READING MTS FS TAPES AND MTS FINAL BACKUP TAPES WITHOUT MTS

Tapes. Tapes. Tapes are my LIFE!!

Michael Betker (The Cannon Fodder) *9-June-92*

Abstract

With MTS being shut off and there still being several thousand FS tapes hanging around in the machine room, someone thought that it would be a good idea to write a utility to read them when MTS was gone. Peter Van Epp was given the chore of figuring out how it should be done. As he says in the paper "SFU's migration from MTS to UNIX", he did all of the fun and interesting stuff and left the rest as an exercise for the student, that student being me. This paper is an attempt to explain why, and how everything got done the way it did.

Summary

With the demise of MTS it was necessary to make sure that there was a way to still deal with all of the FS tapes that users had on the system. As previously stated, this project was started by Peter Van Epp, who figured out most of the technical parts of the process needed to provide this service. Peter then passed the project off to Bill Baines who had the foresight and wisdom to hire someone else to do the actual work. When this project began, it originally was only supposed to deal with MTS FS tapes; however, as things progressed it was then decided that we also needed a way to backup all of the files that still existed on MTS just before it was shut down. This extended the amount of work necessary to be done for this tape project. How each of these types of tapes were dealt with will be explained separately below.

This project was done on the VAX because it had both of the necessary tape drives required to read both formats of tapes that had to be read: the 3420 FS tapes and the 3480 final backup tapes. Most of the MTS to UNIX migration projects were done with as much 'off the shelf' software as possible; however, it was determined that, for these two tape systems to work, most of the software required would have to be developed in house. It was also decided that the best language to use would be C, as it has the capability to do low level accesses of the tape drives which might be required.

SYSTEM DESIGN IDEAS.

1. How to arrange the data.

1.1 FS data tree structure

When we first sat down to start this project, there were several key issues that we thought we had to resolve:

1) What do you do with the MTS line numbers?
2) How do you decide whether to translate the file to ASCII?
3) How does UNIX read any of this after you write it to a tape format that UNIX likes?
4) How many blocks do you have to read to get a file from the tape?
5) What is the best way to separate the index and data files?
6) How much owner validation is necessary?

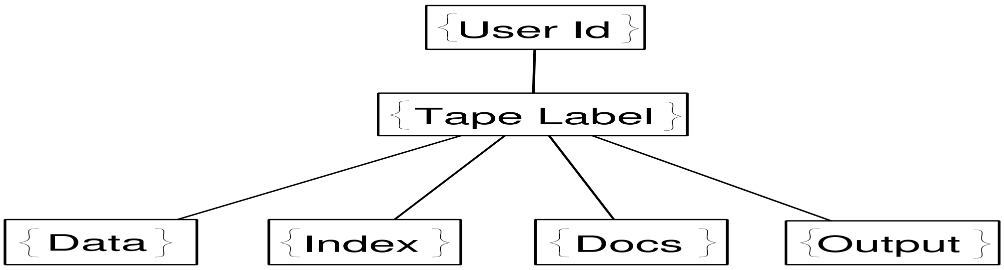
The answer to the first three points are explained in the paper "SFU's migration from MTS to UNIX". I won't reiterate Peter Van Epp's answers here, but I may expand on them if necessary.

The answer to point four is obviously, as few as possible, but there unfortunately must be a minimum. The answer turned out to be that you need the VOL1 label for the tape to be verified and the start of the tape to be guaranteed. The next block needed but not necessary is the HDR2 block of the empty tape info file. This block is used to ensure that the MTS user id passed in, is in fact the owner of the tape being worked on. If the owner is not verified by the first file on the tape, the HDR2 block of the first file is used to verify the owner of the tape. If the owner is not verified by this point in the process, the process is stopped. The next block of any importance is the HEADER block for the actual file. If this block is not read successfully the file will be skipped. For each data block to be read, a set of counters are kept. If the block is read successfully, a good block counter is incremented. If there is a checksum error with the data block, the block is written out and a warning counter is incremented and a warning is written to the log file. If a block can't be read, a bad block counter is incremented, and a message is written to the log file.

Point five became a concern for us when we came up with the idea to separate the line numbers into a different file as the solution to point one. It was decided that we would create a directory with the same name as the MTS user id for each person that we did a restore for. This directory would be their *root* directory. For each FS tape that was restored, there would be a sub-directory that corresponded to the volume label of the tape restored. Each sub-directory would have four sub-directories. They would be:

1) DATA - to store the data section of the files.
2) INDEX - to store the index section of the files.
3) DOCS - to store the DSAVE information that may exist for any file.
4) OUTPUT - an empty directory needed on the UNIX side for the combine program to place the final results of the recovery.

An example of the structure is shown below. As each tape was being processed, a log file was created and it was decided that the log file would be copied into the sub- directory corresponding to the label of the tape.



The answer to point six is that we tried to make sure that a person who asks for an MTS FS tape to be restored actually owns that tape. The way I did this was to pass the MTS id and tape label number to the recovery program and verify that they matched the data that was on the tape itself. As I explained in the answer for point four, I only tried to verify the owner up till the HDR2 block of the first file, and if I hadn't verified the owner by that point I quit the process and sent the request back for some other form of verification.

1.2 Final Backup data tree structure

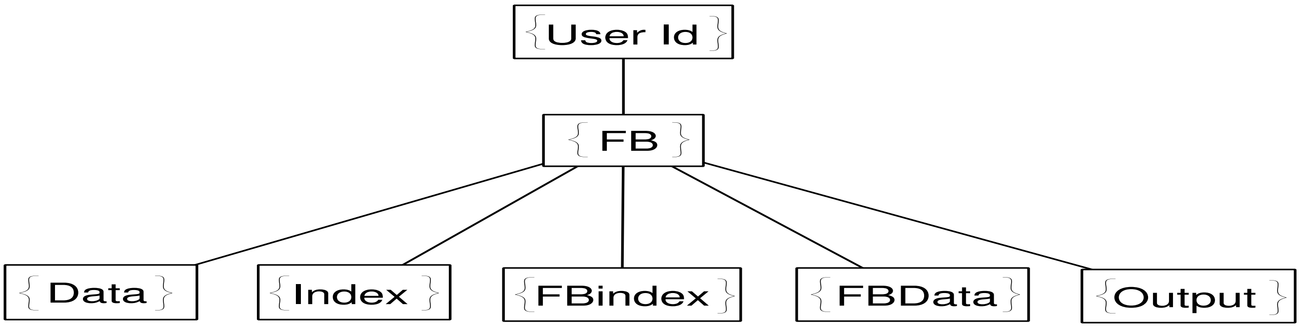
Given that the final backup procedures were started after the FS system was basically complete, most of the solutions used to resolve some of the issues that were of concern to FS tapes, were used to solve the similar final backup issues.

We used pretty much the same tree structure that was described for the FS tapes. There were two differences in the structures. The first was that instead of using the FS tape label as the first sub-directory, I used the value "FB" to signify that the files had come from a Final Backup restore. The other difference was that there were five sub-directories under the FB sub-directory. There was no need for the DOCS sub-directory, so it was dropped. However, two extra sub-directories were needed because of the data conversions that had to take place. These data conversions will be explained later on in the paper. The two extra sub-directories were:

1) FBINDEX - the pre data conversion index files

2) FBDATA - the pre data conversion data files

A log file was also created from the process of doing a final backup restore and that file was stored in the FB sub-directory. A picture showing the structure follows.



DEALING WITH FS TAPES

1. CREATING FS TAPES

1.1 Writing FS Tapes

FS Tapes use the MTS Tape Label format which is described in Appendix D on page 69 of chapter 19 of the MTS manuals. A detailed description of the FS Tape data section format is presented in page 112 of the same chapter. As this utility was available to the general public on MTS, not all of the tapes will be the same. This is because some of the users chose to use the DSAVE option when they were saving their files. This option allowed them to save up to 2000 bytes of documentation with their files.

I will quickly describe the data block order for a file on an FS tape, so that the explanations and descriptions of some of the issues and problems that I had, make sense. The format is as follows:

- HDR1 - contains file name and creation information
- HDR2 - contains tape type information
- TM - a tape mark
- HEADER - contains specific file information
- DATA - multiple blocks of actual file data
- TM - end of data tape mark
- EOF1 - duplicate of the HDR1 information
- EOF2 - duplicate of the HDR2 information
- TM - end of file tape mark

To begin a tape there is a VOL1 block which gives the tape volume number, then there is an empty file with no header or data blocks. To signify the end of the tape, another empty file is written to the tape.

2. RECOVERING FS TAPES

2.1 How to read something that isn't there to most tape drives.

When I started this project, Peter Van Epp had developed most of the tools necessary to read an FS Tape. It was up to me to use those tools correctly in order to actually salvage a person's files from their FS tapes. The tools that Peter had written would read a block of data from a buffer into a series of data sets that correspond to the various blocks that exist on an FS tape.

The first problem that had to be overcome was that when an FS tape is created, the first thing that gets put on the tape is an empty file. When a tape drive on a system that doesn't understand FS format tries to read the tape, it reads the VOL1 label, the HDR1 label, the HDR2 label, followed by two tape marks in a row. This generally signifies the end of the tape, and therefore, it quits.

This problem was overcome by using the OPEN function in C to open a channel directly to the tape drive, and then reading each physical block into a buffer and manually dealing with each block.

This solution of using the OPEN function and dealing with each block manually caused its own problem. The problem was that it was necessary to keep track of which block should be next and test to make sure that you got the block you were expecting. This also meant that I had to try and write the code such that if a major tape error occurred, it would be possible for the system to recover itself quite easily.

The second major problem that I encountered was in reading the data from the tape. I was able to use the OPEN function and read each physical tape block into a buffer, and then process the information in the buffer using the tools that Peter had written. However, when I tried to use this method to read the actual data blocks from the tape, I would get nothing back into the buffer. I believe the reason for this, may have been a string of data bits that were being interpreted as control characters or as a null, terminating the string of data prematurely. To get around this problem, I ended up using a VAX system routine that allowed me to access the tape drive at level below the C OPEN function. Using this method I was then able to get the data off of the tape.

A third problem was in dealing with the DSAVE option. Several of the first users that I did recoveries for had used the DSAVE option to save documentation with their files on tape. The problem was that the DSAVE option changed the length of the HEADER data block and made calculating the checksum value rather tricky. Peter had written a routine that would check the checksum of a block, if the checksum flag was present. With the length of the HEADER block not always being the same length, it was necessary to make sure that the block was extended out to be divisible by 4 and that any padding blocks were set to zero. Peter had done this in the actual data section, but not for the header, and it took me a while to track down this error as it was not a problem for all DSAVE files, just some of them.

While not really a problem, it was necessary to make sure that the MTS file names on the tape were translated to remove all of the characters that are valid on MTS but not valid on the VAX or UNIX. It was also important to devise a way so that if there were multiple files with the same name on the tape, that they would have unique names when they got to the UNIX side of things. The easiest way to do this was by reading all of the data in the header block and using it to my advantage. To guarantee uniqueness, I used the file's position number on the tape as the first part of the new name. This was followed by an underscore, then the MTS file name, another underscore, and then version number of the file according to the tape. I then ran this new name through a filter process to remove all of the MTS characters that aren't allowed and replace them with underscores. A list of illegal characters, and the format of a file name follow:

- The characters # % . : ! > < / | became _

- (TAPE POSS NO)_(MTS FILE NAME)_(TAPE FILE VERSION NUMBER)

DEALING WITH FINAL BACKUP TAPES

1. CREATING THE FINAL BACKUP TAPES.

1.1 Creation of a monster

The final backup of MTS to tape was done by a small program written by Ray Davison. The final backup of MTS resulted in 108 3480 tapes being created. The tapes were written using standard ANSI tape format. Each file that was backed up ended up on the tape in two files. The first file contains the data from the file that was dumped out in EBCDIC format with no carriage returns. This first file, if it didn't fill a full block, was padded at the end with ASCII blanks. The second file contains the line numbers from the file, and the length of each line. Each line number, line length sequence are on a separate line in the index file. At the end of the backup procedure there was a log file that for each file that was backed up, contained its owner, its name, its file positions on the tape, and the label of the tape it was written to. As each file was written to tape, Ray's program made sure that both of the data file and index file were successfully written to the tape before preceding to the next file. If both the data and index files didn't fit on the tape that was being used. The tape was backed up to the start of the data file and an end of tape mark was written to the tape. The files were then written as the first files on the next tape to be used.

2. RECOVERING DATA FROM THE FINAL BACKUP TAPES.

2.1 Putting the monster in a cage

After the backup tapes had been created, Ray Davison and Michael Hayward did some testing on the VAX to see how easy it was to recover the files from the tape. They discovered that the VAX copy command would read the files from the tapes. The only catch was that with all of the special characters that could be used in a file name, it was necessary to specify the file name in quotes in order for the copy command to work. The problem with this is that it eliminated the use of wildcard selections in the copy command. However, this didn't seem like too big of a deal as each file name had to be translated to make sure that it would be acceptable not only on the VAX but also on UNIX. So I went ahead and wrote a command file with several options for file recovery.

- The first option that the command file had was the ability to see what files had been backed up for a given user. This was just a matter of reading the log file, searching for the records with MTS user id belonging to the person, and printing the information to the screen.

- The second option was the ability to recover a specific file for a user. This was done by inquiring for the person's MTS id and the name of the file that they want recovered. The file name was then converted so that it would be acceptable on both the VAX and on UNIX. Then the tape was mounted and the two files for the file specified were copied from the tape.

- The third option to recover all of the files for a given MTS id is pretty much the same as the option to recover a single file. The difference is that instead of asking which file is to be converted, the log file was read to generate a list of all the files, and then a loop was entered to copy the files, one at a time from the tape.

This seemed like a pretty good solution when I got it all done, but after doing some testing, we discovered that the process was quite slow if the file was near the end of the tape. Bill and I tried to figure out how we could do this more efficiently.

The first thing we looked at was a VAX command line called "SET MAGTAPE" which has the ability to skip a certain number of files on the tape. Since we knew the file's position numbers, this command seemed like the right way to go; however, we then discovered that this command only works if the tape is mounted in a "FOREIGN" state. This was a problem as the copy command would only work if the tape was mounted in a normal state (i.e. not "FOREIGN").

The next idea was to write a small C program that would do the same thing, (i.e. it would skip a certain number of files). This wasn't overly hard to write after having written the whole FS Tape recovery program set. When I first tested this new procedure for getting to the correct position on the tape, all seemed much better. However, after some further testing I discovered that the program would skip the right number of files and leave the tape at the correct position, but when I then did the copy command, the system would rewind the tape to the beginning and start the search for the file in question, from the start of the tape. This wasn't very useful, so it was decided to abandon this line of thought and write the whole process from scratch.

The resulting process that I wrote was pretty much the same thing as the FS tape recovery routine. However, there were one or two small catches as should be expected. The first is that a user may have several FS tapes, but we were doing FS recoveries on a tape by tape scenario so there was no problem. However, with the backup tapes that Ray wrote, it was quite possible that a person's files may span from one tape to another. In fact it was quite possible that a person's files might actually cover several tapes, if they had enough data still left on MTS when the system was shut down. This multiple tape to an id problem, meant that the program used to recover all of a persons files would have to be smart enough to be able to read from several tapes. The second catch that had to be overcome, was that when I recovered FS tapes, I split the file into two parts myself. When Ray wrote the backup tapes, he split each file into two parts: a data part and an index part. The problem was that the format for the index parts in the two processes weren't the same, nor were the data parts. This would mean either writing a second UNIX program to recombine text files for Ray's formats, or changing one of the formats to be the same as the other. We decided that we would do a change of format of one of the index styles, while we had the files on the VAX. The reason for this decision was so that the program to combine files whether they be FS files or final backup files would be the same for the users[who?], and therefore, easier for them. It was decided that since I had already written the UNIX program to deal with the FS index format, that the final backup index files would be changed to FS format.

Having rewritten the whole process in C, how much did it change the way things were done? In reality, not much. I still used the same command script that I had written originally. Of course there needed to be some modifications.

The first option to display what files were backed up for a user, did not need to be changed in any way in fact. This is because it did not actually read any data from the tapes, it just read the file index file that Ray created while doing the final backup.

The second option to recover just one file changed only slightly. The change was that I would pass the position number of the file to be recovered into the program. The program would then skip that many files on the tape, and then read off the next two files from the tape. The first file would be the data and the second file would be the index.

The third option to recover all of a users files, needed to be changed quite a bit. Instead of generating a list of files to copy, the command file now figures out what tapes need to be read and what the positions of the first files belonging to the user are. If there are multiple tapes, it keeps the tape label and the position number in an array. Then for each entry set in the array the command script runs the program and passes in the position number, and a flag that says that all of the users files are to be read. The program then proceeds to read and recover sequential files until it finds either the end of the tape, or a file that belongs to a different user. It then either does the next tape or proceeds to the next part of the script.

The last part of the script for either of the file recovery processes is to do the data and index conversions necessary. The reason that it is necessary to do this conversion is because the format that Ray Davison used to stored the index information is different from the way created by the FS process, and used on the UNIX side. The format that Ray used was to output the line number and the length of the line, on separate lines in the index file. The difference from the FS format is that instead of the length of the line, I generated the offset into the file to get to that line. Therefore, it was just a matter of starting at zero and adding successive line lengths to get the offsets. One other thing had to be corrected. When the data section of a file was output to the tape, the MTS I/O routines would pad the end of the file with blanks if the data didn't fill a whole block. These blanks had to be removed. This was done by figuring out the total number of bytes of actual data and then reading that many bytes from the FBDATA file and writing them to a DATA file of the same name.

As in the FS tape recovery, it was necessary to translate the names of the data files. When I first wrote the command script using the copy command, I would replace any invalid character with a digit and underscore sequence. The digit used corresponded to the number of characters that had been replaced. With the new process of recovering files, it was necessary to translate the file names another way. The solution was found in the HDR blocks on the tapes, as I was able to do the same thing as I had done to translate file names in the FS process.

FS DATA AND UNIX

1. GETTING FS DATA TO UNIX.

1.1 Moving the data to UNIX

Now that I had the data from the FS tapes and the final MTS backup tapes in the same format and style on the VAX, I had to find a way to get them to UNIX. As Peter stated in the paper "SFU's migration from MTS to UNIX", "[t]he end result of all this is a process that will...spit out a UNIX tar file..." Unfortunately the VAX doesn't support tar natively. The solution was to hunt on the internet for a utility that would do the trick. Bill found one from the Queen's University in Kingston, Ontario. This utility worked like a charm and did the job. We could now tar a user's data tree structure on the VAX and ftp the tar file to their UNIX id. We also found a utility that would compress the tar file so that it didn't take up too much of a users disk space on UNIX.

One neat thing that should be noted regarding the two tree structures for FS files and Final Backup files is how similar they are. This actually was done on purpose for the explicit reason that if we did several FS tapes and some Final Backup recoveries for a user, they would have a tree with their id as the root, and the main sub- directories would be the labels of the FS tapes they had recovered and "FB" for the final backup recoveries. The structure inside each main sub-directory would be pretty much the same with a DATA, INDEX, and OUTPUT directories in common. The FS tape structures would also have the DOCS sub-directory. The FBINDEX and FBDATA sub- directories for the FB sub-directory will have been deleted after the data conversion was done. We could then create one tar file containing all of the files that had been recovered for the user instead of having to give them several tar files.

2. USING FS DATA ON UNIX.

2.1 Now that I have my data how do I use it?

Now that the users' data was on UNIX in a tar file, there needed to be a program to combine the data file and the index file, and translate the EBCDIC data portion translated to ASCII if the file was just a text file. It would make no sense to do this to a sequential or binary file, as they just wouldn't work. To do this combining, I wrote a small program to which you pass the location of the root of the tar file extract, the tape volume number or "fb", and the name of the file to be combined. It is possible to specify several files to be combined at the same time. This program has been used on the SGIs, SUN machines and on NeXTs with no problems other than having to recompile the program for the various hardware platforms. The output for this program will be the users file from the MTS final backup tapes or from the users FS tapes.