Operating Systems Lab
Dept of CSE, LNMIIT, Jaipur
08 February 2021
Lab Session 04

Introduction

Many activities on the computers are repetitive. They need to be performed multiple times. For example, employee payrolls routines need to run each month. Their tax recordings are needed each year. Leave records may need frequent updates.

Shell scripts provide a way to perform these tasks. Each of this task is a sequence of actions with many similarities to computer programs. A program computes by working on the values. A script works on objects and resources located on the computers. Files and devices are examples of such objects. A script is a sequence of shell commands to action on these computer-based resources.

You are already aware how Linux shell runs a command. A command received by a shell is usually serviced by an external program while the shell waits. The external service can be a set of shell commands stored in a file instead of an executable code. When invoked these commands are run by a separate shell while the first shell waits their completion. The second (spawned) shell, in turn may run yet another shell script.

A simple tutorial introducing some commonly used Linux commands is https://linuxcommand.org/lc3_learning_the_shell.php A fuller set of Linux commands may be found at https://www.howtogeek.com/412055/37-important-linux-commands-you-should-know/ Only a few of us need to know all these commands.

Like a program, flow control constructs are available in Linux shells – there are many shells, but we will use one called BASH. You can learn about these flow control construct by browsing this page: https://linuxcommand.org/lc3_writing_shell_scripts.php or at https://learnxinyminutes.com/docs/bash/ and for fuller details https://devhints.io/bash

Students should read a few example scripts, and related (suggested) documents to improve their skills in writing BASH scripts. To start the journey, we have a simple exercise for this lab session.

Caution:

Shell and script interpreters were created for professionals in the era when computer resources were meagre. These were not done for the entertainment and pleasure purposes which is dominant use of computers today. Script interpreters are very strict about use of spaces, tabs and lines in some places. Be sure that you do respect these requirements. For example, often spaces around (and) are mandatory. On the other hand, when assigning value to an environment variable, there should not be any space around symbol = on either side!

An easy to read description of common Linux commands is here: https://www.computerhope.com/unix.htm

A note on program compilation

For the tasks in this lab, is suggested that you compile programs using the following command-style:

```
gcc -o progName progName.c
```

(Note that we are seeking to name the executable files matching the original name of the C program). If you need to install `gcc` compiler on your computer, then run the following commands:

```
sudo apt-get update
sudo apt-get install gcc
```

Paging Algorithms

You are given a set of C programs. The set simulates a few paging algorithms under different virtual memory scenarios.

Program nameSorter.c

The program defines a problem, and we can set the size of the problem over range 2000 to 9000. The problem size is specified by the first argument in the command line.

The second command line parameter is also essential There are some debugging options which we will ignore. The only useful value for the second parameter for our purpose is `listaccess`.

The given version of the program always end through statement `exit(0);` in function `main()`. This marks a successful end of the program.

Edit this program so that it terminates through statement `exit(1);` for all cases when the command line arguments are invalid or out of their expected range. The exit statement should be `exit(0);` only for the proper successful terminations of the program.

After you have changed the program, test it. The exit status of the last command is available in shell through symbol $? Try this command sequence.

```
gcc -o nameSorter nameSorter.c
./nameSorter 2000 listaccess
echo $?
```

Are you bothered by too much output? Find out how to redirect the program outputs. You should be able to explain the following commands once you have completed your survey.

```
./nameSorter 2000 access 2>/dev/null
./nameSorter 2000 listaccess >/dev/null
./nameSorter 2000 listaccess >myoutput
./nameSorter 2000 listaccess >>myoutput
```

Program pageRefGen.c

This program expects only one command line argument. This value represents the page-size. This value is the size of each page frame in the simulated computer memory. Each reference

to a virtual memory address is to a page. Each page is referred to by its page number. This program converts memory addresses to their page numbers. Only some specific values are correct values as the command line arguments for this program. Compile the program as `pageRefGen` and determine the meaning of the following command combinations:

```
./pageRefGen
./pageRefGen 128 <myouput
./nameSorter 2000 listaccess | ./pageRefGen
```

Programs Opt.c, LRU.c, NRU.c and FIFO.c

These programs implement 4 paging policies. Program NRU requires two command line parameters. But to keep the things simple, it has been modified to use a default value for the second argument. All other programs require only one parameter, that should be a number less than 129. This argument value is the number of pages in the simulated computer memory.

Let us explain the simulation in a little more detail. Size of computer memory is given by two parameters:

frameSize = size of each memory frame. This parameter is used by program pageRefGen

numberFrames = Number of page frames available in the memory. This is used by the policy programs.

Total memory size is frameSize*numberFrames

We run a problem (`nameSorter`) on a computer. Problem size can be set as a number from 2000 to 8000 names to be sorted.

An operating system providing a virtual memory management facility transfer pages between computer memory and the disk several times. Total number of transfers between the disk and memory are computed as values Reads from the disk and as writes to the disk by the programs given to you (These programs are neither carefully tested, nor do they model the situation perfectly.)

Once, you have the programs compiled, you can evaluate paging policy performances through the simulated performance results. To perform an evaluation, use the following command format (This was the theme of Lab 03 last week):

./nameSorter *problemSize* listaccess | ./pageRefGen *frameSize* | *./policy numberFrames*

Here are two examples:

```
./nameSorter 2000  listaccess | ./pageRefGen 128 | ./LRU 64
./nameSorter 5000  listaccess | ./pageRefGen 64 | ./Opt 64
```

The output, I get for the second case is:

```
Stats
Read accesses 94077
Write accesses 22090
Reads from the disk 6818
Writes to disk 1734
```

Command Grep

We are only interested in knowing the number of disk reads and writes. Study command `grep` to find how can you eliminate unneeded lines from the output.

Use pipe to let command `grep` select the useful lines.

Command grep is very useful and often needed command. You should study it in detail, if not immediately then over the next few weeks. And, even after this semester.

Learning about pattern specification. You may begin here to learn about grep: https://www.geeksforgeeks.org/grep-command-in-unixlinux/

Variable substitution

Type command:

```
echo $PATH
```

It prints a long list of paths to directories where shell looks for executables to run your commands. PATH is an important environmental variable in the shell. Other variables of interest include: HOME, TERM, USER, … Well, you can give command env to list the whole lot of BASH environment variables.

You can set your own variables. Remember there is no white space before and after =.

```
PrimeMinister=”Narendra Modi”
echo $PrimeMinister
```

Ok enough from me. Go and find about more about the shell variables. As you learn about variables, remember to find out about single quotes (‘ ‘) and double quotes (“ “). And then there are special shell variables. We have already used $? earlier. Find a list of special variables here: https://wiki.bash-hackers.org/syntax/shellvars

Command substitution

Now learn about the command substitution (back quotes ` `). The modern BASH uses `$(command)` instead of back-quotes.

Construct a command to get the useful lines of the memory simulator (described previously) into a shell variable. I call it `DiskIO` for easy reference.

The command below printed a line with two numbers in it:

```
echo $DiskIO
```

One number is count of disk reads and other is count of disk writes. I want to compute their sum as my total disk activity.

Substrings

One can use substring manipulation on a value to separate two numerical values in variable `DiskIO`. Set up variables `DiskReads` and `DiskWrites` by extracting substrings of `$DiskIO`.

Some information is here: https://devhints.io/bash But, it does not solve the problem completely.

To avoid the challenging issues, I have printed all number using 15 spaces. There are more advanced utilities like `awk` that can be used but that is well outside the scope of this subject.

Arithmetic Expansion

Bash supports limited arithmetic on values using double parentheses. https://tldp.org/LDP/abs/html/arithexp.html You should use it to compute total Disk-Memory transfers. And use command echo to print the total `DiskActivity` computed.

For you to learn later

History commands https://tldp.org/LDP/abs/html/histcommands.html

This seems to have a lot of info for what I want you to do next: https://tldp.org/LDP/abs/html/

Other topics that are worth definite attention are: command `find`; regular expressions for specifying patterns, vim editor. It is important to know about man pages and various sections. Find out what `man -k` does. What is command `apropos`? How does one read a man page from a specific section? By default, if man page in two different sections have the same name, one in the lower numbered section is displayed.

Donot forget to run command man on man on your computer terminal.

```
man man
```

These are your self-study tasks. For now, let us continue with the lab exercise.

Also there are some interesting hidden files like .profile, .bashrc that are worth knowing about.

Our Next Aim

What we want to develop is a shell script called `simulate.sh`.

The script will take four command line parameters: Problem-Size Page-Frame-Size Page-Frame-Count Policy. Students already know the valid values for these parameters for the available programs in our set. In addition, we will allow one of these parameters to be All. That is, in a simulation run one command line parameter can be specified as All. This will be interpreted as a request for multiple simulations using different values for the corresponding parameter.

Parameter marked All	Range of options to be simulated
Problem-Size	2000, 3000, 4000, 5000, 6000, 7000, 8000
Page-Frame-Size	16, 32, 64, 128, 256
Page-Frame-Count	16, 32, 64, 128, 256, 512
Policy	FIFO, NRU, LRU, Opt

The desired output from the script will report the single valued parameters and their settings in its first line.

The lines below the first will report in two column fashion the results of the simulation. One line for each level of the parameter chosen in response to keyword All. Obviously, the reported values in the line will be chosen parameter level and the estimated disk activities.

It is suggested that you complete this task as two separate scripts. Script 1 is to run simulator with single values for all 4 parameters. We describe this script in the next few paragraphs. Script 2 will be the requited script `Simulate.sh`.

`Simulate101.sh`

This simulator is simply a script that collects all the commands we have worked through in the earlier tasks of this lab exercise.

Create this script in a directory other than where you have located your compiled codes. Augment your PATH to access the programs.

In doing so take care to know what is mandated first line of each BASH script. Be sure to set the first line correctly. Also, the script file needs to have execute permit set. Use command, `chmod u+x` to add the needed access permission.

Look for information about the control flow features for BASH. Here are some suggested locations: https://tldp.org/LDP/abs/html/ and https://devhints.io/bash

Try to include cautionary messages in the script if a user invokes the script with incorrect parameters.

`Simulate.sh`

It should now be relatively easy task for you to complete the final script. This script will invoke the previously created script in a for loop.

Please do so and keep it ready for use when the virtual memory is discussed in the class.

Other Resources

https://www.computerhope.com/unix/ubash.htm

https://www.openvim.com/tutorial.html

https://www.tutorialspoint.com/vim/index.htm
