7-1-1984

A disk operating system for forth

Somchai Tupavong
Atlanta University

Follow this and additional works at: http://digitalcommons.auctr.edu/dissertations

Part of the Mathematics Commons

Recommended Citation
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>i</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>iii</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION TO FORTH</td>
<td>1</td>
</tr>
<tr>
<td>2. THE DISK SUBSYSTEM</td>
<td>5</td>
</tr>
<tr>
<td>3. SYSTEM OPERATION AND COMMANDS</td>
<td>8</td>
</tr>
<tr>
<td>4. EDITOR COMMANDS</td>
<td>22</td>
</tr>
<tr>
<td>5. SETTING UP A SYSTEM</td>
<td>26</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>A-1</td>
</tr>
<tr>
<td>BIBLIOGRAPHIES</td>
<td>B-1</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I wish to express my deepest appreciation, especially for my advisor, Dr. Bennett Setzer. This thesis could not be accomplished without his support and patience. Furthermore, I would like to extend my respect to the following Atlanta University Faculty:

Dr. Benjamin J. Martin
Dr. Johnny L. Houston
Dr. Nazir A. Warsi
Dr. Arthur M. Jones
Dr. Henry Gore
Dr. Scott Owen
Dr. Ronald Biggers
Prof. Grover Simmons
Mr. John Hsang Kao

Those instructors have been extremely helpful to me for all the time I have been Atlanta University student.
ABSTRACT

COMPUTER SCIENCE

Somchai Tupavong  B.S. Chulalongkorn University,
Bangkok, Thailand, 1977

DISK OPERATING SYSTEM FOR FORTH

Advisor : Dr. Bennett Setzer
Thesis date : July, 1984

A disk operating system is the heart of any computer system. It should be powerful and easy to use. A disk operating system for FORTH language lacks utilities to handle the normal operating system constructs common in computer system. For example, no options are available to have named files, to query disk space, to run named programs, and so on.

This thesis tries to add some of the utilities to the system so that any FORTH user may find this system easier and more powerful to use than the original one.
INTRODUCTION

This thesis concerns implementing a disk operating system for the FORTH language. FORTH has a kind of disk operating system, but it lacks some features which are quite normal or standard in other disk operating systems. FORTH does not support named files and does not provide allocation of disk space. The object of the system that we implement is to solve these problems and provide the user of this system a more complete and powerful disk operating system.

The program implemented is converted from an original program written by Peter Reece (BYTE, April 1982) for the TRS-80. The target system is a Z-89.

The overall design of the system has not been changed much. However, several modifications were needed to work with the new system, and some improvements have been added: All commands of this system will be loaded with the system whereas in the original system those commands would have been loaded separately when needed. An editor (by Dr. Bennett Setzer) has been modified and added to the program to help keep track of the files. We
have also left out all printing routines from the original program because of the differences in the basic system. All read, write and copy commands have been modified to suit the new system.

This gives a broad view of this program, the details of the program will be discussed in the sequel including all the modifications mentioned above.
CHAPTER 1

INTRODUCTION TO FORTH

"FORTH is a language that has been developed for years. Mainly it has been used with a small computer but it also can be used in a large application in a large computer. Corresponding time in FORTH is fast and the program will use less memory compared to other high-level language." [1, p. B-1]

FORTH begins with a set of standard commands (words) which you can use to define your own commands. These commands also can be defined in assembly language, using FORTH's assembler, in order to increase speed and decrease memory requirements.

FORTH's words or commands are composed of other words previously defined, so you can think of a FORTH program as a series of calls to previously defined
procedures.

There are several components to the FORTH system that the words manipulate. In the following, we describe these components: the dictionary, the parameter stack, the return stack, disk storage, and the interpreter.

One of the most important parts of the FORTH system is a dictionary. A dictionary contains all FORTH words and their definitions. When a word is defined by a user, it will be placed in the dictionary. Variable and constant names also are placed in the dictionary.

To handle numbers, FORTH uses two stacks: the "parameter stack" and the "return stack". The parameter stack is normally used more often by a FORTH programmer, it is the "working" storage for FORTH. Also, it holds parameters being passed from word to word. The return stack holds pointers which the system uses to locate the next step of the program. The return stack can be used to hold values temporarily while performing operations on the parameter stack. Normally, the return stack cannot be used to pass parameters from one word to another.

In FORTH, words normally execute immediately, operating on data stored at the top of the parameter
stack. So, data must already be placed on the stack before the command is given. This means that arithmetic expressions are entered in reverse polish notation. To perform a calculation such as an addition, the two numbers must already be on the stack before the add command is given. As an example, if you want to add 2 and 5, first you have to put the number 2 onto the stack, then the number 5 and follow this by typing the plus sign ('+' is the name of the addition command).

The addition command will take the first two numbers from the top of the stack, perform an addition and put the result (in this case, 7) back onto the stack.

In FORTH, disk memory is divided into units called "blocks" or "screens". Each block holds 1024 bytes. The 1024 characters are divided, for display, into 16 lines of 64 characters each. The transferring of data between disk memory and computer memory will consist of a whole block. The system reserves buffer areas in memory to accept these transferred data.

FORTH uses an interpreter to interpret a program. The interpreter will interpret a source program by searching for each word of the program in the dictionary.
If the word is found, its definition will be executed. If the word is not found, the system attempts to interpret it as a number. If this is successful, the number is put onto the stack. If this is not successful, the system indicates an error.
CHAPTER 2

THE STANDARD DISK SUBSYSTEM

As mentioned in CHAPTER 1, FORTH divides disk storage into blocks. To transfer data from disk to internal memory, a whole block must be transferred in each operation. The FORTH system has buffer areas already allocated in memory to accommodate this.

Most FORTH systems use one primitive command to actually cause data to transfer between main memory and the disks. This is the R/W command. This command assumes that there are three values on the parameter stack: addr, blk, and f. (f is assumed to be on top). The command will transfer data between the block numbered blk and the 1024 byte area in memory beginning at addr. If f is 1, the transfer is from disk to memory; if f is 0, the transfer is from memory to disk. R/W removes the three
values from the stack and leaves nothing on the stack. R/W can transfer the data to and from any given address (normally, one of the system buffers, but this need not be so).

The usual way an application transfers data from disk memory to the main memory is to use the BLOCK command. This command will take the first value on the parameter stack as a designated block to transfer. The transfer will be from a given block of the disk to an I/O buffer area in the main memory. BLOCK removes the first value from the stack and leaves the starting address of the buffer on top of the stack. The BLOCK command checks if the desired block is already in memory, if so, no physical I/O need take place. Buffer allocation is handled automatically by the system.

The UPDATE command can be used to cause the transfer of data from main memory to a disk. This command will mark a buffer in main memory as updated. When the buffer is required by another block (normally by BLOCK command), the updated block will be written back onto the disk. Note that no data is written to the disk by this command.

The FLUSH command causes the system to transfer any
updated buffers in main memory back onto the disk.

This disk operating system works quite well, but it still lacks some controls that the user may need, such as write protection on some blocks or the ability to transfer a series of blocks by name without knowing the number of every block in the series.
The system we implemented consists of a number of
FORTH word definitions. At present, these must be loaded
after booting up the basic FORTH system. LOADDIR command
is the command used to load this system. At the end of
the load the HELP is automatically invoked to show a
brief description of all the system commands.

This system allows user to use named files (instead
of keeping track of each block), to run a program by name
(instead of loading of the first block of the program),
to protect a file (instead of no protection of a block at
all), to edit a file by name (instead of by block). It
also automatically allocates space to files (instead of
using a manual method) and allows non-contiguous program
files (instead of loading each separate block or to have
to use consecutive blocks).

To keep track of the activity of each block, this system uses a bitmap. A bitmap is a bit array in which each bit represents the availability of a corresponding disk block: a 0 bit means the block is free while a 1 bit means the block is used by some file. The length of the bitmap can be determined by dividing the disk size (total blocks on a disk) by 8 (1 byte) plus 1 (for the remainder). This bitmap will represent the activity of each block. This same method can be used to keep track of activity on each file. Each file has a bitmap which has the same size as the master bitmap and works similarly. However, instead of keeping track of all the blocks on the disk, in this map, a bit will be set only when the block has been allocated to that file. When the bit in the file map is set, the corresponding bit in the master bitmap will be set also. With these maps the system can determine which block is free to be allocated to a file. The system will allocate the next free block to the requested file, it may not be the one next to the last block of that file but it will not allocate any block which preceeds that last block. The system will search
the file map from bit 0 to the last bit of that map for the bit that was set to 1. The position of the last 1 on the file map will represent the last block of that file (since a free block below that position will never be allocated to the requested file). And from that last position, the system will search in the master bitmap for next available block to allocate to that file.

To read a sequential file is just to follow the map of that file and load the blocks that correspond to the on bit of the map. That is, the order of blocks within a file is the same as the natural ordering from the disk. However, the blocks allocated to a file need not be consecutive on the disk. And to write such a file is just adding the next available block from the bitmap to that file.

This system uses a directory as a place to keep data about the files in the disk. The directory will be placed at block number 99 (the last block) of the disk using this system. As mentioned before, a block is divided into 16 lines of 64 characters each. In a directory block the first line of the block will contain the name of the disk, the second line will contain the bit map of the
disk and each of the last 14 lines will contain a file name, its' map and a protection flag. The name of the disk can be 64 characters long (how to provide it will be discussed under the SETDIR command later). A file name can be 15 characters long but the system will distinguish only the first four characters for speed. In the directory, after the name of a file, the map of that file will follow and the last (sixty-fourth) character of that line will contain a flag (*) to show if the file is protected or not. The system allows 14 files to each disk since there are 14 free lines available to a directory block.

The following is the list of all the available features (commands) and the description of the function of each.

- Loading a directory (DMOUNT)

This command will load the directory from a disk (block 99) into the main memory at a reserved area (an array 1024 characters long). This command is given automatically by the LOADDIR command. This command is used to change a disk without loading the system all over.
again.

- Setup a directory ( SETDIR )

This command will prompt

New Directory

Build a new directory ( & destroy all files )

ENTER 1-IF YES 0-IF NO

You reply with 1 if you want to build a new directory or 0 if you do not. A message

ERROR TRY AGAIN

will be shown if the character you enter is neither 1 nor 0 and the system will let you try again. If you enter 1 a message

WRITE PROTECT BELOW BLOCK#

will be shown which at this time you can enter a number of the block that you want the blocks below it to be lock out from this system. Then a message

Block# to be locked out (9999 if none)

will be shown which at this time you can enter any single block number you which to lock from this system and this message will keep repeating until you enter 9999.
- Create a file (MAKE)

This command will allow you to create a new file. First MAKE will ask for a name of the file by prompting up
FILENAME:
which at this time you can name up to 15 characters. A message
*** File already exists ***
will be shown if the named file already existed in the directory or a message
*** File name space is full ***
will be shown if the disk already has 14 files.

- Destroy a file (KILL)

This command will allow you to destroy any file and create a space for future use. First KILL will prompt up
FILENAME:
which at this time you can enter a name of the file you want to destroy. A message
*** File not found ***
will be shown if the file you entered is not in
the directory.

- Write protection a file (PROTECT)

This command marks a file to prevent the system from writing on the file. PROTECT will prompt up FILENAME:

which at this time you can enter a name of the file you want to protect. A message

*** File not found ***

will be shown if the file is not in the directory.

Note that, this command cannot protect a block if you use normal BLOCK, FLUSH or R/W commands.

- Unprotect a file (UNPROTECT)

This is the opposite command of the PROTECT command. It will unprotect any protected file by prompting up FILENAME:

which at this time you can enter a name of the file you want to unprotect. A message

*** File not found ***

will be shown if the file is not in the directory.
- Rename a file (RENAME)

This command will ask for a name of the file you want to change by prompting up

OLD FILENAME:

A message

*** File not found ***

will be shown if the file is not in the directory or a message

NEW FILENAME:

will be shown if the file is found which at this time you can enter its new name. A message

*** File already exists ***

will be shown if the name you entered is already in the directory.

- Name a file you want to work on (NAME)

This command will ask you for a name of a file by prompting up

FILENAME:

which at this time you can enter a name of the file you want to work on. This command must be given before using any of the following commands: WRITE, READ,
RWRITE, RREAD, SHOW and BLOCKS. These commands will refer automatically to the file named by this command.

- Sequential write (WRITE)

WRITE (addr --> flag) Writes the 1024 bytes starting at addr to the next block in the 'NAMED' file. A flag will be left on the parameter stack to show that the file is protected or not (1 if the file is protected, 0 if not).

- Sequential read (READ)

READ (--> addr) Read a block of the 'NAMED' file into the main memory and leave the starting address on top of the parameter stack. This command will read from the first block of the file on first called and the second block on the second call of the command and so on until end of file is reached which it will put 0 on top of the parameter stack. To read the whole file you must use this command within some kind of loop and keep checking for the end of file (0) and do not forget to do whatever you wish with the block before using another READ command, since it may put a new content into that
area of memory.

- Random write ( RWRITE )

  RWRITE ( addr  block# --> flag ) This command is used the same way as WRITE command. The different is that you have to specify the relative block number of the file you want to write on, this command will not add a new block to the file. Note that the old data on the block will be destroy when you use this command.

- Random read ( RREAD )

  RREAD ( block# --> addr ) This command will read the specify block into the system memory and leave the starting address of the memory on the stack. To use this command you have to specify which block of the file you wish to read before using the command.

- To look into and edit a file ( SHOW )

  Before enter this command, you must enter a number of relative block of the 'NAMED' file you wish to see or edit ( first block, second block, etc. ). The system then will show that block onto the screen for you to look
at the contents of that block. While the block is shown on the screen you can edit it as you normally do when using an EDITOR which will be discuss in detail in the next chapter. At the end of the file, the system will prompt

**Do you want to add a new block**

**ENTER 1-IF YES 0-IF NO**

which at this time you can enter 1 if you want to add a new block to that file or 0 which a message **End of file**

will be shown. A message **ERROR TRY AGAIN**

will be shown if the character you entered is not a 1 or a 0 and the system will allow you to try again.

- Directory information (FILES)

  This command will show you name of the files in the directory, how many blocks each of these files took, which files are protected and how many blocks are left in the disk.

- Files information (BLOCKS)
This command will show numbers of the block that has been used by the 'NAMED' file.

- Run a program (RUN)

This command will prompt up

FILENAME:

which at this time you can enter a name of the file you want to load (run). A message

*** File not found ***

will be shown if the file is not in the directory.

- Copy a file (FCOPY)

This command will prompt up

Copy from

for you to enter a name of a file you want to make a copy. A message

*** File not found ***

will be shown if the file is not in the directory.

Then

to

will show up for you to enter a name of the file you want to copy to. A message
New file

will be shown if the name is not in the directory.

A message

Copy over old file?

ENTER 1-IF YES 0-IF NO

will be shown if the name is already in the directory, you can respond to this message by entering 1 if you want to copy it over an existing file or entering 0 if you do not want to and the system will get out of this command.

- Description of commands (HELP)

This command will show a list of all the command provided by this system onto the screen with a brief description.

- Get out of the system (END)

This command is the only one that you have to use every time you use this system. It will save and update the directory of what you have done, if you forget to use this command before you get off the terminal, all the things you have done at that time will not be recorded on
the directory, so do not forget to use this command every time you use this system.
CHAPTER 4

EDITOR COMMANDS

The following are the commands you can use to edit your source text on the disk. These commands can be used when you are in the editing mode (when you use SHOW command). Note: esc is the escape function key.

- Up-arrow command key - This command will move the cursor up one line but not beyond the first line.

- Down-arrow command key - This command will move the cursor down one line but not further than the sixteenth line.

- Right-arrow command key - This command will move the cursor one position to the right along the line but
not beyond the sixty-fourth character.

- Left-arrow command key - This command will move the cursor one position to the left but it will not move the cursor further if you already are at the first position of the line.

- Home command key - This command will move the cursor to the starting position of the screen, that is at the first position of the first line.

- Command key no. 1 on a key pad - This command will insert a blank line at the cursor position and automatically move the following line down one line each. Note that if you have any data in the sixteenth line, it will be lost.

- Command key no. 3 on a key pad - This command will delete a line at the cursor position and move the rest of the line up one line each.

- Command key no. 9 on a key pad - This command will
delete a character at the cursor position and move the
rest of the character one position to the left.

- Command key no. 7 on a key pad - This command will
insert a blank at the cursor position and move the rest
of the character one position to the right. Note that if
you already have the sixty-fourth character, it will be
lost.

- Red function key - This command key will allow you
to save the contents on the screen back onto the disk
(providing that the file is not protected) and get out of
the edit mode. A message "File is protected" will be
shown if the file you are working is a write protect
file.

- Gray function key - This command key will allow
you to save the contents on the screen back onto the disk
(if the file is not protected) and continue work in the
edit mode. A message "File is protected" will be shown if
the file is a write protect file.
- F-4 function key - This command key will allow you to move to the next screen of the file. See SHOW command on chapter 3 if you reach the end of a file.

- F-5 function key - This command key will allow you to move back to the previous screen but not further than the first screen.

These commands work the same way as the original editor except that instead of automatic updating, these commands will wait until you key in any of the save commands before marking the block as updated so the system can check if the file is protected or not first. The save commands then have been changed so that they can check for protected file before updating anything. The other changes are in the last two commands, instead of working with the consecutive blocks it has been changed to be able to work with the blocks of the file which may be consecutive one or it may be not by putting all the blocks of the file onto an array first and get these block number for the editor from this array.
CHAPTER 5

SETTING UP A SYSTEM

To use this system first you have to format the disk to correspond to this system. A system disk (volume 31) that contains this thesis program already has a format program under the file name INIT. To format a disk you just follow the following steps.

- (insert system disk)
- 31 MOUNT
- 5 LOAD 20 LOAD
- RUN (see CHAPTER 3)
- INIT
- v TSSET (v = volume of the new disk)
- 0 0 4 VSET (then insert new disk)
- FORMAT (reply Y)
After formatting a disk then you have to create (or copy) the FORTH system onto the new disk. You can do this by following these steps.

- (insert system disk)
- 0 MOUNT
- 0 BLOCK 1 BLOCK 2 BLOCK 3 BLOCK 4 BLOCK
- (insert new disk that has been formatted)
- UPDATE FLUSH
- (insert system disk)
- 9 LOAD
- (insert new disk)
- SYSSAVE

Now you are ready to generate the directory system by following these steps.

- (insert system disk)
- 31 MOUNT
- 5 LOAD
- LOADDIR
- ( insert new disk )
- v MOUNT ( v volume of the new disk )
- SETDIR ( see CHAPTER 3 )

Now you have the directory for the new disk and can start editing the disk. When you finish editing do not forget to use command END before taking the disk out of the computer. Eventhough the contents of the files you edited are saved, the data about the files themselves will be lost if you forget to use the END command.

The directory will be on the block number 99 on the disk while you are using this system this block will not be touched but if you use a different editor on the disk and want to keep the directory, be sure not to edit block 99.
Block # 20

1  ( #1 EDITOR --- 5.10.82 )
2  DECIMAL
3  0 VARIABLE FSCR  0 VARIABLE CK
4  0 VARIABLE FCLIN  0 VARIABLE ADDR
5  0 VARIABLE FCCOL  0 VARIABLE SCRN
6  0 VARIABLE FDEPTH
7  PAD 1+ VARIABLE FTOP
8  : FESC 27 EMIT ;
9  : FCLEAR FESC ." E" ;
10 : FPOSIT FESC ." y" 32 + EMIT 32 + EMIT ;
11 : FROW 0 SWAP FPOSIT ;
12 : FREPOS FCCOL @ 4 + FCLIN @ 2 + FPOSIT ;
13 : FCHAD 64 * + FSCR @ DUP SCRN ! BLOCK DUP ADDR ! + ;
14 : FCURAD FCCOL@ FCLIN@ FCHAD ;
15  -->

Block # 21

1  ( #2 EDITOR --- 5.10.82 )
2  : FVECTOR <BUILDS DUP + ALLOT DOES> OVER + + ;
3  24 FVECTOR FCNDS
4  : FCNDS -FIND 0= 17730 ?ERROR DROP CFA SWAP FCNDS ! ;
5  : FCNDS-1 OVER < OVER 24 < AND
6  IF FCNDS @ 1 ELSE DROP 0 ENDIF ;
7  -->
8
9
10
11
12
13
14
15
16

Block # 22

1  ( #3 EDITOR --- 5.10.82 )
2  : .FLINE DUP 2 + FROW DUP 1 + 2 .R SPACE SPACE
3  0 SWAP FCHAD 64 TYPE ;
4  : .FSCREEN FCLEAR ." Block " FSCR @ .
5  16 0 DO I .FLINE LOOP ;
6  7
7  : FRFRSH FCLIN@ .FLINE FREPOS ;
8  9
9  : FHOME 0 FCCOL ! 0 FCLIN ! FREPOS ;
10  : FREST. FSCREEN FHOME ;
11 12
13 14
14  -->
15
16
Block # 23

( #4 EDITOR --- 5.10.82 )

1 : FUP FCLIN @ IF -1 FCLIN +1 FREPOS ENDIF ;
2 : FDOWN FCLIN @ 15 < IF 1 FCLIN +1 FREPOS ENDIF ;
3 : FLEFT FCCOL @ IF -1 FCCOL +1 FREPOS ENDIF ;
4 : FRIGHT FCCOL @ 63 < IF 1 FCCOL +1 FREPOS ENDIF ;
5 : FNEWLIN FDOWN 0 FCCOL ! FREPOS ;
6 : FTAB FCCOL @ 5 + 5 / 5 * DUP 64 < IF FCCOL ! FREPOS ELSE DROP ENDIF ;
7 : FREPLACE DUP EMIT 8 EMIT FCURAD C! FRIGHT ;
8 -->

Block # 24

( #5 EDITOR --- 5.10.82 )

1 : FDEL FLEFT 32 FREPLACE FLEFT ; 0 VARIABLE FLAG
2 : FEDITOR PAD 1+ FTOP ! 0 FDEPTH !
3 FSCR ! FRESET FESC ." x6"
4 BEGIN KEY
5 DUP 127 - IF DROP FDEL ELSE
6 DUP 31 > IF FREPLACE
7 ELSE DUP 27 = IF
8 DROP KEY 64 - FCMNDIF EXECUTE ENDIF
9 ELSE DUP 13 = IF DROP FNEWLIN
10 ELSE 9 = IF FTAB
11 ENDF ENDF ENDF ENDF ENDF AGAIN ;
12 -->
13

Block # 25

( #6 EDITOR --- 5.10.82 )

1 : FNUMC 63 FCCOL @ - ;
2 : FIBL FCURAD FCURAD 1+ FNUMC CMOVE
3 BL FCURAD C! FRFRSH ;
4 : FDCH FCURAD DUP 1+ SWAP FNUMC CMOVE
5 BL 63 FCLIN @ FCHAD C! FRFRSH ;
6 : FILN FCLIN @ 1 - 14 DO
7 0 I FCHAD 0 I 1+ FCHAD 64 CMOVE
8 I 1+ .FLINE -1 +LOOP
9 0 FCLIN @ FCHAD 64 BL FILL FRFRSH ;
10 : FDLN 15 FCLIN @ DO
11 0 I 1+ FCHAD 0 I FCHAD 64 CMOVE
12 I .FLINE LOOP
13 0 15 FCHAD 64 BL FILL
14 15 .FLINE FREPOS ;
15 -->

A-2
( #7 EDITOR --- 5.10.82 )

1 : FSAVE R> DROP FCLEAR 6 ;
2 : FEXIT R> DROP FCLEAR 7 ;
3 : FPREV R> DROP FCLEAR 4 ;
4 : FNEXT R> DROP FCLEAR 5 ;
5
6 -->

( #8 EDITOR --- 5.10.82 )

1 FCMND FIBL 1 FCMND FUP 2 FCMND FDOWN 3 FCMND FRIGHT
2 4 FCMND FLEFT 8 FCMND FHOME 12 FCMND FILN 13 FCMND FDLN
3 14 FCMND FDCH 15 FCMND FIBL
4 17 FCMND FEXIT ( RED ) 18 FCMND FSAVE ( GRAY )
5 22 FCMND FPREV ( F-4 ) 23 FCMND FNEXT ( F-5 )
6
7 -->

( #1 DOSF --- 6.5.84 )

1 : HELP CR CR ." COMMANDS LIST"
2 CR ." SETDIR-set up a directory MAKE-create a file"
3 CR ." KILL-destroy a file PROTECT-write protect a file"
4 CR ." UNPROTECT-unprotect a file RENAME-rename a file"
5 CR ." WRITE-write onto a file READ-read a file"
6 CR ." WRITE-write onto a file RREAD-random read a file"
7 CR ." FILES-directory information BLOCKS-files information"
8 CR ." RUN-load a file NAME-name a file"
9 CR ." END-get out of system SHOW-edit a file"
10 CR ." FCOPY-copy a file" CR ;
11
12 -->

A-3
Block # 29

1  ( #2 DOSF --- 6.5.84 )
2  : <= - DUP 0= SWAP 0< OR ;
3  : >= SWAP <= ;
4  : THEN [COMPILE] ENDF ; IMMEDIATE
5  : <> DUP ROT DUP ROT < ROT ROT SWAP > OR ;
6  : NOT DUP 0= IF DROP 1 ELSE 1 - THEN ;
7  0 VARIABLE NUMB
8  : #IN 50 BUFFER NUMB ! NUMB @ 1 + 5 EXPECT 0.
9  NUMB @ (NUMBER) DROP DROP ;
10  : Y/N CR." ENTER 1-IF YES 0-IF NO" KEY DUP 48 = IF DROP 0 ELSE
11  49 = IF 1 ELSE CR." ERROR TRY AGAIN " THEN THEN ;
12  : -! DUP @ ROT - SWAP ! ;

Block # 30

1  ( #3 DOSF --- 6.5.84 )
2  : CARRAY <BUILDS ALLOT DOES> + ;
3  99 CONSTANT DSKSZ
4  99 VARIABLE DIRBLK
5  DSKSZ 8 / 1 + CONSTANT MAPSZ
6  MAPSZ CARRAY BITMAP
7  MAPSZ CARRAY QMAP
8  1024 CARRAY DBUFF 0 DBUFF VARIABLE DADRS
9  15 CARRAY FNAME 15 CARRAY ONAME
10  0 VARIABLE NBLK 0 VARIABLE BM1 0 VARIABLE BM2
11  0 VARIABLE BM3 0 VARIABLE N 0 VARIABLE N1
12  15 CARRAY TMP 0 VARIABLE NEW
13  0 VARIABLE DCOUNT 0 VARIABLE FMAP 200 ALLOT
14  -->

Block # 31

1  ( #4 DOSF --- 6.5.84 )
2  : T# 0 <# # S #> TYPE ;
3  : C+! DUP @ ROT + SWAP ! ;
4  : C-! DUP @ ROT - SWAP ! ;
5  : 2T DUP 0= IF DROP 1 ELSE 1 SWAP 0 DO 2 * LOOP THEN ;
6  : DZONK 0 BITMAP MAPSZ 0 FILL ;
7  : DGT DUP 8 / DUP ROT ROT 8 * - SWAP DUP ROT 2T ;
8  : DON DGT SWAP BITMAP @ OR SWAP BITMAP ! ;
9  : DOFF DGT BM2 ! BITMAP @ -1 BM2 @ - AND SWAP BITMAP ! ;
10  : DON? 0 BM2 ! DUP DGT SWAP BITMAP @ AND SWAP DROP 0=
11  IF DROP ELSE BM2 ! THEN ;
12  0 VARIABLE 1SB
13  : DNXT 0 BM1 ! DSKSZ 5 DO I DON? BM2 @ 0= I 1SB @ >= AND
14  IF I BM1 ! LEAVE THEN LOOP BM2 @ 0 <> BM1 @ 0= OR
15  IF -1 BM2 ! ." *** Thread full ***" THEN ;
16  -->

A-4
1 Block # 32

1 ( #5 DOSF --- 6.5.84 )
2 : DREAD DADRS @ DIRBLK @ 1 R/W ;
3 : DWRITE DADRS @ DIRBLK @ 0 R/W ;
4 : DRD DADRS @ 0 BITMAP MAPSZ CMOVE ;
5 : DWR 0 BITMAP DADRS @ MAPSZ CMOVE ;
6 : DMOUNT DZONK DREAD DRD ;
7 : DSAVE DWR DWRITE ;
8 : REMOVE DSAVE FLUSH CR ." REMOVE DISKETTE " ;
9 : DINIT DMOUNT DADRS @ 1024 0 FILL DZONK ;
10 : LOOKUP -1 BM1 ! DADRS @ 128 + N !
11 14 0 DO N @ @ 0 <> IF N @ @ 0 FNAME @ =
12 14 IF N @ 1 + @ 1 FNAME @ = IF N @ 2 + @ 2 FNAME @ =
13 IF N @ 3 + @ 3 FNAME @ = IF I BM1 ! LEAVE
14 THEN THEN THEN THEN THEN 64 N +!
15 LOOP BM1 @ ;
16 -->

2 Block # 33

1 ( #6 DOSF --- 6.5.84 )
2 : DPUTNUM LOOKUP -1 <>
3 IF CR ." *** File already exists ***"
4 ELSE DADRS @ 128 + N !
5 13 0 DO N @ @ 0 =
6 IF 0 FNAME N @ 15 CMOVE LEAVE 0 N !
7 ELSE 64 N +! THEN
8 LOOP N @ 0 <>
9 IF CR ." *** File name space is full ***"
10 THEN THEN ;
11 : FON? DUP DGT SWAP QMAP @
12 AND SWAP DROP 0= IF DROP 0 BM2 !
13 ELSE BM2 ! THEN ;
14 -->
15
16 Block # 34

1 ( #7 DOSF --- 6.5.84 )
2 : NAME 0 FNAME 15 BL FILL SPACE CR ." FILENAME:" SPACE
3 0 FNAME 15 EXPECT ;
4 : DGTMAP LOOKUP -1 <>
5 IF DADRS @ 128 + BM1 @ 64 * + 20 +
6 DUP N ! 0 QMAP MAPSZ CMOVE
7 ELSE CR ." *** File not found ***" THEN ;
8 : DADD
9 DGTMAP DNXT BM2 @ -1 <>
10 IF BM1 @ DUP DON DGT SWAP QMAP @ OR SWAP QMAP !
11 0 QMAP N @ MAPSZ CMOVE BM1 @ ELSE 0 THEN ;
12 : SETF 2 DCOUNT ! 0 BM3 ! 100 0 DO 0 FMAP I 2 * +1 LOOP
13 DGTMAP BM1 @ -1 <> IF DSKSZ 0 DO I FON? BM2 @
14 0 <> IF 1 BM3 +! I FMAP DCOUNT @ + ! 2 DCOUNT +! THEN LOOP
15 THEN ;
16 -->

A-5
```
Block # 35

1 ( #8 DOSF --- 6.5.84 )
2   : 1STBK DGTMAP 98 0 DO I FON?
3       BM2 @ DUP DUP 0 <> IF ISB ! LEAVE ELSE DROP THEN LOOP ;
4   : MAKE 0 ISB ! NAME DPUTNM 1STBK ;
5   : RENM ADDR @ 128 + BM1 @ 64 * + N ! ;
6   : KILL NAME LOOKUP -1 <> IF DGTMAP DSKSZ 0
7       DO I FON? BM2 @ 0 <> IF BM2 @ DOFF THEN
8       LOOP RENM N @ 64 0 FILL THEN ;
9
10
11
12
13
14
15
16

Block # 36

1 ( #9 DOSF --- 6.5.84 )
2   : FSIZE
3       DGTMAP BM1 @ -1 <>
4       IF 0 BM1 ! DSKSZ 0
5       DO I FON? BM2 @ 0 >
6       IF 1 BM1 +!
7       THEN
8       LOOP
9       BM1 @ SPACE T# SPACE ." BLOCKS"
10       THEN ;
11   : PROT?
12       LOOKUP -1 <> IF RENM N @ 62 + @ 170 = IF 1 ELSE 0 THEN
13       ELSE CR ." *** File not found ***" THEN ;
14
15
16

Block # 37

1 ( #10 DOSF --- 6.5.84 )
2   : WRITE
3       N1 ! PROT? 0= IF DADD DUP 0 >
4       IF N1 @ SWAP 0 R/W 0 ELSE DROP 1 THEN
5       ELSE 1 ." *** Try to write on protect file ***" THEN ;
6   : RFIND
7       DGTMAP DUP N ! BM1 ! DSKSZ 0 DO I FON? BM2 @ 0 >
8       IF 1 BM1 C-! BM1 @ 0 > NOT IF I N ! LEAVE THEN THEN LOOP ;
9   : RREAD
10      RFIND BM1 @ 0 > IF 0 ELSE N @ BLOCK THEN ;
11   : RWRITE
12      RFIND N1 ! BM1 @ 0 > IF 0 ELSE N @ PROT? 1 =
13      IF DROP 0 ELSE N ! N1 @ N @ 0 R/W THEN THEN ;
14
15
16
```

A-6
( #11 DOSF --- 6.5.84 )

1: PROTECT
2   NAME LOOKUP -1 <> IF RENM 170 N @ 62 + !
3   ELSE CR ." *** File not found ***" THEN ;
4: UNPROTECT
5   NAME LOOKUP -1 <> IF RENM 0 N @ 62 + !
6   ELSE CR ." *** File not found ***" THEN ;
7: READ
8   DGTMAP BM1 @ -1 <> IF
9   BEGIN NBLK @ DSKSZ > IF 0 NBLK ! 0
10   ELSE 1 NBLK +! NBLK @ FON? BM2 @ 0 <>
11   IF BM2 @ BLOCK 1 ELSE 0 THEN
12   THEN UNTIL THEN ;

( #12 DOSF --- 6.5.84 )

1: SHOW
2   NEW ! SETF NEW @ 0 <= IF 2 DCOUNT ! ELSE
3   NEW @ 2 * DCOUNT ! THEN
4   999 0 DO FMAP DCOUNT @ + ! DUP 0 <> IF
5   FEDITOR DUP DUP 5 = IF 2 DCOUNT +! DROP DROP
6   ELSE 4 - IF 2 DCOUNT -! DROP ELSE DUP 6 = IF
7   PROT? 0= IF UPDATE ELSE ." File is protected"
8   LEAVE THEN DROP ELSE 7 = IF PROT? 0= IF
9   UPDATE ELSE ." File is protected" THEN LEAVE
10   THEN THEN THEN ELSE
11   ." Do you want to add a new block ?" Y/N
12   1 = IF DADD DROP SETF 2 BM3 @ * DCOUNT !
13   ELSE LEAVE FCLEAR ." End of file " THEN THEN LOOP ;

( #13 DOSF --- 6.5.84 )

1: FREE 0 BM1 ! DSKSZ 0
2   DO I DON? BM2 @ 0 > IF 1 BM1 +! THEN LOOP
3   CR DSKSZ BM1 @ - T# SPACE ." FREE DISK BLOCKS"
4   SPACE ." OUT OF" SPACE DSKSZ T# CR ;
5: RUN NAME LOOKUP -1 <> IF CR DGTMAP DSKSZ 0 DO I FON?
6   BM2 @ 0 <> IF I LOAD THEN LOOP
7   ELSE CR ." *** File not found ***" THEN CR ;
8: FILES DADRS @ 128 + N1 ! CR ." DISK DIRECTORY"
9   CR DADRS @ 64 + 64 TYPE
10   0 BM3 ! CR 13 0 DO N1 @ @ 0 <> IF N1 @ 0 FNAME
11   15 CMOVE PROT? 1 = IF ." * " ELSE SPACE SPACE
12   THEN 0 FNAME 15 TYPE FSIZE CR BM3 @ 0= IF 1 BM3 !
13   ELSE 0 BM3 ! THEN THEN 64 N1 +! LOOP CR
14   FREE ." (*) - write protected file)" CR ;

-->
Block # 41

1 ( #14 DOSF --- 6.5.84 )
2 : LOCK BEGIN ." Block# to be locked out (9999 if none):"
3 #IN DUP 9999 = IF DROP 1 ELSE DON 0 THEN CR UNTIL ;
4 : COPY O DO
5 0 FNAME 0 TMP 15 CMOVE
6 0 ONAME 0 FNAME 15 CMOVE
7 READ DUP 0 = IF DROP LEAVE ELSE
8 0 TMP 0 FNAME 15 CMOVE WRITE 1 = IF LEAVE THEN THEN
9 LOOP CR 0 ONAME 15 TYPE SPACE ." copied to "
10 0 TMP 15 TYPE CR ;
11 : FCOPY 0 NBLK ! CR ." Copy from " NAME LOOKUP -1 =
12 IF CR ." *** File not found ***" ELSE FSIZE BM1 @
13 N1 I O FNAME 0 ONAME 15 CMOVE ." to " NAME LOOKUP
14 -1 = IF CR ." New file" DPUTNM N1 @ COPY ELSE CR
15 ." Copy over old file?" Y/N IF N1 @ COPY THEN THEN THEN
16 -->

Block # 42

1 ( #15 DOSF --- 6.5.84 )
2 : SETDIR CR
3 ." New Directory" CR
4 ." Build a new directory ( destroy all files )" Y/N
5 IF DINIT CR ." WRITE PROTECT BELOW BLOCK#"
6 SPACE #IN 0 DO I DON LOOP
7 CR LOCK 0 DNAME 64 BL FILL
8 DIRBLK @ DON DWR DADRS @ 64 + N !
9 ." Diskette description: "
10 0 DNAME 64 EXPECT
11 0 DNAME N @ 64 CMOVE 0 DNAME 64 BL FILL
12 DSAVE THEN CR ;
13 -->
14
15
16

Block # 43

1 ( #16 DOSF --- 6.5.84 )
2 : RENAME SPACE BM1 @ -2 <>
3 IF ." OLD" NAME LOOKUP -1 =
4 IF SPACE ." *** File not found ***"
5 ELSE RENM SPACE ." NEW"
6 NAME O FNAME N @ 15 CMOVE CR
7 THEN
8 ELSE LOOKUP -1 <>
9 IF @ O FNAME SWAP 15 CMOVE
10 THEN THEN ;
11 -->
A-9
BIBLIOGRAPHIES


