Assignment 5

1. a. When the child run reaches the end, it blocks the call to read from the pipe. Because the close was removed, the pipe isn’t closed in the parent and the child blocks the read until the program ends.  
   b. The parent code reads the file and writes to the pipe buffer and closes after. When reading the file from the child, the output depended on what pager was used. Using the ‘more’ pager, it printed the first page of output and closed, while using less pager didn’t output anything. Because the parent does not wait for the child to finish, it terminates early and is not able to print the whole file.
2. If the argument passed to popen is a nonexistent command, then the output will be Null or error due to it being unable to run the command and open the pipe. Technically the popen command will still return a valid file pointer but considering the fact that the command does not exist it will return a value such as NULL or end of file error because the shell will try to execute a nonexistent command. The following output and code demonstrate what a successful popen command looks like and then what occurs when a nonexistent command is fed to popen.   
   Code:

#include <stdio.h>

#include <stdlib.h>

int main() {

FILE \*a;

char b[128];

a = popen("bad command", "r");

if (a == NULL) {

printf("couldn't open pipe\n");

} else {

if (fgets(b, sizeof(b), a) == NULL) {

printf("command could not be executed\n");

}

pclose(a);

}

return 0;

}

Output:

A screen shot of a computer

Description automatically generated

1. 1. Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define MAXLINE 4096

void err\_sys(const char \*msg) {

perror(msg);

exit(EXIT\_FAILURE);

}

void err\_msg(const char \*msg) {

fprintf(stderr, "%s\n", msg);

}

int main(void) {

int n, fd1[2], fd2[2];

pid\_t pid;

char line[MAXLINE];

if (pipe(fd1) < 0 || pipe(fd2) < 0)

err\_sys("pipe error");

if ((pid = fork()) < 0) {

err\_sys("fork error");

} else if (pid > 0) { /\* parent \*/

close(fd1[0]);

close(fd2[1]);

while (fgets(line, MAXLINE, stdin) != NULL) {

n = strlen(line);

if (write(fd1[1], line, n) != n)

err\_sys("write error to pipe");

if ((n = read(fd2[0], line, MAXLINE)) < 0)

err\_sys("read error from pipe");

if (n == 0) {

err\_msg("child closed pipe");

break;

}

line[n] = 0; /\* null terminate \*/

if (fputs(line, stdout) == EOF)

err\_sys("fputs error");

}

if (ferror(stdin))

err\_sys("fgets error on stdin");

exit(0);

} else { /\* child \*/

close(fd1[1]);

close(fd2[0]);

if (fd1[0] != STDIN\_FILENO) {

if (dup2(fd1[0], STDIN\_FILENO) != STDIN\_FILENO)

err\_sys("dup2 error to stdin");

close(fd1[0]);

}

if (fd2[1] != STDOUT\_FILENO) {

if (dup2(fd2[1], STDOUT\_FILENO) != STDOUT\_FILENO)

err\_sys("dup2 error to stdout");

close(fd2[1]);

}

if (execl("./add2", "add2", (char \*)0) < 0)

err\_sys("execl error");

}

exit(0);

}

Output: A computer screen with blue text

Description automatically generated  
I can the parent was terminated by SIGPIPE because of the exit code from the program correlated to the code for SIGPIPE (13). 

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#define MAXLINE 4096

void err\_sys(const char \*msg) {

perror(msg);

exit(EXIT\_FAILURE);

}

void err\_msg(const char \*msg) {

fprintf(stderr, "%s\n", msg);

}

int main(void) {

int fd1[2], fd2[2];

pid\_t pid;

char line[MAXLINE];

FILE \*fpin, \*fpout;

if (pipe(fd1) < 0 || pipe(fd2) < 0)

err\_sys("pipe error");

if ((pid = fork()) < 0) {

err\_sys("fork error");

} else if (pid > 0) { /\* parent \*/

close(fd1[0]);

close(fd2[1]);

fpout = fdopen(fd1[1], "w");

fpin = fdopen(fd2[0], "r");

while (fgets(line, MAXLINE, stdin) != NULL) {

if (fputs(line, fpout) == EOF)

err\_sys("fputs error to pipe");

fflush(fpout);

if (fgets(line, MAXLINE, fpin) == NULL)

err\_msg("child closed pipe");

else {

if (fputs(line, stdout) == EOF)

err\_sys("fputs error to stdout");

}

}

if (ferror(stdin))

err\_sys("fgets error on stdin");

fclose(fpout);

fclose(fpin);

waitpid(pid, NULL, 0);

exit(0);

} else { /\* child \*/

close(fd1[1]);

close(fd2[0]);

fpin = fdopen(fd1[0], "r");

fpout = fdopen(fd2[1], "w");

if (dup2(fd1[0], STDIN\_FILENO) != STDIN\_FILENO)

err\_sys("dup2 error to stdin");

if (dup2(fd2[1], STDOUT\_FILENO) != STDOUT\_FILENO)

err\_sys("dup2 error to stdout");

fclose(fpin);

fclose(fpout);

if (execl("./add2", "add2", (char \*)0) < 0)

err\_sys("execl error");

}

exit(0);

}

1. Each platform treats the stat options differently for example windows does not let you use fstat and requires a different socket function but it still does not provide the same information that you can get by running stat functions like fstat on a platform like Unix or ubuntu. For the online C compiler compared to the Linux ran code, they both could support the same stat structure members, with the main difference being their values for certain ones. For example, the i-node numbers, group and user ID numbers were different.   
   Code:

#include <stdio.h>

#include <sys/socket.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

int

main(void)

{

struct stat statbuf = {};

const int fd = socket(AF\_UNIX, SOCK\_STREAM, 0);

if (fd < 0) {

perror("socket");

return 1;

}

if (fstat(fd, &statbuf) < 0) {

perror("fstat");

return 1;

}

printf("st\_mode: %u\n", statbuf.st\_mode);

printf("st\_ino: %lu\n", statbuf.st\_ino);

printf("st\_dev: %lu\n", statbuf.st\_dev);

printf("st\_rdev: %lu\n", statbuf.st\_rdev);

printf("st\_nlink: %lu\n", statbuf.st\_nlink);

printf("st\_uid: %u\n", statbuf.st\_uid);

printf("st\_gid: %u\n", statbuf.st\_gid);

printf("st\_size: %lu\n", statbuf.st\_size);

printf("st\_atime: %lu\n", statbuf.st\_atime);

printf("st\_mtime: %lu\n", statbuf.st\_mtime);

printf("st\_ctime: %lu\n", statbuf.st\_ctime);

printf("st\_blksize: %lu\n", statbuf.st\_blksize);

printf("st\_blocks: %lu\n", statbuf.st\_blocks);

close(fd);

return 0;

}

Output for Ubuntu:  
A computer screen with blue text

Description automatically generated  
Output from online C compiler:  
A screen shot of a computer

Description automatically generated

1. Regular expression is essentially a search pattern. This search pattern allows you to look up certain instances of a phrase, or other instances of text that occur within data. Regular expressions can also be utilized in text replacement. Regular expressions return the location of the instance that you search up using the search() method and then utilizing the replace() method they allow you to change that text. Regular expressions do not have to be case sensitive and this broadens their ability to search for text or data but it requires the use of modifiers. By using brackets, regular expressions allow for the lookup of certain range of characters or numbers. They can also find special instances of characters like spaces or digits. Quantifiers also allow for the expansion of their abilities because it allows for more broad results from a search. They dictate quantities of occurrence of characters to be searched for.

Cloud Computing is the ability to obtain certain system resources without direct interference by the user. This pertains to mainly computing power and storage. Clouds are often spread apart and are able to do this by the use of various data centers. Cloud computing is built on the features of being able to access shared resources concurrently and also allows for crash protection. Cloud computing allows for multiple users to update systems and access them at the same time. Cloud computing services can further be split into two different types provided. Infrastructure as a service and platform as a service are the two types. Infrastructure as a service pertains to providing users with access to information and resources through the use of various types of servers. Platform as a service pertains to providing users, mainly developers, with a platform that allows for active updating, a variety of tools, and the running of various applications from anywhere. The servers are all hosted from the cloud provider as well as the Operating systems. This enables developers to build, test, and maintain applications from anywhere as well as concurrently.