Michael Smith

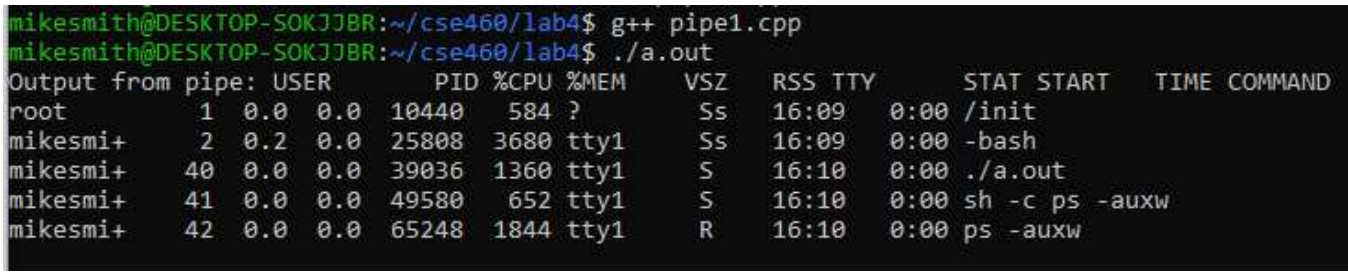
Lab 4

Total Points: 20 Points

1 February 2018

2. Process Pipes

Pipe1.cpp output



This program creates a "file" variable to read from the popen which creates a pipe, forks and then creates a shell. Then runs the command ps –auxw with the modifier –r which makes the pipe read the output from the shell. The program reads the shell output and stores them in a character array and prints it out on the terminal, which is what we see in the screen capture.

Pipe1a.cpp and output

```cpp
//pipe1a.cpp
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <iostream>

using namespace std;

int main(int arg, char** argc)
{
  FILE *fpi;                            //for reading a pipe
  if(arg != 3)
  {
        cout << "Use: " << argc[0] << " for the command argument" << endl;
        return 0;
  }

  char com[20];
  strcpy(com, argc[1]);
  strcat(com, " ");
  strcat(com, argc[2]);
  cout<< "Command: " << com << endl;

  char buffer[BUFSIZ+1];                //BUFSIZ defined in <stdio.h>

  int chars_read;
  memset ( buffer, 0,sizeof(buffer));   //clear buffer
  fpi = popen ( com , "r" );    //pipe to command "ps -auxw"
  if ( fpi != NULL ) {
    //read data from pipe into buffer
    chars_read = fread(buffer, sizeof(char), BUFSIZ, fpi );
    if ( chars_read > 0 )
     cout << "Output from pipe: " << buffer << endl;
    pclose ( fpi );                     //close the pipe
    return 0;
  }

  return 1;
}
~
```

```
mikesmith@DESKTOP-SOKJJBR:~/cse460/lab4$ ./a.out ls -l
Command: ls -l
Output from pipe: total 28
-rwxrwxrwx 1 mikesmith mikesmith 13680 Jan 31 16:20 a.out
-rw-rw-rw- 1 mikesmith mikesmith   896 Jan 31 16:20 pipe1.cpp
```

Pipe2.cpp output

```
mikesmith@DESKTOP-SOKJJBR:~/cse460/lab4$ ./a.out
0000000   A   r   n   o   d       s   a   i   d   ,       '   I   f
0000020   I       a   m       e   l   e   c   t   e   d   ,       .   .
0000040   '   ,       a   n   d       t   h   e       f   a   i   r   y
0000060       t   a   l   e       b   e   g   i   n   s  \n
0000075
```

The Program sends the text out of the pipe when it runs the od command with the –c modifier. Printing out the statement shown above.

3. Pipe3.cpp Output

```
mikesmith@DESKTOP-SOKJJBR:~/cse460/lab4$ ./a.out
Sent 5 bytes to pipe.
Read 5 from pipe: CSUSB
```

This code will display the number of bytes sent to the pipe and then displays the number of bytes and the data sent. When the pipe is created with the fd argument, it makes the fd handle the read end and the write end for the pipe. The write will display the first line in the output, while the read will output the second line of the output.

4. Parent and Child Processes

Had to modify the code in pipe4 to read out to the terminal to prompt user to send data, then store the data in the array instead of "123." Here is the modified lines

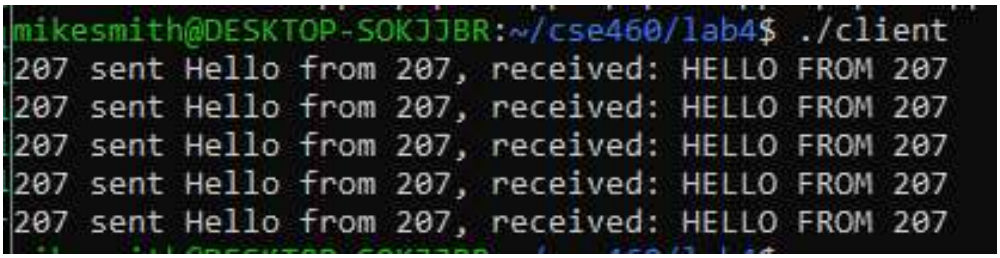
```
//pipe4.cpp
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main()
{
    int data_processed;
    int file_pipes[2];

        char some_data[256];
        printf("Insert the data being sent: ");
        gets(some_data);
```

Output

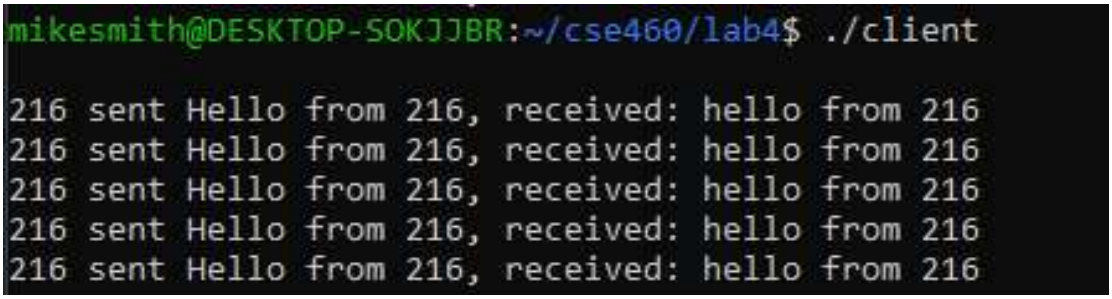
```
mikesmith@DESKTOP-SOKJJBR:~/cse460/lab4$ ./pipe4
Insert the data being sent: Hello
151 - wrote 5 bytes
mikesmith@DESKTOP-SOKJJBR:~/cse460/lab4$ 152 - read 5 bytes: Hello
```

5. Special Pipes

Code test before modification



Modifying the code in server.cpp to make the output into lower case instead of upper only requires the change of the line 49 code. Change the code from "*tmp_char_ptr = toupper(*tmp_char_ptr);" to "*tmp_char_ptr = tolower(*tmp_char_ptr);". Then the output becomes



6. XV6 Coding

Here is the code for the cp

```c
#include "types.h"
#include "stat.h"
#include "user.h"
#include "fcntl.h"
#define O_RDWR 0x002
#define O_CREATE 0x200
#define BUF_SIZE 256
int main(int argc, char *argv[])
{
        if (argc !=3)
        {
                printf(1, "please input the command as [cp source
                destination]\n");
                exit();
        }

        int fd0,fd1;

        char buf1[512];

        if((fd0 = open(argv[1],0)) < 0 ){
                printf(1, "cp: cannot open %s %d\n", argv[1] ,fd0);
                exit();

        }else{
                printf(1,"Read file opened\n" );

        if((fd1 = open(argv[2],O_CREATE | O_RDWR)) < 0 ){
                printf(1, "cp: cannot open %s %d\n", argv[2] ,fd1);
                exit();
        }

        int n;

        while((n = read(fd0, buf1, sizeof(buf1))) > 0){
                        write(fd1, buf1, n);
                }
        }

        exit();

        return 0;
}
```

Here is the output for the cp test

```
$ cp README myFile1 myFile2
$ ls
.              1 1 512
..             1 1 512
README         2 2 2290
cat            2 3 13344
echo           2 4 12412
forktest       2 5 8128
grep           2 6 15160
init           2 7 13000
kill           2 8 12464
ln             2 9 12360
ls             2 10 14584
mkdir          2 11 12484
rm             2 12 12464
sh             2 13 23104
stressfs       2 14 13140
usertests      2 15 56012
wc             2 16 13992
cp             2 17 12848
zombie         2 18 12192
console        3 19 0
myFile1        2 20 2290
myFile2        2 21 2290
$ cat myFile1
xv6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,
but is implemented for a modern x86-based multiprocessor using ANSI C.

ACKNOWLEDGMENTS

xv6 is inspired by John Lions's Commentary on UNIX 6th Edition (Peer
to Peer Communications; ISBN: 1-57398-013-7; 1st edition (June 14,
2000)). See also http://pdos.csail.mit.edu/6.828/2016/xv6.html, which
provides pointers to on-line resources for v6.

xv6 borrows code from the following sources:
    JOS (asm.h, elf.h, mmu.h, bootasm.S, ide.c, console.c, and others)
    Plan 9 (entryother.S, mp.h, mp.c, lapic.c)
    FreeBSD (ioapic.c)
    NetBSD (console.c)

The following people have made contributions: Russ Cox (context switching,
locking), Cliff Frey (MP), Xiao Yu (MP), Nickolai Zeldovich, and Austin
Clements.

We are also grateful for the bug reports and patches contributed by Silas
Boyd-Wickizer, Anton Burtsev, Cody Cutler, Mike CAT, Tej Chajed, Nelson Elhage,
Saar Ettinger, Alice Ferrazzi, Nathaniel Filardo, Peter Froehlich, Yakir Goaron,
Shivam Handa, Bryan Henry, Jim Huang, Alexander Kapshuk, Anders Kaseorg,
kehao95, Wolfgang Keller, Eddie Kohler, Austin Liew, Imbar Marinescu, Yandong
Mao, Hitoshi Mitake, Carmi Merimovich, Joel Nider, Greg Price, Ayan Shafqat,
Eldar Sehayek, Yongming Shen, Cam Tenny, Rafael Ubal, Warren Toomey, Stephen Tu,
Pablo Ventura, Xi Wang, Keiichi Watanabe, Nicolas Wolovick, Grant Wu, Jindong
Zhang, Icenowy Zheng, and Zou Chang Wei.

The code in the files that constitute xv6 is
Copyright 2006-2016 Frans Kaashoek, Robert Morris, and Russ Cox.

ERROR REPORTS

Please send errors and suggestions to Frans Kaashoek and Robert Morris
(kaashoek,rtm@mit.edu). The main purpose of xv6 is as a teaching
operating system for MIT's 6.828, so we are more interested in
simplifications and clarifications than new features.

BUILDING AND RUNNING XV6

To build xv6 on an x86 ELF machine (like Linux or FreeBSD), run
"make". On non-x86 or non-ELF machines (like OS X, even on x86), you
will need to install a cross-compiler gcc suite capable of producing
x86 ELF binaries. See http://pdos.csail.mit.edu/6.828/2016/tools.html.
```