CSCI 347 - Computer Systems II
Winter 2018 Assignment 1, 100 points

Due: January 19, 2017

In class we will discuss a micro-shell. (Source code may be found on my web site, http://facultyweb.cs.wwu.edu/~phil/classes/w18/347/microshell.c.) I want you to extend this micro-shell to do simple argument processing in the shell. Specifically, I want you to do the following:

- Read the introduction to UNIX found on the class web page. Even if you already know UNIX. Especially notice the information on the program a2ps(1). I want what you turn in to me to be at least as nice as what a2ps produces. Your a2ps output may be two logical pages per physical page.

- Read the man pages on execve(2) and related functions (exec(3)). (Yes, you will need to use a different exec... function than what is currently used in the microshell.)

- Read the man page on malloc and if needed, consult a C programming text on how to use malloc(3) and the C sizeof function. Also, read about free(3).

- Create a new private project on gitlab.cs.wwu.edu with the name of “csci347_w18” (without the quotes). Give user “phil” access to your ush project.

- Clone your new project. First, add a README.md that has your name and quarter in it. Next, a add directory named “ush” to your project. All ush files must be in this directory. Copy the microshell.c to your project’s ush directory and name it ush.c. Commit and push both files.

- Write a function “arg_parse” that must have the prototype: char** arg_parse (char *line, int *argcptr)!. The “line” parameter points to the line (character array) that contains the command to be processed. The “argcptr” parameter points to an integer variable where this function must put the number of arguments found in the “line”. The return value must be a pointer to a malloced area that points into the line parameter and matches what is needed by execve for the “argv” parameter. The malloced area should be the exact size needed,
not too small and not too big. You should call malloc(3) only once. Make sure you check for malloc(3) errors.

Do your processing by breaking up the line on spaces. Do not do more complex processing at this time. You should not copy any characters out of the line parameter. You only put zero characters in the line parameter and set pointers to characters in the line parameter in your malloced area. Notice that two or more consecutive spaces is equivalent to a single space. For example, a call to arg_parse with the parameter line

```plaintext
mycommand   arg1   nextarg arg3 not-arg-5-but-4
```

should return a pointer to an “array” of 6 elements. The first element (index 0) should point to the character ‘m’ in mycommand, the second element should point to the ‘a’ of arg1, and so forth. The last element should be a NULL pointer. The characters “mycommand” are not copied to any other storage. The space just after “mycommand” is turned into a zero character so “mycommand” is a zero terminated string. Also, leading and trailing spaces should be ignored! (Before the command name and after the last argument.) Note: Do not use strtok(3) from the string library.

- Modify the micro-shell function processline() to use arg_parse to run commands with arguments. Call arg_parse before the fork() call and if there are no parameters on the line, do not fork(), but just return from processline() without doing more. Do not forget to free the pointer returned by arg_parse() in the parent shell process. You will also need to modify the child code in processline() to call a different exec function. Do not use execve.

- Make sure you have no warnings when compiled with “gcc -Wall” on Ubuntu as provided by the department in the labs. (Remember, you have remote access to the linux-01 to linux-12 machines. Domain is cs.wwu.edu. Some kind of ssh client is required to remote access these machines.)

- When you are completed with your work for this assignment, create a branch named “a1” in your git repo. This branch will be used to check out and grade your assignment 1. You should not have any commits on your “a1” branch. All of your work should be on branch “master”. (Note: If you accidentally get commits on your “a1” branch, you will
want to merge the “a1” branch back into “master” to get changes on “a1” back on “master” where you are to work. This work-flow is different than many companies and open source projects want when using git. A very typical work-flow is for a person to do their work on a branch and get it working. Then they merge a working branch back to “master”. In that way, nobody is working on “master” and committing directly to master. All work is merged from a branch.)

After you create the branch, you will need to push that branch to gitlab. You do this using the command:

```
git push --set-upstream origin a1
```

- **Turn in** a printed copy of your code and a “typescript” of a compile and run of your shell. Use script(1) to make the “typescript”. (Read the man page.) Remember to use a2ps or something better. I also want a cover page that includes your name, “CS 347 Quarter year” and the assignment number. You need to do this for ALL your assignments this quarter.

**Note:** These are the only modifications you are to do to this shell. You must start with my code. Any other kinds of modifications will cause you to lose points. (e.g. Changing the prompt.)

**Note 2:** I will be sending e-mail from time to time to the class. I will be using your student e-mail address. Make sure that you either read your student e-mail often or have it forwarded to your preferred e-mail address.

**Note 3:** You should have an account on the CS machines. You need to try to login before class on January 12. If you can’t login, send me email and I’ll make sure you can login. Commit times on your gitlab repository will be used to determine if your assignment is late. ANY commit after the due date and time on your assignment branch will make your assignment late. (Commit on master after the due date and time will be considered as part of the next assignment.)