George Suarez

CSE 460

Lab 4 – Study of Pipes and XV6

1. Pipes
   * Pipes are pseudo files that are used to communicate with other inter-processes which allows for data to flow from one process to another.
   * Syntax for the pipe command is
     + **$** command 1 | command 2 | command 3
2. Process Pipes
   * What do you see when you execute "pipe1"? Why?

* *pipe1.cpp*

//pipe1.cpp

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <iostream>

using namespace std;

int main()

{

FILE \*fpi; //for reading a pipe

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

int chars\_read;

memset(buffer, 0, sizeof(buffer)); //clear buffer

fpi = popen("ps auxw", "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;

}

* Output

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe1

Output from pipe: USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND

georgesuarez 25952 5.7 1.4 5075764 242616 ?? S 8:56AM 1:13.66 /Applications/Google Chrome.app/Contents/MacOS/Google Chrome

georgesuarez 26313 1.3 0.0 4276948 832 s000 S+ 9:27AM 0:00.00 ./pipe1

georgesuarez 25978 0.9 0.8 4574400 142220 ?? S 8:56AM 0:04.49 /Applications/Utilities/Terminal.app/Contents/MacOS/Terminal

georgesuarez 25931 0.7 2.4 8111744 409236 ?? S 8:55AM 2:07.43 /Applications/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn\_0\_1097996

root 26314 0.6 0.0 4268148 1100 s000 R+ 9:27AM 0:00.00 ps auxw

\_windowserver 162 0.5 0.7 5956228 113804 ?? Ss Mon08AM 36:07.05 /System/Library/PrivateFrameworks/SkyLight.framework/Resources/WindowServer -daemon

root 397 0.3 0.0 4331732 5096 ?? Ss Mon08AM 0:08.63 /usr/libexec/taskgated -s

georgesuarez 25974 0.2 0.6 5229304 104344 ?? S

* Explanation:
  + The program opens a pipe for which it passes a command which is *ps auxw* which is passed into a buffer to output of the command.
* Modify the program **pipe1.cpp** to **pipe1a.cpp** so that it accepts a command (e.g. "ls -l") from the keyboard. For example, when you execute "./pipe1a ps -auxw", it should give you the same output as **pipe1.cpp**.
* *pipe1a.cpp*

//pipe1.cpp

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <iostream>

using namespace std;

int main(int argc, char \*argv[])

{

FILE \*fpi; //for reading a pipe

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

for (int i = 1; i < argc; ++i)

{

strcat(buffer, argv[i]);

strcat(buffer, " ");

}

int chars\_read;

fpi = popen(buffer, "r");

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;

}

memset(buffer, 0, sizeof(buffer)); //clear buffer

return 1;

}

* Output:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe1a ls -l

Output from pipe: total 360

-rwxr-xr-x 1 georgesuarez staff 8932 Apr 24 22:15 client

-rw-r--r--@ 1 georgesuarez staff 2172 Apr 24 22:13 client.cpp

-rwxr-xr-x 1 georgesuarez staff 16056 Apr 22 00:20 pipe1

-rw-r--r-- 1 georgesuarez staff 640 Apr 22 00:20 pipe1.cpp

-rwxr-xr-x 1 georgesuarez staff 16104 Apr 23 18:59 pipe1a

-rw-r--r-- 1 georgesuarez staff 778 Apr 23 18:59 pipe1a.cpp

-rwxr-xr-x 1 georgesuarez staff 8748 Apr 24 10:59 pipe2

-rw-r--r--@ 1 georgesuarez staff 689 Apr 24 11:00 pipe2.cpp

-rwxr-xr-x 1 georgesuarez staff 8748 Apr 24 10:59 pipe2a

-rw-r--r--@ 1 georgesuarez staff 761 Apr 24 10:59 pipe2a.cpp

-rwxr-xr-x 1 georgesuarez staff 16224 Apr 24 11:00 pipe3

-rw-r--r--@ 1 georgesuarez staff 712 Apr 24 11:00 pipe3.cpp

-rwxr-xr-x 1 georgesuarez staff 8916 Apr 24 16:13 pipe4

-rw-r--r-- 1 georgesuarez staff 992 Apr 24 16:13 pipe4.cpp

-rwxr-xr-x 1 georgesuarez staff 20080 Apr 24 22:10 pipe4a

-rw-r--r-- 1 georgesuarez staff 1508 Apr 25 09:10 pipe4a.cpp

drwxr-xr-x 3 ص/??

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe1a ps auxw

Output from pipe: USER PID %CPU %MEM VSZ RSS TT STAT STARTED TIME COMMAND

georgesuarez 27961 4.9 1.4 5088640 240276 ?? S 12:05PM 0:26.62 /Applications/Google Chrome.app/Contents/MacOS/Google Chrome

georgesuarez 27991 0.6 0.7 4508052 114828 ?? S 12:10PM 0:01.01 /Applications/Utilities/Terminal.app/Contents/MacOS/Terminal

\_windowserver 162 0.5 0.8 5953896 133448 ?? Ss Mon08AM 42:27.77 /System/Library/PrivateFrameworks/SkyLight.framework/Resources/WindowServer -daemon

georgesuarez 27981 0.4 0.6 5206612 99772 ?? S 12:05PM 0:04.92 /Applications/Google Chrome.app/Contents/Versions/65.0.3325.181/Google Chrome Helper.app/Contents/MacOS/Google Chrome Helper --type=renderer --field-trial-handle=6845048885018761593,12072289396916254927,131072 --service-pipe-token=4C04DCAE9B701CC5D6F788DC608BBF56 --lang=en-US --metrics-client-id=1315d3b5-788e-4b91-b3a6-a9dd71f0f911 --enable-offline-auto-reload --enable-offline-auto-reload-visible-only --num-rast?U??

* What do you see when you execute "pipe2"? Why?
* *pipe2.cpp*

//pipe2.cpp

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <iostream>

using namespace std;

int main()

{

FILE \*fpo; //for writing to a pipe

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

//Write buffer a message

sprintf(buffer, "Arnod said, 'If I am elected, ..', and the fairy tale begins\n");

fpo = popen("od -c", "w"); //pipe to command "od -c"

//od -- output dump, see "man od"

if (fpo != NULL)

{

//send data from buffer to pipe

fwrite(buffer, sizeof(char), strlen(buffer), fpo);

pclose(fpo); //close the pipe

return 0;

}

return 1;

}

* Output

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe2

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

* Explanation:
  + A literal string is being stored in the buffer using the *sprintf()* function, and a pipe is being open using the *popen()* function with the command *od -c* where *od­* is the command that filters what is being displayed from either a specified file or standard input in a user specified format which in this case a *-c* option is passed which outputs *C-styled* escape characters. Then, the buffer is being processed in the pipe using the *fwrite()* function which outputs what is being processed.
* Modify the program so that it prints out the first three words of the sentence in reverse by making use of **awk** (see lab 2) (i.e. 'If said, Arnod....).
* *pipe2a.cpp*

//pipe2a.cpp

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <iostream>

using namespace std;

int main()

{

FILE \*fpo; //for writing to a pipe

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

//Write buffer a message

sprintf(buffer, "Arnod said, 'If I am elected, ..', and the fairy tale begins\n");

fpo = popen("od -c", "w"); //pipe to command "od -c"

//od -- output dump, see "man od"

fpo = popen("awk ' { for (i = 3; i > 0; i--) printf $i }' ", "w");

if (fpo != NULL)

{

//send data from buffer to pipe

fwrite(buffer, sizeof(char), strlen(buffer), fpo);

pclose(fpo); //close the pipe

return 0;

}

return 1;

}

* Output

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe2a

'Ifsaid,Arnod**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

1. The Pipe Call

* What do you see when you execute "pipe3"? Why?
* *pipe3.cpp*

//pipe3.cpp

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

#include <iostream>

using namespace std;

int main()

{

int nbytes;

int fd[2]; //file descriptors for pipe

const char s[] = "CSUSB";

char buffer[BUFSIZ + 1];

memset(buffer, 0, sizeof(buffer)); //clear buffer

if (pipe(fd) == 0)

{ //create a pipe

nbytes = write(fd[1], s, strlen(s)); //send data to pipe

cout << "Sent " << nbytes << " bytes to pipe." << endl;

nbytes = read(fd[0], buffer, BUFSIZ); //read data from pipe

cout << "Read " << nbytes << " from pipe: " << buffer << endl;

return 0;

}

return 1;

}

* Output

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe3

Sent 5 bytes to pipe.

Read 5 from pipe: CSUSB

* Explanation:
  + The program creates an array of file descriptors for the pipe to write and read from. The *nbytes* is being used to hold the number of bytes that is being processed from one file descriptor to another using the same pipe. Since one file descriptor is holding the data, “CSUSB”, that means that *nbytes* is holding 5 bytes since each character is 1 byte which is why 5 bytes is being outputted from this program.

1. Parent and Child Processes
   * **Modify pipe4.cpp** so that it accepts a message from the keyboard and sends it to **pipe5**.
   * *pipe4.cpp*

//pipe4.cpp (data producer)

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <iostream>

using namespace std;

int main(int argc, char \*argv[])

{

int data\_processed;

int file\_pipes[2];

char buffer[BUFSIZ + 1];

pid\_t fork\_result;

memset(buffer, '\0', sizeof(buffer));

int index = 0;

cout << "Input a message to send to pipe5: ";

while (cin >> buffer[index])

{

if (cin.peek() == '\n')

{

break;

}

else

{

buffer[index++];

}

}

if (pipe(file\_pipes) == 0)

{ //creates pipe

fork\_result = fork();

if (fork\_result == (pid\_t)-1)

{ //fork fails

fprintf(stderr, "Fork failure");

exit(EXIT\_FAILURE);

}

if (fork\_result == 0)

{ //child

sprintf(buffer, "%d", file\_pipes[0]);

(void)execl("pipe5", "pipe5", buffer, (char \*)0);

exit(EXIT\_FAILURE);

}

else

{ //parent

data\_processed = write(file\_pipes[1], buffer,

strlen(buffer));

printf("%d - wrote %d bytes\n", getpid(), data\_processed);

}

}

exit(EXIT\_SUCCESS);

}

* + *pipe5.cpp*

// The 'consumer' program, pipe5.cpp, that reads the data is much simpler.

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int main(int argc, char \*argv[])

{

int data\_processed;

char buffer[BUFSIZ + 1];

int file\_descriptor;

memset(buffer, '\0', sizeof(buffer));

sscanf(argv[1], "%d", &file\_descriptor);

data\_processed = read(file\_descriptor, buffer, BUFSIZ);

printf("%d - read %d bytes: %s\n", getpid(), data\_processed, buffer);

exit(EXIT\_SUCCESS);

}

* + Output

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./pipe4a

Input a message to send to pipe5: Hello from pipe 4

28170 - wrote 14 bytes

28171 - read 14 bytes: Hellofrompipe4

1. Special Pipes
   * *fifo1.cpp*

//fifo1.cpp

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/stat.h>

int main()

{

int res = mkfifo("/tmp/my\_fifo", 0777);

if (res == 0)

printf("FIFO created\n");

exit(EXIT\_SUCCESS);

}

* + Output

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./fifo1

FIFO created

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ls -lF /tmp/my\_fifo

prwxr-xr-x 1 georgesuarez wheel 0 Apr 25 12:24 /tmp/my\_fifo|

* + *server.cpp*

//server.cpp

#include <ctype.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <limits.h>

#include <sys/types.h>

#include <sys/stat.h>

#define SERVER\_FIFO\_NAME "/tmp/serv\_fifo"

#define CLIENT\_FIFO\_NAME "/tmp/client\_fifo"

#define BUFFER\_SIZE 20

struct data\_to\_pass\_st

{

pid\_t client\_pid;

char some\_data[BUFFER\_SIZE - 1];

};

int main()

{

int server\_fifo\_fd, client\_fifo\_fd;

struct data\_to\_pass\_st my\_data;

int read\_res;

char client\_fifo[256];

char \*tmp\_char\_ptr;

mkfifo(SERVER\_FIFO\_NAME, 0777);

server\_fifo\_fd = open(SERVER\_FIFO\_NAME, O\_RDONLY);

if (server\_fifo\_fd == -1)

{

fprintf(stderr, "Server fifo failure\n");

exit(EXIT\_FAILURE);

}

sleep(10); /\* lets clients queue for demo purposes \*/

do

{

read\_res = read(server\_fifo\_fd, &my\_data, sizeof(my\_data));

if (read\_res > 0)

{

// In this next stage, we perform some processing on the data just read from the client.

// We convert all the characters in some\_data to uppercase and combine the CLIENT\_FIFO\_NAME

// with the received client\_pid.

tmp\_char\_ptr = my\_data.some\_data;

while (\*tmp\_char\_ptr)

{

\*tmp\_char\_ptr = toupper(\*tmp\_char\_ptr);

tmp\_char\_ptr++;

}

sprintf(client\_fifo, CLIENT\_FIFO\_NAME, my\_data.client\_pid);

// Then we send the processed data back, opening the client pipe in write-only, blocking mode.

// Finally, we shut down the server FIFO by closing the file and then unlinking the FIFO.

client\_fifo\_fd = open(client\_fifo, O\_WRONLY);

if (client\_fifo\_fd != -1)

{

write(client\_fifo\_fd, &my\_data, sizeof(my\_data));

close(client\_fifo\_fd);

}

}

} while (read\_res > 0);

close(server\_fifo\_fd);

unlink(SERVER\_FIFO\_NAME);

exit(EXIT\_SUCCESS);

}

* + *client.cpp*

#include <sys/types.h>

#include <sys/stat.h>

#define SERVER\_FIFO\_NAME "/tmp/serv\_fifo"

#define CLIENT\_FIFO\_NAME "/tmp/client\_fifo"

#define BUFFER\_SIZE 20

struct data\_to\_pass\_st

{

pid\_t client\_pid;

char some\_data[BUFFER\_SIZE - 1];

};

int main()

{

int server\_fifo\_fd, client\_fifo\_fd;

struct data\_to\_pass\_st my\_data;

int times\_to\_send;

char client\_fifo[256];

server\_fifo\_fd = open(SERVER\_FIFO\_NAME, O\_WRONLY);

if (server\_fifo\_fd == -1)

{

fprintf(stderr, "Sorry, no server\n");

exit(EXIT\_FAILURE);

}

my\_data.client\_pid = getpid();

//sprintf(client\_fifo, CLIENT\_FIFO\_NAME, my\_data.client\_pid);

sprintf(client\_fifo, CLIENT\_FIFO\_NAME);

if (mkfifo(client\_fifo, 0777) == -1)

{

fprintf(stderr, "Sorry, can't make %s\n", client\_fifo);

exit(EXIT\_FAILURE);

}

// For each of the five loops, the client data is sent to the server.

// Then the client FIFO is opened (read-only, blocking mode) and the data read b ack.

// Finally, the server FIFO is closed and the client FIFO removed from memory.

for (times\_to\_send = 0; times\_to\_send < 5; times\_to\_send++)

{

sprintf(my\_data.some\_data, "Hello from %d", my\_data.client\_pid);

printf("%d sent %s, ", my\_data.client\_pid, my\_data.some\_data);

write(server\_fifo\_fd, &my\_data, sizeof(my\_data));

client\_fifo\_fd = open(client\_fifo, O\_RDONLY);

if (client\_fifo\_fd != -1)

{

if (read(client\_fifo\_fd, &my\_data, sizeof(my\_data)) > 0)

{

printf("received: %s\n", my\_data.some\_data);

}

close(client\_fifo\_fd);

}

}

close(server\_fifo\_fd);

unlink(client\_fifo);

exit(EXIT\_SUCCESS);

}

* + Output:
    - Terminal 1:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./server

* + - Terminal 2:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./client

28484 sent Hello from 28484, received: HELLO FROM 28484

28484 sent Hello from 28484, received: HELLO FROM 28484

28484 sent Hello from 28484, received: HELLO FROM 28484

28484 sent Hello from 28484, received: HELLO FROM 28484

28484 sent Hello from 28484,

* + *server.cpp (modified)*:

//server.cpp

#include <ctype.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <fcntl.h>

#include <limits.h>

#include <sys/types.h>

#include <sys/stat.h>

#define SERVER\_FIFO\_NAME "/tmp/serv\_fifo"

#define CLIENT\_FIFO\_NAME "/tmp/client\_fifo"

#define BUFFER\_SIZE 20

struct data\_to\_pass\_st

{

pid\_t client\_pid;

char some\_data[BUFFER\_SIZE - 1];

};

int main()

{

int server\_fifo\_fd, client\_fifo\_fd;

struct data\_to\_pass\_st my\_data;

int read\_res;

char client\_fifo[256];

char \*tmp\_char\_ptr;

mkfifo(SERVER\_FIFO\_NAME, 0777);

server\_fifo\_fd = open(SERVER\_FIFO\_NAME, O\_RDONLY);

if (server\_fifo\_fd == -1)

{

fprintf(stderr, "Server fifo failure\n");

exit(EXIT\_FAILURE);

}

sleep(10); /\* lets clients queue for demo purposes \*/

do

{

read\_res = read(server\_fifo\_fd, &my\_data, sizeof(my\_data));

if (read\_res > 0)

{

// In this next stage, we perform some processing on the data just read from the client.

// We convert all the characters in some\_data to uppercase and combine the CLIENT\_FIFO\_NAME

// with the received client\_pid.

tmp\_char\_ptr = my\_data.some\_data;

while (\*tmp\_char\_ptr)

{

\*tmp\_char\_ptr = tolower(\*tmp\_char\_ptr);

tmp\_char\_ptr++;

}

sprintf(client\_fifo, CLIENT\_FIFO\_NAME, my\_data.client\_pid);

// Then we send the processed data back, opening the client pipe in write-only, blocking mode.

// Finally, we shut down the server FIFO by closing the file and then unlinking the FIFO.

client\_fifo\_fd = open(client\_fifo, O\_WRONLY);

if (client\_fifo\_fd != -1)

{

write(client\_fifo\_fd, &my\_data, sizeof(my\_data));

close(client\_fifo\_fd);

}

}

} while (read\_res > 0);

close(server\_fifo\_fd);

unlink(SERVER\_FIFO\_NAME);

exit(EXIT\_SUCCESS);

}

* + Output:

**georgesuarez at MacBook-Pro in ~/University/CSE-460/Labs/Lab 4 on master\***

**$** ./client

28568 sent Hello from 28568, received: hello from 28568

28568 sent Hello from 28568, received: hello from 28568

28568 sent Hello from 28568, received: hello from 28568

28568 sent Hello from 28568, received: hello from 28568

28568 sent Hello from 28568, received: hello from 28568

1. Study of XV6
   * *cp.c*

#include "types.h"

#include "stat.h"

#include "user.h"

#include "fcntl.h"

char buf[512];

int

main(int argc, char \*argv[])

{

int fd0, fd1, fd2, n;

if(argc <= 2){

printf(1, "Need 2 arguments!\n");

exit();

}

if((fd0 = open(argv[1], O\_RDONLY)) < 0){

printf(1, "cp: cannot open %s\n", argv[1]);

exit();

}

if((fd1 = open(argv[2], O\_CREATE|O\_RDWR)) < 0){

printf(1, "cp: cannot open %s\n", argv[2]);

exit();

}

if ((fd2 = open(argv[3], O\_CREATE|O\_RDWR)) < 0){

printf(1, "cp: cannot open %s\n", argv[3]);

exit();

}

while ( ( n = read ( fd0, buf, sizeof(buf))) > 0 ){

write ( fd1, buf, n );

write ( fd2, buf, n );

}

close(fd0);

close(fd1);

close(fd2);

exit();

}

* + Output

cpu1: starting 1

cpu0: starting 0

sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58

init: starting sh

$ ls

. 1 1 512

.. 1 1 512

README 2 2 2290

cat 2 3 13672

echo 2 4 12680

forktest 2 5 8116

grep 2 6 15548

init 2 7 13268

kill 2 8 12732

ln 2 9 12636

ls 2 10 14820

mkdir 2 11 12812

rm 2 12 12796

sh 2 13 23280

stressfs 2 14 13460

usertests 2 15 56396

wc 2 16 14212

cp 2 17 13420

zombie 2 18 12460

console 3 19 0

$ cp README myFile1 myFile2

$ ls

. 1 1 512

.. 1 1 512

README 2 2 2290

cat 2 3 13672

echo 2 4 12680

forktest 2 5 8116

grep 2 6 15548

init 2 7 13268

kill 2 8 12732

ln 2 9 12636

ls 2 10 14820

mkdir 2 11 12812

rm 2 12 12796

sh 2 13 23280

stressfs 2 14 13460

usertests 2 15 56396

wc 2 16 14212

cp 2 17 13420

zombie 2 18 12460

console 3 19 0

myFile1 2 20 2290

myFile2 2 21 2290

Discussion:

I successfully completed all the sections so I would give myself **20/20** points.