

## Lab1—UNIX Shell (Part I)

**Due:** 2:30 p.m., Friday, Jan 25<sup>th</sup>.

**Note:** to ensure submission process working, please check as soon as possible whether you have any problems with issuing the command: “submit c660ab lab0 <any-test-file-name>”.

**Group Size:** 1, which means you are required to finish this lab assignment by yourself.

**Goal:** This lab helps you understand the concept of processes, how to create, and how to terminate a process. In addition, you are expected to get familiar with system calls related to processes and file operations.

**Introduction:** As the first step of the project in Chapter 3, this lab assignment ask you to build a simple SHELL interface that accepts user commands, creates a child process, and executes the user commands in the child process. The SHELL interface provides users a prompt after which the enxt command is entered. The example below illustrates the prompt `sh>` and the user’s next command: `cat prog.c`. This command displays the file `prog.c` content on the terminal using the UNIX `cat` command.

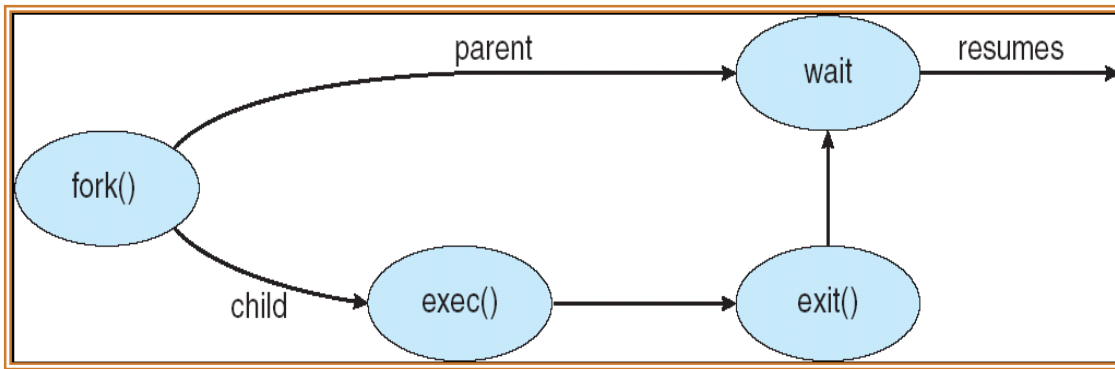
```
sh> cat prog.c
```

One technique for implementing a shell interface is to have the parent process first read what the user enters on the command line (i.e. `cat prog.c`), and then create a separate child process that performs the command. Unless otherwise specified, the parent process waits for the child to exit before continuing. This is similar in functionality to what is illustrated in Figure 1. However, UNIX shells typically also allow the child process to run in the background – or concurrently – as well by specifying the ampersand (`&`) at the end of the command. By rewriting the above command as

```
sh> cat prog.c &
```

the parent and child process now run concurrently.

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 more details about the system call, you can use the `man` command).



**Figure 1.** Process Creation & Termination

## A Simple Shell

A C program that provides the basic operations of a command line shell is supplied in the file `shell.c`, which you can copy from the following location "[www.cse.ohio-state.edu/~agrawal/660/shell.c](http://www.cse.ohio-state.edu/~agrawal/660/shell.c)". This program is composed of two functions: `main()` and `setup()`. The `setup()` function reads in the user's next command (which can be up to 80 characters), and then parses it into separate tokens that are used to fill the argument vector for the command to be executed. (If the command is to be run in the background, it will end with '&', and `setup()` will update the parameter background so the `main()` function can act accordingly. This program is terminated when the user enters <Control><D> and `setup()` then invokes `exit()`).

The `main()` function presents the prompt `COMMAND->` and then invokes `setup()`, which waits for the user to enter a command. The contents of the command entered by the user are loaded into the `args` array. For example, if the user enters `ls -l` at the `COMMAND->` prompt, `args[0]` will be set to the string `ls` and `args[1]` will be set to the string `-l`. (By "string", we mean a null-terminated, C-style string variable.)

```

#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 80

/** setup() reads in the next command line string stored in inputBuffer, separating it
into distinct tokens using whitespace as delimiters. setup() modifies the args
parameter so that it holds pointers to the null-terminated strings that are the tokens in
the most recent user command line as well as a NULL pointer, indicating the end of
the argument list, which comes after the string pointers that have been assigned to
args. */

void setup(char inputBuffer[], char *args[],int *background)
{
    /** full code available in the file shell.c */
}

int main(void)
{
    char inputBuffer[MAX_LINE]; /* buffer to hold the command entered */
    int background; /* equals 1 if a command is followed by '&' */
    char *args[MAX_LINE/+1]; /* command line arguments */

    while (1){
        background = 0;
        printf(" COMMAND->\n");
        setup(inputBuffer,args,&background); /* get next command */

        /* the steps are:
        (1) fork a child process using fork()
        (2) the child process will invoke execvp()
        (3) if background == 0, the parent will wait,
            otherwise returns to the setup() function. */
    }
}

```

**Figure 2.** Outline of shell.c

This lab assignment asks you to create a child process and execute the command entered by a user. To do this, you need to modify the *main()* function in `shell.c` so that upon returning from *setup()*, a child process is forked. After that, the child process executes the command specified by a user.

As noted above, the *setup()* function loads the contents of the *args* array with the command specified by the user. This *args* array will be passed to the *execvp()* function, which has the following interface:

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

where *command* represents the command to be performed and *params* stores the parameters to this command. You can find more information on *execvp()* by issuing the command “man *execvp*”. Note, you should check the value of *background* to determine if the parent process is to wait for the child to exit or not.

**Test:** We provide several test cases and readme in the file “[www.cse.ohio-state.edu/~agrawal/660/readme](http://www.cse.ohio-state.edu/~agrawal/660/readme)”, you need to follow the instructions in the file, and compare your results with the expected output in the file.

**Submission:** Put all your modification in the file “`shell.c`” and only submit that file using the following command “`submit c660ab lab1 shell.c`”. At the beginning of the file “`shell.c`”, you need to tell the grader your full name, how to compile your file, run your compiled program, and make sure your instructions are correct.

**In addition, please bring a print-out of your complete code, including runs on the test cases to the class on 25<sup>th</sup> January. This print-out must have your name and e-mail address.**