CSCI 340 - Operating Systems I

Assignment 4 (Part III of III) Total Points 50

Objectives

In this assignment you will extend the simple *command line interpreter* (or *shell*) developed in assignments 2 and 3. This assignment will allow you to gain experience, or extend your knowledge, 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.
- **File Descriptor Redirection**: Learn how to redirect `stdout` information into to a text file.
- **Non-Blocking Operations**: Learn how to *fork* and *execute* a child process that runs in the background while the parent continues to work (i.e. no *wait* operation is performed).
- **Signals**: Learn how to write signal handlers used in asynchronous non-blocking operations.

System and Standard Lib Functions

In addition to the system and standard library functions used in assignments 2 and 3, 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 dup2`, will show the man page (short for manual page) for the `dup2` function.

- **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)`
- **Signals**: `sighandler_t signal(int signum, sighandler_t handler)`

Provided Files

Modify the *hw3.c* and *shell.c* files submitted in assignment 3. Instructor solutions for *hw3.c* and *shell.c* will be provided and may be used as a starting point for your assignment 4.

Todo

Please perform the following three tasks:

1. **Include Header Files**: In the *hw3.c* and *shell.c* files include `fcntl.h` and `signal.h`.
2. **Modify Execute Function**: Add the following shell operations:
    - Fork a child process that runs in the background. This operation can be identified by determining if an ampersand (&) is the last element in the command struct `argv` array. For instance, and example shell command would be "`geany &`", where `geany` is a text editor in Puppy Linux.

- Redirect `stdout` 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, and example shell command would be "`ls -lrot > text.txt`".

3. **Add Signal Hander**: Add a `SIGCHLD` signal handler function (prototype and implementation) in the *hw3.c* that is able to asynchronously reclaim (or reap) a forked child process that has terminated normally.

For further help regarding a background process, implementing a signal handler, and file redirection, see the Additional Guidance section at the end of this document. Please note, *shell.h* does not need to be modified in this assignment.

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 *hw4.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
signal handler implementation	5 points
execute() function implementation	25 points

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

- does not compile: 0 of 50 points
- compiles but does not run: 10 of 50 points
- compiles and runs with errors: 15 of 50 points
- compiles and runs without errors: 20 of 50 points
- all functions correctly implemented: 50 of 50 points

Additional Guidance

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.

Signal Handler and Background Processes

An example code segment that illustrates how implement two different signal handlers. Specifically. the `SIGCHLD` signal handler is used by the parent process to reclaim forked child process that ran in the background and exited normally, and the `SIGINT` signal handler is used to handle *Ctrl-C* signals sent by the user to terminate the running process. 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.

```
static void sig_child_handler( int sig );
static void sig_int_handler( int sig );

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

        pid_t pid;
        char* pargs[] = { "gedit", NULL };

        if ( signal( SIGCHLD, sig_child_handler ) == SIG_ERR ) {
```

```
                perror("Unable to create new SIGCHLD signal handler!");
                exit(-1);
        }

        if ( signal( SIGINT, sig_int_handler ) == SIG_ERR ) {
                perror("Unable to create new SIGINT signal handler!");
                exit(-1);
        }

        pid = fork();

        if ( pid == 0 ) {
                execv("/usr/bin/gedit", pargs);
                perror("Child process terminated in error condition!");
                exit(-1);
        }

        // endless loop that sleeps every second
        while ( 1 ) {
                printf("parent is working ... la la la ...\n");
                sleep(1);
        }
        return 0;
}

static void sig_int_handler( int sig ) {
        printf("In SIGINT handler\n");
        exit( 0 );
}

static void sig_child_handler( int sig ) {
        printf("In SIGCHLD handler\n");
        int status;
        pid_t pid;

        while ( ( pid = waitpid(-1, &status,  WNOHANG ) ) != -1 )  {
                  printf("Child Process (%d) has Terminated\n", pid );
        }
}
```

My information about the `signal` function can be found in Section 1.9 and Chapter 10 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.