Module 16: Some Other Tools in UNIX

We have emphasized throughout that the philosophy of Unix is to provide - (a) a large number of simple tools and (b) methods to combine these tools in flexible ways. This applies to file compression, archiving, and transfer tools as well. One can combine these tools in innovative ways to get efficient customized services and achieve enhanced productivity. Usually, the use of these tools is required when user or system files need to be moved enblock between different hosts.

In this module we shall discuss some tools that help in archiving and storing information efficiently. Efficient utilization of space requires that the information is stored in a compressed form. We discuss some of the facilities available in Unix for compression. Also, compressed files utilize available network bandwidth better for file transfers on the internet. Without compression, communicating postscript, graphics and images or other multimedia files would be very time consuming.

One of the interesting tools we have included here is a profiler. A profiler helps in profiling programs for their performance. In program development it is very important to determine which segments of program are using what resources. We discuss a few program performance optimization strategies as well.

We begin the module with a description of archiving tools available in the Unix environment.

16.1 Tar and Other Utilities

In the past, archives were maintained on magnetic tapes. The command tar appropriately stands for the tape archiving command for historical reasons. These tapes used to be stacked away in large archival rooms. The tar command carries over to archiving files from a file system onto a hard disk. The command has a provision for a default virtual tape drive as well.

As a user, we distribute our files within a directory structure. Now suppose we need to archive or copy an entire set of files within a directory subtree to another machine. In this situation the tar command comes in handy. It would also be very handy when one has to copy a large set of system files to another machine. Many of the newer applications are also installed using a copy from a set of archived files. The *tar* command has the following structure:

tar options target source

As an example suppose we wish to archive all the .c files under directory. /M/RAND and place these under directory. /T, we may give a command as follows:

bhatt@SE-0 [~/UPE] >>tar cvf ./T/cfiles.tar ./M/RAND/.c*

a ./M/RAND/dum.c 1K

a ./M/RAND/main.c 1K

a ./M/RAND/old_unif.c 1K

a ./M/RAND/templet.c 4K

a ./M/RAND/unif.c 1K

We may now change to the directory T and give a *ls* command to see what we have there.

bhatt@SE-0 [T] >>ls

ReadMe cfiles.tar

The options used in *tar* command have the following interpretations:

The option c suggests to create, v suggests to give a verbose description on what goes on, and f indicates that we wish a file to be created in a file system. An absence of f means it will be created in /etc/remt0 = or a tape. To see the table of contents within a tarred file use the t option as shown below:

bhatt@SE-0 [T] >>tar tvf cfiles.tar

tar: blocksize = 18

-rw-r--r-- 10400/14210 414 Nov 30 15:53 1999 ./M/RAND/dum.c

-rw-r--r-- 10400/14210 364 Nov 30 16:38 1999 ./M/RAND/main.c

-rw-r--r-- 10400/14210 396 Nov 30 16:29 1999 ./M/RAND/old_unif.c

-rw-r--r-- 10400/14210 3583 Nov 29 11:22 1999 ./M/RAND/templet.c

-rw-r--r-- 10400/14210 389 Oct 16 09:53 2000 ./M/RAND/unif.c

To extract the files from the tarred set *cfiles*, we may use the x option as follows:

tar xvf cfiles.tar

This creates a directory M under T (M was the parent directory of RAND) under which RAND and files stood in the first place.

In particular, *tar* has the following options:

c: create an archive.

r: append files at the rear end of an existing archive.

t: list table of contents.

x : extract individual contents

f : file to be created within a file system

f : write/read from standard output/input

o : change file ownership

v : verbose mode, give details of archive information

Note that it is dangerous to tar the destination directory (i.e. take the archive of the destination directory) where we propose to locate the .tar file as this results in a recursive call. It begins to fill the file system disk which is clearly an error.

The *tar* command is very useful in taking back-ups. It is also useful when people move - they can tar their files for porting between hosts.

16.2 Compression

The need to compress arises from efficiency consideration. It is more efficient to store compressed information as the storage utilization is much better. Also, during network transfer of information one can utilize the bandwidth better. With enormous redundancy in coding of information, files generally use more bits than the minimum required to encode that information. Let us consider text files. The text files are ASCII files and use a 8 bit character code. If, however, one were to use a different coding scheme one may need fewer than 8 bits to encode. For instance, on using a frequency based encoding scheme like Huffman encoding, we would arrive at an average code length of 5 bits per character. In other words, if we compress the information, then we need to send fewer bits over the network. For transmission one may use a bit stream of compressed bits.

As such *tar*, by itself, preserves the ASCII code and does not compress information. Unix provides a set of compression utilities which include a compress and a *uuencode* command. The command structure for the compress or uncompress command is as follows:

compress options filename

uncompress options filename

On executing the *compress* command we will get file with a .Z extension, i.e. with a file filename we get filename.Z file. Upon executing *uncompress* command with filename.Z as argument, we shall recover the original file filename.

The example below shows a use of compress (also uncompress) command which results in a .Z file.

bhatt@SE-0 [T] >>cp cfiles.tar test; compress test; ls

M ReadMe cfiles.tar test.Z

bhatt@SE-0 [T] >>uncompress test.Z; ls

M ReadMe cfiles.tar test

Another method of compression is to use the *uuencode* command. It is quite common to use a phrase like *uuencode* a file and then subsequently use *uudecode* to get the original file. Let us *uuencode* our test file. The example is shown below:

*bhatt@SE-0 [T] >>uuencode test test > test.uu ; ls; rm test ; *

ls ; uudecode test.uu ; rm test.uu; ls

ReadMe cfiles.tar test test.uu M

ReadMe cfiles.tar test.uu M

ReadMe cfiles.tar test

Note that in using the *uuencode* command we have repeated the input file name in the argument list. This is because the command uses the second argument (repeated file name) as the first line in the compressed file. This helps to regenerate the file with the original name on using *uudecode*. Stating it another way, the first argument gives the input file name but the second argument helps to establish the file name in the output. One of the most common usages of the *uuencode* and *uudecode* is to send binary files. Internet expects users to employ ASCII format. Thus, to send a binary file it is best to *uuencode* it at the source and then uudecode it at the destination.

The way to use *uuencode/uudecode* is as follows:

uuencode my_tar.tar my_tar.tar > my_tar.uu

There is another way to deal with internet-based exchanges. It is to use MIME (base 64) format. MIME as well as SMIME (secure MIME), are Internet Engineering Task Force defined formats. MIME is meant to communicate non-ASCII characters over the net as attachments to a mail. Being a non-ASCII file, it is ideally suited for transmission of post-script, graphics, images, audio or video files over the net. A software *uudeview* allows one to decode and view both MIME and *uuencoded* files.

A *uuencoded* file must end with end without which the file is considered to end improperly. A program called *uudeview* is very useful to decode *uuencoded* files as well as files in the base 64 format.

Zip and unzip: Various Unix flavors, as also MS environments, provide instructions to compress a file with the *zip* command. A compressed file may be later unzipped by using an *unzip* command. In GNU environment the corresponding commands are *gzip* (to compress) and *gunzip* (to uncompress). Below is a simple example which shows use of these commands:

```
bhatt@SE-0 [T] >>gzip test; ls; gunzip test.gz; ls;
M       ReadMe      cfiles.tar test.gz
M       ReadMe      cfiles.tar test
```

In MS environment one may use .ZIP or PKZIP to compress and PKUNZIP to decompress the files.

One of the best known compression schemes is the LZW compression scheme (the letters LZW stand for the initials of the two inventors and the one who refined the scheme).

It was primarily designed for graphics and image files. It is used in the .gif format. A discussion on this scheme is beyond the scope of this book.

Network file transfers: The most frequent mode of file transfers over the net is by using the file transfer protocol or FTP. To perform file-transfer from a host we use the following command.

```
ftp <host-name>
```

This command may be replaced by using an *open* command to establish a connection with the host for file transfer. One may first give the *ftp* command followed by *open* as shown below:

```
ftp
open <host-name>
```

The first *ftp* command allows the user to be in the file transfer mode. The *open* arranges to open a connection to establish a session with a remote host. Corresponding to open we may use *close* command to close a currently open connection or FTP session. Most FTP protocols would leave the user in the FTP mode when a session with a remote host is closed. In that case a user may choose to initiate another FTP session immediately. This is useful when a user wishes to connect to several machines during a session. One may use the *bye* command to exit the FTP mode. Usually, the *ftp ftp* connects a user to the local server.

With anonymous or guest logins, it is a good idea to input one's e-mail contact address as the password. A short prompt may be used sometimes to prompt the user. Below we show an example usage:

user anonymous e-mail-address

Binary files must be downloaded using the BINARY command. ASCII files too can be downloaded with binary mode enabled. FTP starts in ASCII by default. Most commonly used *ftp* commands are *get* and *put*. See the example usage of the *get* command.

get <rfile> <lfile>

This command gets the remote file named *rfile* and assigns it a local file name *lfile*. Within the FTP protocol, the *hash* command helps to see the progression of the *ftp* transfers. This is because of # displayed for every block transfer (uploaded or downloaded). A typical *get* command is shown below.

ftp> hash

ftp> binary

ftp> get someFileName

During multiple file downloads one may wish to unset interactivity (requiring a user to respond in y/n) by using the *prompt* command. It toggles on/off on use as shown in the example below:

ftp> prompt

ftp> mget filesFromAdirectory

The *mget* or *mput* commands offer a selection to determine which amongst the files need to be transferred. One may write shell scripts to use the *ftp* protocol command structure. This may be so written to avoid prompts for *y/n* which normally shows up for each file transfer under the *mget* or *mput* commands.

Unlike *tar*, most *ftp* protocols do not support downloading files recursively from the subdirectories. However, this can be achieved in two steps. As a first step one may use the tar command to make an archive. Next, one can use the *ftp* command to effect the file transfer. Thus all the files under a directory can be transferred. In the example below, we additionally use compression on the tarred files.

1. Make a tar file: create xxx.tar file
2. Compress: generate xxx.tar.z file
3. Issue the ftp command: ftp

Below we have an example of such a usage:

1. **Step 1:** *$ tar -cf graphics.tar /pub/graphics*
   This step takes all files in /pub/graphics and its subdirectories and creates a tar file named graphics.tar.
2. **Step 2:** $ compress graphics.tar
   This step will create graphics.tar.z file.
3. **Step 3:** uncompress graphics.tar.z to get graphics.tar
4. **Step 4:** tar gf graphics.tar will give file gf .

16.3 Image File Formats for Internet Applications

With the emergence of the multimedia applications, it became imperative to offer services to handle a variety of file formats. In particular, we need to handle image files to support both still and dynamic images. Below we give some well known mutimedia file formats which may be used with images.

File extension	Usage
.AVI	Audio visual interleave
.DL	Animated picture files (with .PICT .MAC extensions).
.PCX and .BMP	May be IBM PC image files
.WPG	A word perfect file.
.RAW	May be 24 bit RGB picture file.

JPEG is a compression standard specification developed by the Joint Photographic Engineering Group. There are utilities that would permit viewing the images in the .jpg files. Most JPEG compressions are lossy compressions. Usage of .jpg files is very popular on internet because it achieves a very high compression. In spite of being lossy, jpeg compression invariably works quite well because it takes into account some weaknesses in human vision. In fact, it works quite adequately with 24-bit image representations.

JPEG files are compressed using a quality factor from 1 to 100 with default being 55. When sending, or receiving, .jpg files, one should seek for quality factor of 55 and above to retain image quality at an acceptable level.

The basic *uuencoding* scheme breaks groups of three 8-bit characters into four 6-bit patterns and then adds ASCII code 32 (a space) to each 6-bit character which in turn

maps it on to the character which is finally transmitted. Sometimes spaces are transmitted as grave-accent (' ASCII 96). The *xxencoding* is newer and limits itself to 0-9, A-Z, a-z and +- only. In case a compressed file is *uuencoded, uudecode* may have to be followed by an *unzip* step.

File sizes up to 100{300K are not uncommon for .gif files. Often UseNet files are delimited to 64k. A typical 640*480 VGA image may require transmission in multiple parts. Typical .gif file in Unix environment begins as follows:

begin 640 image.gif

640 represents the access rights of file.

Steps for getting a .gif or .jpg files may be as follows:

1. **Step 1:** Get all the parts of the image file as part files.
2. **Step 2**: Strip mail headers for each part-file.
3. **Step 3:** Concatenate all the parts to make one .uue file.
4. **Step 4:** uudecode to get a .gif/.jpg or .zip file.
5. **Step 5:** If it is a .zip file unzip it.
6. **Step 6:** If it is a .jpg file either use jpg image viewer like cview or, alternatively, use jpg2gif utility to get .gif file.
7. **Step 7:** View image from .gif file.

On the internet several conversion utilities are available and can be downloaded.

16.4 Performance Analysis and Profiling

One of the major needs in program development environments is to analyse and improve the performance of programs. For our examples here we assume c programs.

One may use some obviously efficient steps to improve efficiency of code. In his book, *The Art Of Computer System Performance Analysis*, Raj Jain advocates the following:

- Optimize the common case. In other words, if the program has to choose amongst several operational paths, then that path which is taken most often must be made very efficient. Statements in this path must be chosen carefully and must be screened to be optimal.
- In case we need to test some conditions and choose amongst many alternative paths by using a sequence of if statements, then we should arrange these if statements such that the condition most likely to succeed is tested first, i.e. optimize the test sequence to hit the most likely path quickly.

- Within an if statement if there are a set of conditions that are ANDED, then choose the first condition which is most likely to fail. This provides the quickest exit strategy.
- If the data to be checked is in the form of arrays or tables, then put these in the order such that the most likely ones to be accessed are up front, i.e. these get checked first.
- Avoid file input and output as much as possible. Also, batch as much input, or output, as possible. For instance, suppose we need to process each line of data. In such a case, get the file first in memory to process it line-by-line rather than reading the line-by-line data from disk for each step of processing.
- In the loops make sure to pull out as much data as possible. In particular, values that do not change within the loop computations need not be within the loop.

Steps To Analyze Performance Of c Programs: The following steps will walk us through the basic steps:

1. Compile with p option, the profiler option
   cc -p a.c
   (Additionally, use -o option for linked routines.)
2. Now run the program a.out. (This step results in a mon.out file in the directory)
3. Next see the profile by using prof command as follows: prof a.out

(For more details the reader is advised to see the options in man pages for prof command.)

The profiler gives an estimate of the time spent in each of the functions. Clearly, the functions that take a large percentage of time and are often used are the candidates for optimization.

A sample program profile: First let us study the following program in c:

```
#include <stdio.h>
#include <ctype.h>
int a1;
    int a2;
add() /* adds two integers */
{
```

```
    int x;
    int i;
    for (i=1; i <=100000; i++)
x = a1 + a2;
return x;
}
main()
 {
   int k;
   a1 = 5;
   a2 = 2;
   for (k=1; k <=1000000; k++);;
   printf("The addition gives %d \n", add());
}
```

Now let us see the way it has been profiled.

```
bhatt@SE-0 [P] >>cc -p a.c
bhatt@SE-0 [P] >>a.out
The addition gives 7
bhatt@SE-0 [P] >>ls
ReadMe a.c a.out mon.out
bhatt@SE-0 [P] >>prof a.out
```

%Time	Seconds	Cumsecs	#Calls	msec/call	Name
77.8	0.07	0.07	1	70.	main
22.2	0.02	0.09	1	20.	add

```
bhatt@SE-0 [P] >>
```

The table above lists the percentage time spent in a certain function, time in seconds, and the number of calls made, average milliseconds on calls and the name of the function activated.

Another profiler is a *gprof* program. It additionally tries to find the number of iterated cycles in the program flow graph for function calls.

Text Processing (Improving Performance): Most often computer programs at tempt text processing. One of the common strategies is the use a loop in the following way:

1. Find the *strlen* (the length of the string).
2. Use a loop and do character-by-character scan to process data.

What most users do not realize is that the *strlen* itself determines the string length by traversing the string and comparing each of the characters with null.

Clearly, we can check in the loop if the character in the string array is null and process it if it is not. This can save a lot of processing time in a text processing program.

Sometimes we are required to copy strings and use *strcpy* to do the task. If the architecture supports memory block copy, i.e. an instruction like *memcpy* then this is a preferred option as it is a block transfer instruction whereas *strcpy* copies byte-by-byte and is therefore very slow.