Student: Jade Cheng

ICS 412

Programming Project #1

Date: September 14, 2009

Submission Components

There are four files in this homework submission: `project#1.sh`, `mycp.c`, `Makefile`, and `project#1-report.pdf`. The bash script `project#1.sh` prompts the user to trace (`` `strace` ``) either `` `cp` `` or `` `mycp` `` and displays statistics for the system call `` `write` ``. The C file `mycp.c` copies a file with a specified buffer size. The `Makefile` builds `mycp.c` using `gcc`. Last but not least, this document is the `project#1-report.pdf`.

Question 1 cp and system calls

Question: The UNIX cp system program copies one file to another. This is done by reading chunks of bytes from the source into a buffer of some size, and then writing the content of the buffer in the destination. This is done until the entire source has been copied into the destination (at the last iteration the buffer may be only partially full since the size of the source in bytes may not be perfectly divisible by the buffer size). Using a system call tracing facility (i.e., strace on a Linux system, or ktrace on a Max OS X system), and a little bit of ingenuity, determine the size of the buffer used by the cp implementation on the machine you're using. Describe the procedure you used, explain why it does indeed allow you to determine the size of the buffer, and say which buffer size was discovered. Hint: this can be done by hand, but could be easier using a script to discover buffer size somewhat automatically (using Perl, Python, whatever scripting language you like).

Answer: I executed `` `project#1.sh cp` `` to trace `` `cp` `` 6 times with 6 files of increasing file sizes and to display the statistics for the system call `` `write` ``.

```
ubuntu@ubuntu:~/Desktop/412 Project 1$ ls
Makefile  mycp.c  project#1-report.pdf  project#1.sh

ubuntu@ubuntu:~/Desktop/412 Project 1$ ./project#1.sh --help
Usage: ./project#1.sh cp
   or: ./project#1.sh mycp bufsize
```

```
ubuntu@ubuntu:~/Desktop/412 Project 1$ ./project#1.sh cp
Running cp...
File size: 10 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n"..., 10)          = 10

File size: 100 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 1000 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 1000) = 1000

File size: 10000 bytes
  Occurrences of 'write':      2
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

File size: 100000 bytes
  Occurrences of 'write':      13
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

File size: 1000000 bytes
  Occurrences of 'write':      123
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

Done.

ubuntu@ubuntu:~/Desktop/412 Project 1$ ls
Makefile  mycp.c  project#1-report.pdf  project#1.sh
```

As we see, `write` was called more times as the file size increased. The maximum length of the buffer that could be written at a time was 8192 bytes. Any file this size or smaller required one `write` to completely copy.

Notice that I printed the first occurrence of a line that contained the `write` function call in the trace output file. After studying these output files, I came to the conclusion that this first occurrence correctly indicated both the maximum read and maximum write buffer sizes (in my distro of Linux). This was not the case for the `read` call, which was used several times before the first `write` to load what looked like supporting libraries.

Therefore, my answer for Question #1 is 8192 bytes.

Question 2 mycp.c

Question: Implement a program, mycp, that takes three command line arguments: an integer > 0 and two file names (i.e., paths). The program copies the content of the first file into the second file. If a file with the second name already exists, then this file is overwritten. This program should be written using the standard Posix API, which is supported by all Linux Systems. The program should gracefully exit if the first file cannot be opened for reading, or the second file cannot be opened for writing, or if any other error occurs. You can choose to use a high-level API (fopen, fread, fwrite, fclose) or a lower-level API (open, read, write, close). All those system calls are located in the man pages (section 3 for the former, section 2 for the latter). The first argument is used to specify the size of the buffer used for reading/writing.

Answer: I used the same script to test my `mycp` program. Run directly, the `mycp` program takes three arguments: buffer size (in bytes), source path, and destination path. Executed through the script, however, I simply passed the buffer size. This traced `mycp` 6 times with 6 files of increasing file sizes.

```
ubuntu@ubuntu:~/Desktop/412 Project 1$ ls
Makefile  mycp.c  project#1-report.pdf  project#1.sh

ubuntu@ubuntu:~/Desktop/412 Project 1$ make
gcc -Wall -DRELEASE -c mycp.c
gcc -Wall -o mycp mycp.o

ubuntu@ubuntu:~/Desktop/412 Project 1$ ./project#1.sh mycp 100
Running mycp with buffer size 100...
File size: 10 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n"..., 10)          = 10

File size: 100 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 1000 bytes
  Occurrences of 'write':      10
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 10000 bytes
  Occurrences of 'write':      100
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 100000 bytes
  Occurrences of 'write':      1000
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 1000000 bytes
```

```
  Occurrences of 'write':      10000
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

Done.

ubuntu@ubuntu:~/Desktop/412 Project 1$ ./project#1.sh mycp 10000
Running mycp with buffer size 10000...
File size: 10 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n"..., 10)          = 10

File size: 100 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 1000 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 1000) = 1000

File size: 10000 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 10000) = 10000

File size: 100000 bytes
  Occurrences of 'write':      10
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 10000) = 10000

File size: 1000000 bytes
  Occurrences of 'write':      100
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 10000) = 10000

Done.

ubuntu@ubuntu:~/Desktop/412 Project 1$ make clean
rm -f *.o mycp

ubuntu@ubuntu:~/Desktop/412 Project 1$ ls
Makefile  mycp.c  project#1-report.pdf  project#1.sh
```

As we can see, `write` was called more times as the file size increased. The maximum length of the buffer that could be written at a time was specified as a command line argument. Files that were this size or smaller required only one `write` to completely copy. Since the tested file sizes and tested buffer sizes were multiples, we have this relation:

$$\text{Occurrences of `write`} \times \text{Buffer Size} = \text{File Size}$$

My `mycp` program was able to write more than 8192 bytes at a time as shown in the second test. In the second test, I used a buffer size of 10000 bytes, and it took 100 times of `write` to copy a 1000000-byte file. Later I experimented with a buffer size of 1000000, and only one `write` appeared in the trace log. I came to the conclusion that `read` and `write` do not break apart large buffers and use more fundamental system calls.

This is in contrast to `fread` and `fwrite`. At first, I implemented my program using these functions, but the trace log showed that `fread` would not read more than 8192 bytes at a time and `fwrite` would not write more than 4096 bytes at a time. Instead the large buffers were broken into smaller pieces and passed to the more fundamental system calls.

Question 3 counting system calls

Question: Trace system calls both for cp and for mycp when copying the same file, with mycp using the same buffer size as cp. Are the number of system calls equal between the two programs? If not, can you tell which extra system calls are placed by cp or by mycp? Try to venture some reason why cp uses more or fewer system calls?

Answer: I executed `project#1.sh mycp 8192` and `project#1.sh cp` and compared the trace output files.

```
ubuntu@ubuntu:~/Desktop/412 Project 1$ ./project#1.sh mycp 8192
Running mycp with buffer size 8192...
File size: 10 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n"..., 10)          = 10

File size: 100 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 1000 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 1000) = 1000

File size: 10000 bytes
  Occurrences of 'write':      2
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

File size: 100000 bytes
  Occurrences of 'write':      13
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

File size: 1000000 bytes
  Occurrences of 'write':      123
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

Done.

ubuntu@ubuntu:~/Desktop/412 Project 1$ ./project#1.sh cp
Running cp...
File size: 10 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n"..., 10)          = 10

File size: 100 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 100) = 100

File size: 1000 bytes
  Occurrences of 'write':      1
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 1000) = 1000

File size: 10000 bytes
  Occurrences of 'write':      2
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

File size: 100000 bytes
  Occurrences of 'write':      13
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

File size: 1000000 bytes
  Occurrences of 'write':      123
  First occurrence of 'write': write(4, "012345678\n012345678\n012345678\n012"..., 8192) = 8192

Done.
```

As we can see, both `cp` and `mycp` wrote the same number of blocks of data when the `mycp` buffer size was set to 8192 bytes. For smaller buffer sizes, `mycp` called `write` more times. For larger buffer sizes, `mycp` called `write` fewer times. The same can be said to the `read` function but only during the main read-write loop.

The main differences between the trace output for `cp` and `mycp` occurred before the main read-write loop when `cp` opened, read, and closed several additional files. These files appeared to be supporting libraries that allow `cp` to handle advanced conditions, flags, and error cases. It's clear `cp` is a complicated program that does much more than just copy files.