

SJSU CS 149 HW1 SPRING 2020

REMINDER: Each homework is **individual**. "Every single byte must come from you." Cut&paste from others is **not** allowed. Keep your answer and source code to yourself **only** - **never** post or share them to any site in any way.

[Type your answer. Hand-written answer is **not** acceptable.]

[Replace YourName and L3SID with your name and last three digit of your student ID, respectively.]

1. (20 pts) Given the following program. At the end of the program execution, what is the **total** number of processes (**including the initial parent process**)? And what is the value of `i` before **each** process terminates? Use a software to draw the space-time diagram (similar to the ones for `fork2` and `fork3` on slides) to justify your answer. **Hand-drawing receives no credit.**

```
int main() {
    int i = 1;
    if (fork() == 0) {
        ++i;
        if (fork() == 0)
            i = 4;
        else {
            ++i;
            fork();
        }
    } else {
        i = 7;
        if (fork() != 0)
            ++i;
    }
    if (i > 5) {
        fork();
        i = i + 2;
    }
    return 0;
}
```

2. (80 pts) [programming question]

(50 pts) 2.1 Design a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process. HW1 can be completed on Linux or Linux VM.

A shell interface gives the user a prompt, after which the next command is entered. The format of the prompt is

FirstName-L3SID>

where L3SID is the last three digits of your student ID. Assuming your first name is `demo` and L3SID is `123`, the example below illustrates the prompt (`demo-123>`) and the user's next command: `cat prog.c` (which displays the file `prog.c` on the terminal using the UNIX `cat` command).

```
demo-123> cat prog.c
```

The shell process first prints the prompt, reads what the user enters on the command line (in the above case, `cat prog.c`), and then creates a separate child process that performs the command. Unless otherwise specified, the shell (parent) process waits for the child (executed in the **foreground**) to exit before printing the next prompt and continuing. However, UNIX shells typically also allow the child process to run in the **background**, or concurrently. To accomplish this, we add an ampersand (`&`) at the end of the command. Thus, if we rewrite the above command as

```
demo-123> cat prog.c &
```

the shell (parent) and child processes will run concurrently. In both cases the separate child process is created using the `fork()` system call, and the user's command is executed by using one of the system calls in the `exec()` family. For any command running in the background, the shell (parent) process does **not** wait for the completion of the child process; the shell prints the prompt and reads the next command immediately.

A C program that provides the general operations of a command-line shell is as follows (you can copy this code).

```
#include <stdio.h>
#define MAXLINE 80 /* The maximum length of a command line */
int main(void)
{
    char *args[MAXLINE/2 + 1]; /* command line with max 40 arguments */
    int should_run = 1; /* flag to determine when to exit program */
    printf("CS149 Shell from FirstName LastName\n"); /* replace w/ name */
    while (should_run) {
        printf("FirstName-L3SID>"); /* prompt- replace FirstName and L3SID */
        fflush(stdout);
        /* After reading user input, the steps are:
        * (1) if the command is exit, set should_run to 0 to exit the while loop
        * (2) fork a child process using fork()
        * (3) the child process will invoke execvp()
        * (4) if the command line included &, parent will NOT invoke wait()
        */
    }
    return 0;
}
```

The `main()` function presents the prompt and outlines the steps to be taken after input from the user has been read. The `main()` function continually loops as long as `should_run` equals 1; when the user enters `exit` at the prompt, your shell will set `should_run` to 0 and terminate.

You should modify the `main()` function so that a child process is forked and executes the command specified by the user. This will require parsing what the user has entered into separate tokens and storing the tokens in an array of character strings `args`. For example, if the user enters the command "`ps -af`" at the prompt, the values stored in the `args` array are:

```
args[0] = "ps"
args[1] = "-af"
args[2] = NULL
```

This `args` array will be passed to the `execvp()` function, which has the following prototype:

```
execvp(char *command, char *params[]);
```

Here, `command` represents the command to be performed and `params` stores the parameters to this command. For Q2, the `execvp()` function should be invoked as `execvp(args[0], args)`. Be sure to check whether the user included an `&` at the tail end of command line to determine whether or not the shell (parent) process is to wait for the child to exit. The character `&`, if it exists at the tail end of a command line, is only meaningful to the shell and is **not** an argument of the command.

When `fork()` or `execvp()` failed, the shell should print out error messages, output the prompt and accept the next command from user.

The shell in Q2 does not need to support any additional functions such as pipe, and I/O redirection.

To read a line from the terminal (i.e., `stdin`), you may utilize `read(2)`, `getline(3)`, `fgets(3)` on Linux, or any other APIs. To extract tokens from a string, you may use `strtok(3)`, `strtok_r(3)`, or any other APIs, or write your own parsing routine.

Note: your shell is **NOT** allowed to hard-code the exact commands (date, ls, ps, etc.) to be executed. Your shell **must** parse the command line, whatever that command line is, and then execute the command specified.

Make sure for each invocation the program always prints out “CS149 Shell from ...” only once. Screenshots must include “CS149 Shell from ...” from the program. Follow the format for the prompt which includes FirstName and L3SID.

Compile your program with “gcc -o shell shell.c”. You can execute the program with “./shell”.

(25 pts) 2.2 After you are done with the simple shell,

a. (15 pts) Execute the followings in a “terminal” on Linux VM:

- In a terminal running a **real** shell, bring up your shell: `./shell`
- Within your shell, invalid command (print error msg): `abc`
- Within your shell, empty command line (move on to next line):
- Within your shell, list the directory /etc: `ls -c /etc`
- Within your shell, get current date: `date`
- Within your shell, sleep 750 seconds in the background: `sleep 750 &`
- Within your shell, show processes: `ps -f`
- Within your shell, exit your shell: `exit`
- Now back to a **real** shell, show processes: `ps -f`
- Take screenshot of the entire sequence (which includes “CS149 Shell from ...” from your program)

b. (10 pts) In a, the output from two executions of “ps -f” should include the same process “sleep 750” with identical PID but with different parent PID (PPID).

- (5 pts) Explain the reason, by starting with “The child process becomes a(n) _____ process”
- (5 pts) Identify which process is the new parent process of “sleep 750”. This can be done by invoking the following command in the **real** shell from a terminal (where NNN is the **new** PPID of “sleep 750”).

`ps -p NNN`

Include a screenshot of the execution of “ps -p NNN”.

(5 pts) 2.3 If your shell never waits for the termination of any user-entered command (regardless whether the tail of the command includes “&” or not), what would happen to these processes (corresponding to user-entered commands) after these processes call `exit()`, assuming your shell is still up and running? Your answer must start with “these processes become _____ process ...”

====

Submit the following files as **individual** files (do not zip them together):

- CS149_HW1_YourName_L3SID (.pdf, .doc, or .docx), which includes
 - Q1: answers and justification
 - Q2: embeds screenshots from step a and step b, and explanation in step b. Screenshots that are not readable, or screenshot without “CS149 Shell from ...” from the program, will receive 0 point for the entire homework.
- shell_YourName_L3SID.c **Ident your source code and include comments. Also follow the exact format for the prompt as specified. Any submission without the proper FirstName and L3SID in the prompt will receive 0 point for the entire homework.**

The ISA and/or instructor leave feedback to your homework as comments and/or **annotated** comment. To access **annotated** comments, click “view feedback” button. For details, see the following URL:

<https://guides.instructure.com/m/4212/l/352349-how-do-i-view-annotation-feedback-comments-from-my-instructor-directly-in-my-assignment-submission>