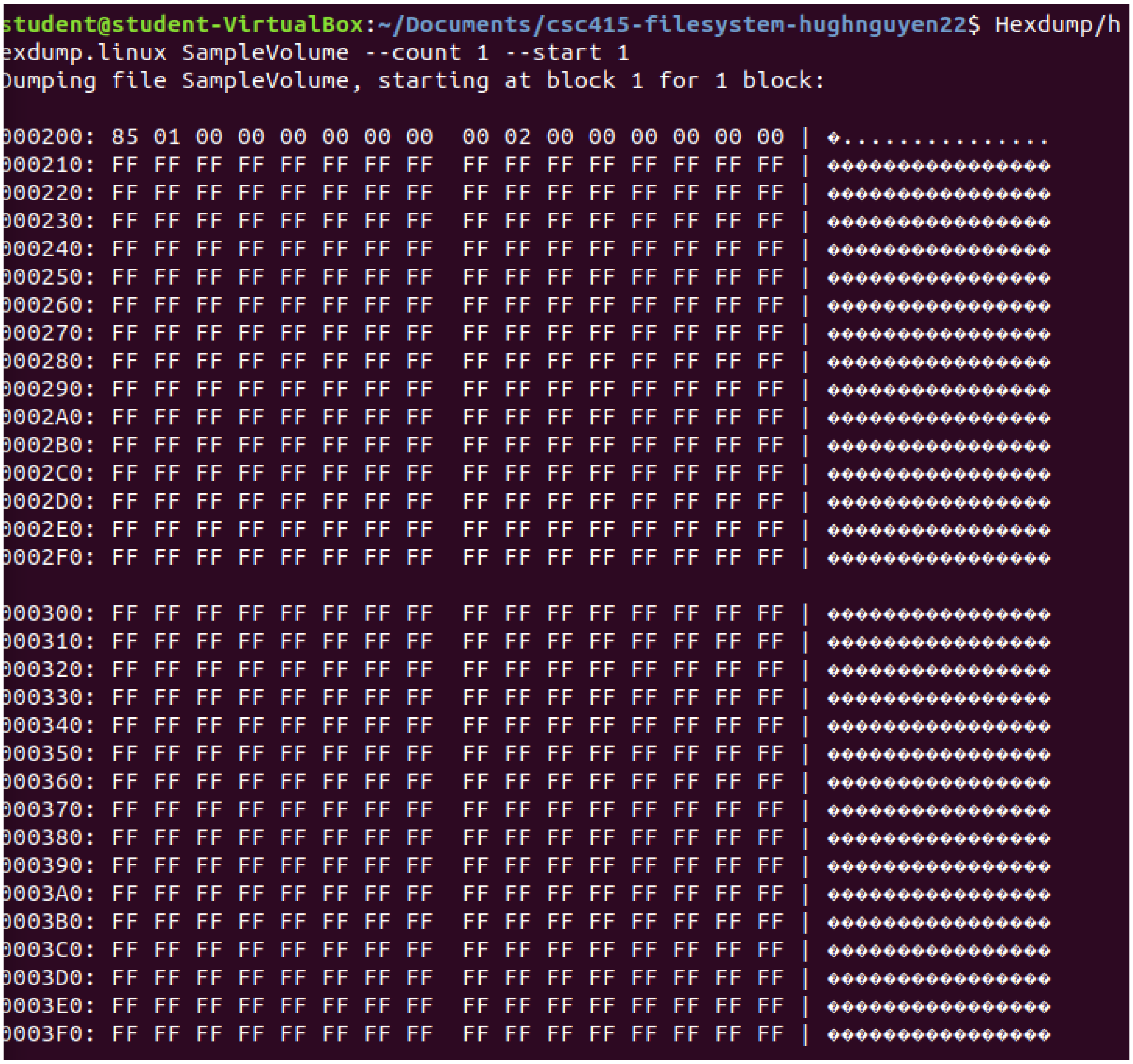
Screenshots:

Screenshot showing initialization messages of the file system, including the first 10 free block numbers after the initialization of the bitmap, fnodes, and fnames in the superblock:



```
student@student-VirtualBox:~/Documents/csc415-filesystem-hughnguyen22$ make run
./fsshell SampleVolume 10000000 512
File SampleVolume does not exist, errno = 2
File SampleVolume not good to go, errno = 2
Block size is : 512
Created a volume with 9999872 bytes, broken into 19531 blocks of 512 bytes.
Opened SampleVolume, Volume Size: 9999872;  BlockSize: 512; Return 0
Initializing File System with 19531 blocks with a block size of 512
size of vcb: 3082464
numBlocks=6021
MAX number of blocks is: 19533
MAX number of words is: 611
Need to initialize vcb.
bitmap size: 4
bitmap size is: 32
0:6021
first datablock at:6021
Initialized
bitmap size is: 32
0:6022
1:6023
2:6024
3:6025
4:6026
5:6027
6:6028
7:6029
8:6030
9:6031
blocks = 6022602360246025602660276028602960306031
Prompt > ^CMakefile:67: recipe for target 'run' failed
make: *** [run] Interrupt
```

Screenshot of the first block that shows part of the bitmap:

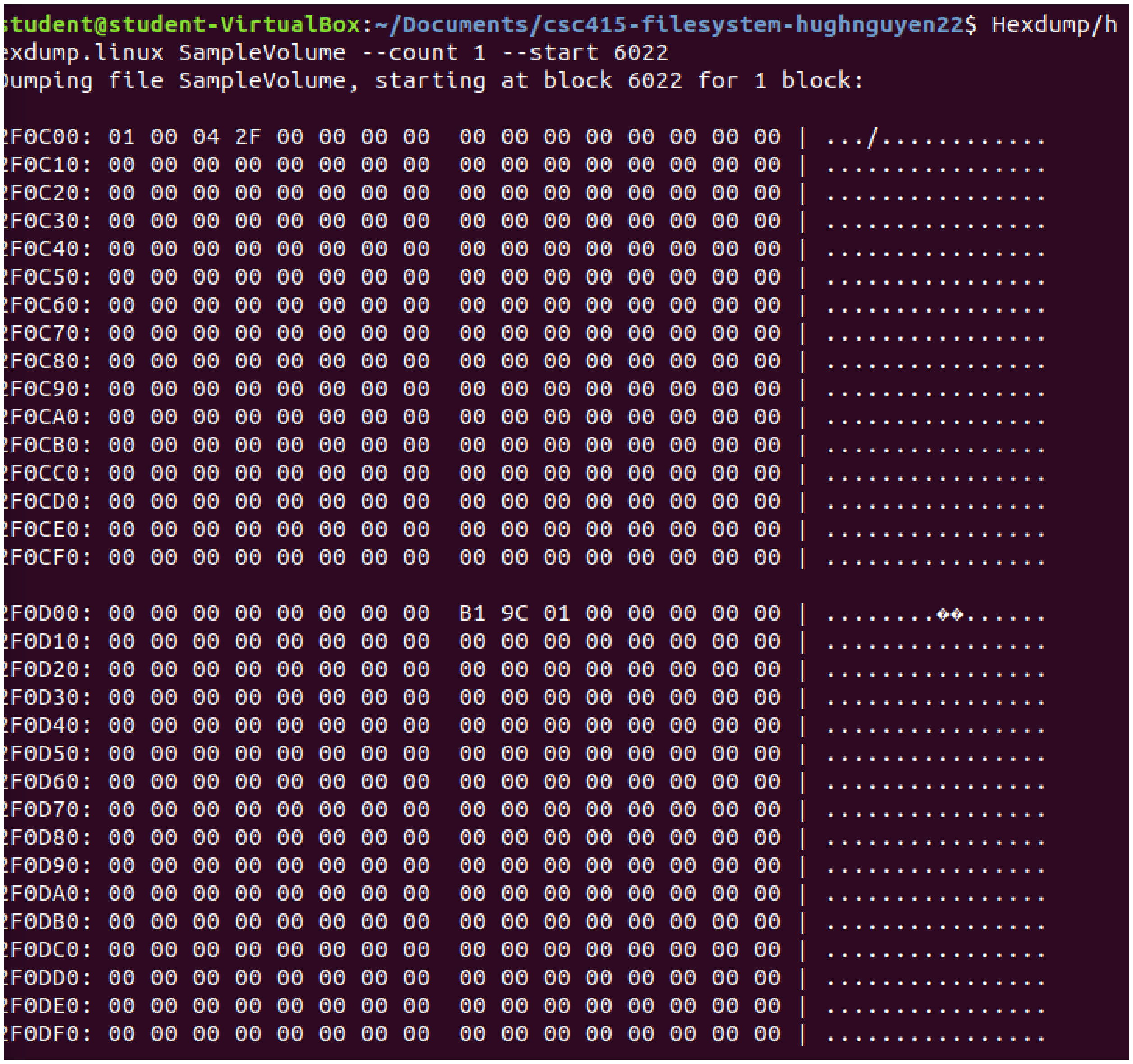


```
student@student-VirtualBox:~/Documents/csc415-filesystem-hughnguyen22$ Hexdump/h
exdump.linux SampleVolume --count 1 --start 1
Dumping file SampleVolume, starting at block 1 for 1 block:

000200: 85 01 00 00 00 00 00 00  00 02 00 00 00 00 00 00 | �...............
000210: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000220: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000230: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000240: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000250: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000260: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000270: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000280: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000290: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0002A0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0002B0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0002C0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0002D0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0002E0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0002F0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������

000300: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000310: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000320: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000330: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000340: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000350: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000360: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000370: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000380: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
000390: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0003A0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0003B0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0003C0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0003D0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0003E0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
0003F0: FF FF FF FF FF FF FF FF  FF FF FF FF FF FF FF FF | ����������������
```

Screenshot of block 6022, where the first directory entry for root has been inserted:



```
tudent@student-VirtualBox:~/Documents/csc415-filesystem-hughnguyen22$ Hexdump/h
xdump.linux SampleVolume --count 1 --start 6022
umping file SampleVolume, starting at block 6022 for 1 block:

F0C00: 01 00 04 2F 00 00 00 00  00 00 00 00 00 00 00 00 | .../............
F0C10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0C90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0CA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0CB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0CC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0CD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0CE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0CF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................

F0D00: 00 00 00 00 00 00 00 00  B1 9C 01 00 00 00 00 00 | ........��......
F0D10: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D20: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D30: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D40: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D50: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D60: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D70: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D80: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0D90: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0DA0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0DB0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0DC0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0DD0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0DE0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
F0DF0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 | ................
```