

CSCI 340 - OPERATING SYSTEMS

Assignment 3 Total Points 50

Objectives

In this assignment you will extend the simple *command line interpreter* (or *shell*). This assignment will allow you to gain experience in the following areas:

- **More 'C' Programming:** This includes variable declaration, data types, arrays, pointers, operators, expressions, selection statements, looping statements, functions, structs, and header files.
- **Process Communication:** Extend your knowledge of processes (i.e. fork, exec, and wait) by including process communication functionality into your shell program.
- **File Descriptor Redirection:** Learn how to redirect stdout information into to a text file.

System and Standard Lib Functions

In addition to the system and standard library functions used in assignment 2, this assignment will also use the system and standard library functions listed below. Please become familiar with the syntax and usage of these calls. Detailed information about each function listed below can be found using the man command from the console: i.e. `man pipe`, will show the man page (short for manual page) for the pipe function.

- **Process Communication:** `int pipe(int filedes[2])`
- **Duplicate File Descriptor:** `int dup(int fildes)`
- **Duplicate a File Descriptor:** `int dup2(int oldfd, int newfd)`
- **Create/Open a New File Descriptor:** `int open(const char* pathname, int flags, mode_t mode)`

Provided Files

This code is a basic working example of a shell. In the attached tarball there are three files: *hw2.c*, *shell.c*, and *shell.h*. The **only file to be modified** is *shell.c* (see Todo below), neither *hw2.c* nor *shell.h* need to be modified in this assignment.

Todo

To include pipe and dup functionality into your shell, only the `int execute(command_t* p_cmd)` function in the *shell.c* file needs to be modified. Furthermore, only a single pipe operation will be used in this assignment. The `|` shell symbol represents a UNIX pipe operation. For instance, `ls | wc -l` is an example shell command that “pipes” the stdout of the `ls` shell command to the `wc` word count shell command. For further help, see the Additional Guidance section at the end of this document.

Redirect stdin to a text file. This operation can be identified by determining if a greater than (`>`) is present in the command struct `argv` array. For instance, an example shell command would be `ls -lrot > text.txt`.

Collaboration and Plagiarism

This is an **individual assignment**, i.e. **no collaboration is permitted**. Plagiarism will not be tolerated. Submitted solutions that are very similar (determined by the instructor) will be given a grade of zero. Please do your own work, and everything will be OK.

Submission

Create a compressed tarball, i.e. *tar.gz*, that only contains the completed *hw2.c*, *shell.h* and *shell.c* files. The name of the compressed tarball must be your last name. For example, *ritchie.tar.gz* would be correct if the original co-developer of UNIX (Dennis Ritchie) submitted the assignment. Only assignments submitted in the correct format will be accepted (no exceptions). Submit the compressed tarball (via OAKS) to the Dropbox setup for this assignment. You may resubmit the compressed tarball as many times as you like, Dropbox will only keep the newest submission.

To be fair to everyone, late assignments will not be accepted. Exceptions will only be made for extenuating circumstances, i.e. death in the family, health related problems, etc. You will be given a week to complete this assignment. Poor time management is not excuse. Please do not email assignment after the due date, it will not be accepted. Please feel free to setup an appointment to discuss the assigned coding problem. I will be more than happy to listen to your approach and make suggestions. However, I cannot tell you how to code the solution. Additionally, code debugging is your job. You may use the debugger (gdb) or print statements to help understand why your solution is not working correctly, your choice.

Grading Rubric

For this assignment the grading rubric is provided in the table shown below.

Program Compiles	10 points
Program Runs with no errors	10 points
Pipe function implementation	15 points
File redirect implementation	15 points
Five test cases (10 points each) * 50 points	

In particular, the assignment will be graded as follows, if the submitted solution

- does not compile: 0 of 100 points
- compiles but does not run: 10 of 100 points
- compiles and runs without errors: 20 of 100 points
- all functions correctly implemented: 50 of 100 points
- all test cases are correct: 100 of 100 points

Additional Guidance

UNIX Pipe

Pipes are the oldest form of UNIX Inter-process communication (IPC) and are provided by all UNIX systems. They have two limitations:

1. They are half-duplex, i.e. data only flows in one direction.

2. They can be used only between processes that have a common ancestor. Normally a pipe is created by a process, that process calls a fork, and the pipe is used between the parent and child.

A pipe is created by calling the `int pipe(int fd[2])` function. Two file descriptors are returned through the `fd` argument: `fd[0]` is open for reading, and `fd[1]` is open for writing. Normally, the process that calls `pipe` then calls `fork`, creating an IPC channel from the parent to the child (or vice versa). Figure 1 illustrates this concept. What happens after the `fork` depends on which direction of data

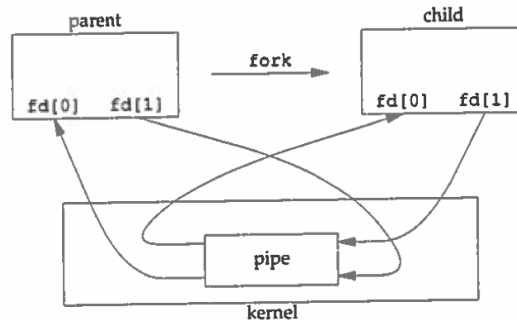


Figure 1: UNIX pipe concept between parent and child process.

flow we want. For a pipe from parent to the child, the parent closes the read end of the pipe (`fd[0]`) and the child closes the write end (`fd[1]`). For a pipe from the child to the parent, the parent closes `fd[1]` and the child closes `fd[0]`.

Duplicate File Descriptor

An existing file descriptor is duplicated using `int dup(int fildes)`. In general, if `fildes` is 0 then the standard in (`stdin`) is duplicated, and if `fildes` is 1 then the standard out is duplicated. These duplicated descriptors can be used interchangeably by the parent and child process.

Example code that uses pipe and dup

An example code segment that illustrates how to code the equivalent `ls | wc -l` shell command. Please note, this example is not considered complete, and is only meant to guide you in this coding assignment.

```
int main( int argc, char** argv ) {

    int child_process_status;
    int fds[2];
    pid_t cpid1, cpid2;
    char* pargs[] = { "ls", NULL };
    char* cargs[] = { "wc", "-l", NULL };

    pipe( fds );

    if ( cpid1 = fork() == 0 ) {
        close(1);                /* close normal stdout */
        dup( fds[1] );           /* make stdout same as fds[1] */
        close( fds[0] );         /* we don't need this */
        execv("/bin/ls", pargs);
    }

    if ( cpid2 = fork() == 0 ) {
        close(0);                /* close normal stdin */
        dup( fds[0] );           /* make stdin same as fds[0] */
    }

    child_process_status = wait(&pid, NULL);
    return 0;
}
```

```

        close( fds[1] );          /* we don't need this */
        execv("/usr/bin/wc", cargs);
    }

    close( fds[0] );
    close( fds[1] );
    waitpid( cpid1, &child_process_status, 0 );
    waitpid( cpid2, &child_process_status, 0 );
    return 0;
}

```

File Descriptor Redirection

An example code segment that illustrates how redirect `stdout` to a text file named *test.txt* using the `dup2` system call. Please note, this example is **not considered complete**, and **should not be blindly copied from here into your homework soution**. The provided example is only meant to guide you in this coding assignment.

```

int main( int argc, char** argv ) {

    int child_process_status;
    int outfile;
    pid_t cpid1;

    // This example is simulating a ls -lrot > test.txt command

    char* pargs[] = { "ls", "-lrot", NULL };
    outfile = open( "test.txt", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR | S_IRGRP );

    if ( outfile == -1 ) {

        fprintf(stderr, "failed to open file\n");

    } else if ( cpid1 = fork() == 0 ) {

        dup2( outfile, 1 );
        execv("/bin/ls", pargs);
        exit(-1);

    }

    close( outfile );
    waitpid( cpid1, &child_process_status, 0 );
    return 0;
}

```

My information about the `dup2` function can be found in Section 3.12 in the Advanced Programming in the UNIX Environment (APUE) textbook written by W. Richard Stevens. A PDF version of the APUE textbook has been placed on OAKS.