CS3026 Operating Systems Assignment 2 Virtual Disk

Name: Hasan Ahmedov

ID: 51768277

1. Description

The program is an attempt to implement a simple file system that allows managing files and directories in a virtual in-memory disk. The file system is based on simplified concepts of a File Allocation Table (FAT). It allows the creation of files and directories within this virtual hard disk and the performance of simple read and write operations on files.

The virtual disk is simulated by an array of memory blocks, where each block is a fixed array of bytes. Each block has a block number (from 0 to MAXBLOCKS-1). The allocation of a new block to a file is recorded in the FAT. The FAT is a table (an array of integers) of size MAXBLOCKS that acts as a kind of block directory for the complete disk content: it contains an entry for each block and records whether this block is allocated to a file or unused. The FAT itself is also stored on this virtual disk at a particular location.

2. Interface

The complete public interface of the file system for this virtual disk is the following:

void format()

- Creates the initial structure on the virtual disk, writing the FAT and the root directory into the virtual disk
- Steps:
    - prepare block 0 : fill it with '\0',
    - use strcpy() to copy some text to it for test purposes
    - write block 0 to virtual disk
    - prepare FAT table (it occupies blocks 1 and 2)
    - write FAT blocks to virtual disk
    - set the other FAT entries to UNUSED

  - copy the FAT table
  - prepare root directory
  - write root directory block to virtual disk
  - set the root directory index after the FAT blocks (3)
  - change the current directory to root

MyFILE * myfopen (const char * filename, const char * mode) ;

- Opens a file on the virtual disk and manages a buffer for it of size BLOCKSIZE, mode may be either "r" for readonly or "w" for read/write/append (default "w")
- Steps:
  - check mode and return if not in correct one
  - assign entry variable to hold initial directory
  - if the path is absolute then set the current directory to root
  - assign tokenization string
  - assign pointer to remaining part of the string
  - copy the path to the string (to be used for string tokenization)
  - assign a token pointer
  - tokenize the string and go over the tokens
  - if there are still tokens remaining
    - get the file entry index (if the entry index is EOF then exit) and set the current directory index to the file entry index
  - allocate memory for the file
  - reset file position to 0
  - set all elements to "\0"
  - set file mode
  - check if the opened file is in read mode (exit if file not found)
    - set file block number and return the file
  - if we find the file on the virtual disk
    - set to the block number of the file (if the path block number is less than the root directory number, then exit)
    - adjust FAT chain and copy changes to FAT
  - create a buffer block and starting from root directory (while we haven't reached EOC)
    - set the current directory index
    - put the virtual disk data to the buffer block

        - assign buffer block entry list to the current entry list
        - find the entry in the buffer block matches the file name and clean it
    - set the directory as used
    - if file not found
        - get the next unused block
        - create a buffer block and transfer current directory block to it
        - allocate memory for the new entry
        - initialize the unused entry as a file
        - get unused directory entry number
        - assign buffer block entry list to the current entry list
        - place the unused entry to the block entry list and set its name to the token
        - write the block to the virtual disk
        - set the file block number
        - go back to initial directory and return the file

void myfclose (MyFILE * stream)

- Closes the file, writes out any blocks not written to disk
- Steps:
    - if the file is in write mode
        - get the next unused block
        - extend the FAT block chain with the next unused block
        - write the buffer block

int myfgetc (MyFILE * stream)

- Returns the next byte of the open file, or EOF (EOF == -1)
- Steps:
    - assign a variable which holds the current block number
    - check whether the block number is EOC or the file is in read mode and return if not
    - if the writer pointer has reached BLOCKSIZE
        - get next buffer block number from the FAT block chain
        - copy the virtual disk data to the buffer block used for reading

        - reset character writer pointer
    - return the read character and increment write pointer

void myfputc (int b, MyFILE * stream)

- Writes a byte to the file. Depending on the write policy, either writes the disk block containing the written byte to disk, or waits until block is full
- Steps:
    - return if the file is in read mode
    - if the write pointer is BLOCKSIZE
        - get the next unused block
        - assign the next FAT entry to point to the next unused block
        - reset the write pointer to the beginning of the stream
        - write the stream buffer to the virtual disk
        - clean the buffer data
        - assign the next unused block to buffer
    - write the character to the file and increment the write pointer position

void mymkdir (const char * path)

- The function creates a new directory, using path, e.g. mymkdir ("/first/second/third") creates directory "third" in parent dir "second", which is a subdir of directory "first", and "first is a sub directory of the root directory
- Steps:
    - declare a buffer block
    - assign entry variable to hold initial directory
    - if the path is absolute then set the current directory to root
    - assign tokenization string
    - assign pointer to remaining part of the string
    - copy the path to the string (to be used for string tokenization)
    - assign a token pointer
    - tokenize the string and go over the tokens
        - get file entry index
        - if the entry index is not EOF, then set the current directory index to the file entry index
        - set buffer block to free directory block

        - get the next unused directory location where the new directory will be placed
        - place the token as the directory entry name
        - initialize the directory entry and write it to the virtual disk
    - revert the directory to initial directory index

char ** mylistdir (const char * path)

- Lists the content of a directory and returns a list of strings, where the last element is NULL
- Steps:
    - assign entry variable to hold initial directory
    - if the path is absolute then set the current directory to root
    - assign tokenization string
    - assign pointer to remaining part of the string
    - copy the path to the string (to be used for string tokenization)
    - assign a token pointer
    - tokenize the string and go over the tokens
        - get the entry index of the file which matches the token (if there is no such match then exit)
        - set the current directory index to the file entry index
    - create a buffer block
    - assign a FAT buffer entry to current directory
    - allocate a character array that will hold the names of files and folders
    - while the FAT buffer entry is not EOC
        - move the FAT block to the buffer block
        - assign buffer block entry list to the current entry list
        - find an used entry in the directory block
        - allocate memory of size file name length
        - copy the file name to the list where we just allocated memory
        - extend the list length so that we have an entry for EOF
    - set last element of the array to NULL (EOF entry)
    - revert to the initial directory and return the list with the contents of the directory

void mychdir (const char * path)

- The function changes the current directory into an existing directory, using path, e.g. mkdir ("/first/second/third") creates directory "third" in parent dir "second", which is a subdir of directory "first", and "first is a sub directory of the root directory
- Steps:
    - if the path is "root" or starts with "/"then return to the root directory
    - if the path is "." don't change directory and return
    - assign tokenization string
    - assign pointer to remaining part of the string
    - copy the path to the string (to be used for string tokenization)
    - assign a token pointer
    - declare variable to hold the file entry index
    - tokenize the string and go over the tokens
        - get the file entry index
        - return if it's EOF
        - set the current directory index to the file entry index
        - switch the global currentDir to the current directory index

void myremove (const char * path)

- Removes an existing file, using path, e.g. myremove ("/first/second/third/testfile.txt")
- Steps:
    - check if path is "/" and if so return (since root directory cannot be removed)
    - assign tokenization string
    - assign pointer to remaining part of the string
    - copy the path to the string (to be used for string tokenization)
    - assign a token pointer
    - tokenize the string and go over the tokens
        - if there are still tokens remaining set the current directory index to the file entry index
        - get the file (to be removed) index at token
        - create a buffer block and move the current directory to it
        - initialize a clean entry
        - take the first block number of the entry to be removed

        - adjust the FAT chain and set the block number to NULL
        - put the clean entries to the buffer block
        - copy changes to FAT
        - write the buffer block to the virtual disk
    - go back to initial directory

void myrmdir (const char * path)

- Removes an existing directory, using path, e.g. myrmdir ("/first/second/third") removes directory "third" in parent dir "second", which is a subdir of directory "first", and "first is a sub directory of the root directory
- Steps:
    - check if path is "/" and if so return (since root directory cannot be removed)
    - assign tokenization string
    - assign pointer to remaining part of the string
    - copy the path to the string (to be used for string tokenization)
    - assign a token pointer
    - tokenize the string and go over the tokens
        - get the file entry number
        - if there are still remaining path directories then enter the one we are considering set the current directory index to the file entry index
        - get the directory entry index which we will be removing
        - create a buffer block
        - move directory data to buffer block so we can delete it
        - if the directory is empty reuse "myremove" function to delete the directory else print an error message
    - go back to initial directory

Helper functions used:

void printBlock (int blockIndex)

- Prints a block number and data (for testing)

void copyFAT ()

- Copies the content of the FAT into one or more blocks

void moveToBlock (diskblock_t * block, int block_address)

- Moves virtual disk data to the block data

int nextUnusedBlock ()

- Returns the next unused block number
- Steps:
    - Iterate over the FAT by starting at block 4 since previous blocks are occupied by the FAT and root directory
        - set the next unused block to EOC
        - return the unused block number

int nextUnusedDir ()

- Returns the next unused directory entry number
- Steps:
    - create a buffer block and set a buffer FAT entry to the current directory index
    - while the FAT entry buffer hasn't reached an EOC
        - move the FAT entry data to the buffer block
        - go through the buffer block directory entries
        - check if the block in the entry list is unused
        - if so set the current directory to that FAT entry
        - return the block number of that entry
    - get the next unused block number and include it to the FAT chain
    - move the current directory to the newly placed FAT entry
    - initialize buffer block to 0
    - write the block to the virtual disk
    - set current directory entry to point to the newly placed FAT entry entry list
    - since we extended the FAT chain return entry number 0

int fileLocation (const char *filename)

- Looks for a file block given a path (file name)
- Steps:
    - create a buffer block and set a buffer FAT entry to the current directory index
    - while the FAT entry buffer hasn't reached an EOC

        - shift the current directory to the next one
        - move virtual disk block data to buffer block
        - set the global currentDir to the buffer block entry list
        - go through the buffer block directory entries
        - check if the name of an entry matches the file name
        - if so return the block of that entry
    - if there is no such file then return FALSE

void cleanFAT (int fatEntry)

- Recursively cleans FAT by setting FAT entries to UNUSED
- Steps:
    - create a buffer block and initialize it to 0
    - get the next FAT entry and if it is not EOC, recursively call cleanFAT() on that entry
    - provide a new empty block by overwriting the FAT entry with EOC in the virtual disk
    - set next FAT entry to unused

diskblock_t freeDirBlock ()

- Returns a newly created free directory block
- Steps:
    - create a buffer block, initialize it to 0 and make it a directory block
    - create and initialize a free entry
    - go through the entry list and set all the entries to the free entry
    - return the free directory block

int fileEntryIndex (const char *path)

- Checks if there is an entry that matches the path. If so returns the entry number and EOF otherwise
- Steps:
    - create a buffer block, initialize it to 0 and make it a directory block
    - while current FAT entry is not EOC
        - move the virtual disk directory data block to the buffer data block
        - check directory entries until we find one with a name that matches the path

            - if found set the current directory to the path and return the block number
        - if no match is found return EOF

int isDirEmpty (dirblock_t dir)

- Checks whether an entry is empty (unused). Returns TRUE if it finds such entry and FALSE otherwise

void printContents (char **contentList)

- Prints the contents of a list (in our case files and folder names)

<u>Extra functions implemented:</u>

void myfvdsys(const char* sysFile, const char* vdFile)

- Copies the contents of a virtual disk file to a physical disk file
- Steps:
    - open a system system file in write mode
    - open a virtual disk file in read mode
    - copy the contents of the virtual disk file to the system file
    - close both files

void myfsysvd(const char* sysFile, const char* vdFile)

- Copies the contents of a physical disk file to virtual disk file
- Steps:
    - open a system system file in read mode
    - open a virtual disk file in write mode
    - copy the contents of the physical disk file to virtual disk file
    - close both files

void myfcopy (char* src, char* dst)

- Copies content from source to destination file (in the virtual disk)
- Steps:
    - open a source file in read mode
    - open a destination file in write mode
    - copy the contents of the source file to destination file
    - close both files

void myfmove (char* src, char* dst)

- Moves content from source to destination file (in the virtual disk) and deletes source file (reuses "myfcopy" and "myremove" functions)

3. How to run

(!) Before you start make sure u have all the files required in the same directory: **filesys.c**, **filesys.h**, **shell.c** and **Makefile**.

On Windows:

1) You will need to install C compiler if you don't have one, such as GCC (https://gcc.gnu.org/). If you are on a Windows computer, you can run the command '**gcc -v**' to check if it's already installed.

2) Open the command prompt by going to the **Start button** and typing '**cmd'** in the search or run bar (or click on Command Prompt if provided).

3) Change your directory to where you have your C program (in our case **filesys.c**, **filesys.h**, **shell.c** and **Makefile**.) using the command '**cd**'. We need to pass the name of the directory in which the program is stored.

    - **Example:** cd Desktop (if the program is already in the user directory)

4) Compile the source code by typing '**make**' in the Command Prompt (compilation is automatically done by the **Makefile**). (additional command '**make clean**' to clean up the compilation files)

5) Run the executable file by typing the name of the executable file without the extension (in our case '**shell**') and hit '**Enter**'.

On Linux:

1) Open the terminal window.

2) If you don't have a compiler already installed you will need to run the following apt-get commands in the terminal to install GNU c/c++ compiler:

```
$ sudo apt-get update
$ sudo apt-get install build-essential manpages-dev
```

3) Navigate to the program directory using the command '**cd**' and type in '**make**' to compile the program (since we have a **Makefile**). (additional command '**make clean**' to clean up the compilation files)

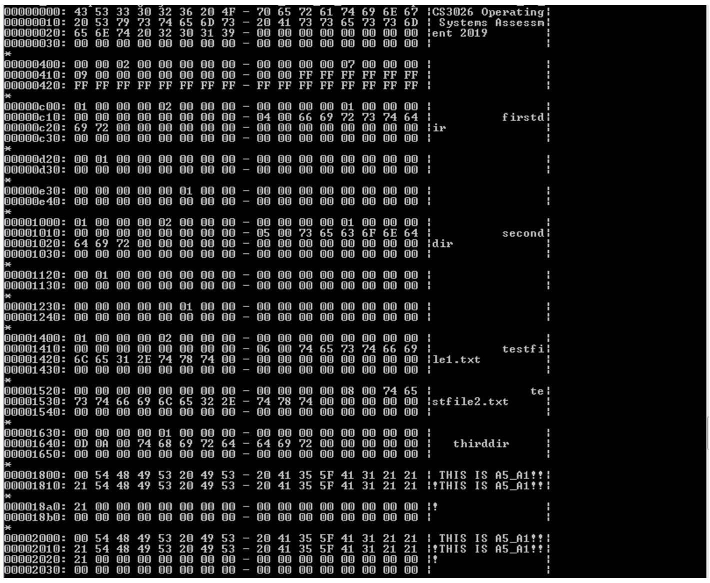
4) To run the executable just type in '**./<name of file>**' ('**./shell**' in our case) and hit '**Enter**'.

In case **Makefile** is missing you can compile the program manually using the following commands:

```
$ gcc -c filesys.c
$ gcc -c shell.c
$ gcc -o shell shell.o filesys.o
```

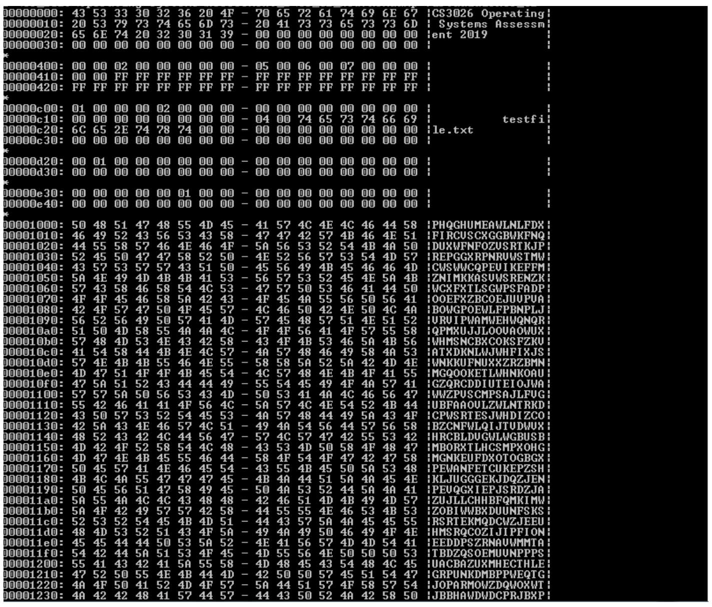
4. Sample output

Figure 1. Hexdump of "virtualdiskA5_A1_a"



```
00000000: 43 53 33 30 32 36 20 4F - 70 65 72 61 74 69 6E 67 ¦CS3026 Operating¦
00000010: 20 53 79 73 74 65 6D 73 - 20 41 73 73 65 73 73 6D ¦ Systems Assessm¦
00000020: 65 6E 74 20 32 30 31 39 - 00 00 00 00 00 00 00 00 ¦ent 2019        ¦
00000030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00000400: 00 00 02 00 00 00 00 00 - 00 00 00 00 07 00 00 00 ¦                ¦
00000410: 09 00 00 00 00 00 00 00 - 00 00 FF FF FF FF FF FF ¦                ¦
00000420: FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF ¦                ¦
*
00000c00: 01 00 00 00 02 00 00 00 - 00 00 00 00 01 00 00 00 ¦                ¦
00000c10: 00 00 00 00 00 00 00 00 - 04 00 66 69 72 73 74 64 ¦          firstd¦
00000c20: 69 72 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦ir              ¦
00000c30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00000d20: 00 01 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00000d30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00000e30: 00 00 00 00 00 01 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00000e40: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001000: 01 00 00 00 02 00 00 00 - 00 00 00 00 01 00 00 00 ¦                ¦
00001010: 00 00 00 00 00 00 00 00 - 05 00 73 65 63 6F 6E 64 ¦          second¦
00001020: 64 69 72 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦dir             ¦
00001030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001120: 00 01 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00001130: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001230: 00 00 00 00 00 01 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00001240: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001400: 01 00 00 00 02 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00001410: 00 00 00 00 00 00 00 00 - 06 00 74 65 73 74 66 69 ¦          testfi¦
00001420: 6C 65 31 2E 74 78 74 00 - 00 00 00 00 00 00 00 00 ¦le1.txt         ¦
00001430: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001520: 00 00 00 00 00 00 00 00 - 00 00 00 00 08 00 74 65 ¦              te¦
00001530: 73 74 66 69 6C 65 32 2E - 74 78 74 00 00 00 00 00 ¦stfile2.txt     ¦
00001540: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001630: 00 00 00 00 01 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00001640: 0D 0A 00 74 68 69 72 64 - 64 69 72 00 00 00 00 00 ¦   thirddir     ¦
00001650: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001800: 00 54 48 49 53 20 49 53 - 20 41 35 5F 41 31 21 21 ¦ THIS IS A5_A1!!¦
00001810: 21 54 48 49 53 20 49 53 - 20 41 35 5F 41 31 21 21 ¦!THIS IS A5_A1!!¦
*
000018a0: 21 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦!               ¦
000018b0: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00002000: 00 54 48 49 53 20 49 53 - 20 41 35 5F 41 31 21 21 ¦ THIS IS A5_A1!!¦
00002010: 21 54 48 49 53 20 49 53 - 20 41 35 5F 41 31 21 21 ¦!THIS IS A5_A1!!¦
00002020: 21 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦!               ¦
00002030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
```

Figure 2. Hexdump of "virtualdiskC3_C1"



```
00000000: 43 53 33 30 32 36 20 4F - 70 65 72 61 74 69 6E 67 ¦CS3026 Operating¦
00000010: 20 53 79 73 74 65 6D 73 - 20 41 73 73 65 73 73 6D ¦ Systems Assessm¦
00000020: 65 6E 74 20 32 30 31 39 - 00 00 00 00 00 00 00 00 ¦ent 2019        ¦
00000030: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00000400: 00 00 02 00 00 00 00 00 - 05 00 06 00 07 00 00 00 ¦                ¦
00000410: 00 00 FF FF FF FF FF FF - FF FF FF FF FF FF FF FF ¦                ¦
00000420: FF FF FF FF FF FF FF FF - FF FF FF FF FF FF FF FF ¦                ¦
*
00000c00: 01 00 00 00 02 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00000c10: 00 00 00 00 00 00 00 00 - 04 00 74 65 73 74 66 69 ¦          testfi¦
00000c20: 6C 65 2E 74 78 74 00 00 - 00 00 00 00 00 00 00 00 ¦le.txt          ¦
00000c30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00000d20: 00 01 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00000d30: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00000e30: 00 00 00 00 00 01 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
00000e40: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 ¦                ¦
*
00001000: 50 48 51 47 48 55 4D 45 - 41 57 4C 4E 4C 46 44 58 ¦PHQGHUMEAWLNLFDX¦
00001010: 46 49 52 43 56 53 43 58 - 47 47 42 57 4B 46 4E 51 ¦FIRCVSCXGGBWKFNQ¦
00001020: 44 55 58 57 46 4E 46 4F - 5A 56 53 52 54 4B 4A 50 ¦DUXWFNFOZVSRTKJP¦
00001030: 52 45 50 47 47 58 52 50 - 4E 52 56 57 53 54 4D 57 ¦REPGGXRPNRVWSTMW¦
00001040: 43 57 53 57 57 43 51 50 - 45 56 49 4B 45 46 46 4D ¦CWSWWCQPEVIKEFFM¦
00001050: 5A 4E 49 4D 4B 4B 41 53 - 56 57 53 52 45 4E 5A 4B ¦ZNIMKKASVWSRENZK¦
00001060: 57 43 58 46 58 54 4C 53 - 47 57 50 53 46 41 44 50 ¦WCXFXTLSGWPSFADP¦
00001070: 4F 4F 45 46 58 5A 42 43 - 4F 45 4A 55 56 50 56 41 ¦OOEFXZBCOEJUVPVA¦
00001080: 42 4F 57 47 50 4F 45 57 - 4C 46 50 42 4E 50 4C 4A ¦BOWGPOEWLFPBNPLJ¦
00001090: 56 52 56 49 50 57 41 4D - 57 45 48 57 51 4E 51 52 ¦VRVIPWAMWEHWQNQR¦
000010a0: 51 50 4D 58 55 4A 4A 4C - 4F 4F 56 41 4F 57 55 58 ¦QPMXUJJLOOVAOWUX¦
000010b0: 57 48 4D 53 4E 43 42 58 - 43 4F 4B 53 46 5A 4B 56 ¦WHMSNCBXCOKSFZKV¦
000010c0: 41 54 58 44 4B 4E 4C 57 - 4A 57 48 46 49 58 4A 53 ¦ATXDKNLWJWHFIXJS¦
000010d0: 57 4E 4B 4B 55 46 4E 55 - 58 58 5A 52 5A 42 4D 4E ¦WNKKUFNUXXZRZBMN¦
000010e0: 4D 47 51 4F 4F 4B 45 54 - 4C 57 48 4E 4B 4F 41 55 ¦MGQOOKETLWHNKOAU¦
000010f0: 47 5A 51 52 43 44 44 49 - 55 54 45 49 4F 4A 57 41 ¦GZQRCDDIUTEIOJWA¦
00001100: 57 57 5A 50 56 53 43 4D - 50 53 41 4A 4C 46 56 47 ¦WWZPVSCMPSAJLFVG¦
00001110: 55 42 46 41 41 4F 56 4C - 5A 57 4C 4E 54 52 4B 44 ¦UBFAAOVLZWLNTRKD¦
00001120: 43 50 57 53 52 54 45 53 - 4A 57 48 44 49 5A 43 4F ¦CPWSRTESJWHDIZCO¦
00001130: 42 5A 43 4E 46 57 4C 51 - 49 4A 54 56 44 57 56 58 ¦BZCNFWLQIJTVDWVX¦
00001140: 48 52 43 42 4C 44 56 47 - 57 4C 57 47 42 55 53 42 ¦HRCBLDVGWLWGBUSB¦
00001150: 4D 42 4F 52 58 54 4C 48 - 43 53 4D 50 58 4F 48 47 ¦MBORXTLHCSMPXOHG¦
00001160: 4D 47 4E 4B 45 55 46 44 - 58 4F 54 4F 47 42 47 58 ¦MGNKEUFDXOTOGBGX¦
00001170: 50 45 57 41 4E 46 45 54 - 43 55 4B 45 50 5A 53 48 ¦PEWANFETCUKEPZSH¦
00001180: 4B 4C 4A 55 47 47 47 45 - 4B 4A 44 51 5A 4A 45 4E ¦KLJUGGGEKJDQZJEN¦
00001190: 50 45 56 51 47 58 49 45 - 50 4A 53 52 44 5A 4A 41 ¦PEVQGXIEPJSRDZJA¦
000011a0: 5A 55 4A 4C 4C 43 48 48 - 42 46 51 4D 4B 49 4D 57 ¦ZUJLLCHHBFQMKIMW¦
000011b0: 5A 4F 42 49 57 57 42 58 - 44 55 55 4E 46 53 4B 53 ¦ZOBIWWBXDUUNFSKS¦
000011c0: 52 53 52 54 45 4B 4D 51 - 44 43 57 5A 4A 45 45 55 ¦RSRTEKMQDCWZJEEU¦
000011d0: 48 4D 53 52 51 43 4F 5A - 49 4A 49 50 46 49 4F 4E ¦HMSRQCOZIJIPFION¦
000011e0: 45 45 44 44 50 53 5A 52 - 4E 41 56 57 4D 4D 54 41 ¦EEDDPSZRNAVWMMTA¦
000011f0: 54 42 44 5A 51 53 4F 45 - 4D 55 56 4E 50 50 50 53 ¦TBDZQSOEMUVNPPPS¦
00001200: 55 41 43 42 41 5A 55 58 - 4D 48 45 43 54 48 4C 45 ¦UACBAZUXMHECTHLE¦
00001210: 47 52 50 55 4E 4B 44 4D - 42 50 50 57 45 51 54 47 ¦GRPUNKDMBPPWEQTG¦
00001220: 4A 4F 50 41 52 4D 4F 57 - 5A 44 51 57 4F 58 57 54 ¦JOPARMOWZDQWOXWT¦
00001230: 4A 42 42 48 41 57 44 57 - 44 43 50 52 4A 42 58 50 ¦JBBHAWDWDCPRJBXP¦
```