**Operating Systems**

**Homework – a better shell**

Start with the shell, smsh, that you created in the lab, and extend it so that it recognizes and handles redirecting output, using both ‘>’ and ‘>>’.

* > f

means overwrite the file f with the output of the preceding command, creating f if necessary

* >> f

means append the output of the preceding command to the end of file f, creating f if necessary

It should also correctly handle all the functionality we discussed in class:

* Present the prompt, “smsh% “
* Accept a command line of maximum length 254 chars
* Behave well if the user enters a non-existent or un-executable program, printing, “**No such program: *progname***”where *progname* is whatever command the user entered.
* It should correctly handle the user entering any number of command-line arguments, up to a max of 100.
* It should be able to run a program in the background, accepting an ampersand ‘&’ as either the very last token, or as the last character of the very last token. For example, both of the following should run the cat program in the background:

smsh% cat file.c&

smsh% cat file.c &

To solve the output redirection problem, you will need to look back at the system call **dup2** that we discussed earlier. Recall that a child process is a copy of its parent, with a copy of the parent’s file descriptor table. When exec is called in a process, the process’ code (text) is replaced with the new program’s code, and the program counter is reset to the beginning of the code. However, *the file descriptor table is not replaced* by the call to exec.

Your shell is required to recognize output redirection only if

* the ‘>’ or ‘>>’ is an individual token (delimited on both sides by spaces)
* it appears as the next-to-last arg in an arglist, where the last arg is the filename
* or the second-from-last arg in the arglist if the very last arg is the & (see examples below)

For example, these are commands your shell must recognize and try to process as output redirection (whether they make sense or not):

smsh% cat file.c > ofile

smsh% cat file.c >> ofile

smsh% grep word infile > ofile

smsh% cat > infile > outfile

smsh% cat infile > >>

smsh% cat > file.c &

But the following do not have to be recognized as output redirection, or error-checked:

smsh% cat file.c> ofile

smsh% cat file.c >>ofile

smsh% cat > ofile infile

smsh% cat file.c > ofile >

The 2 maximum lengths given allow you to create fixed length arrays. For example, in my original code, I have defined constant CMDLEN. You can set it to 256, and you can set your array of args’ MAXARGS to 100. Your code does not have to do error handling related to these maximums.

Your code must not use any stdio.h functions except for printf, but may freely use string.h functions.

Your shell must meet the requirements of the smsh lab assignment, which are reiterated above. While I understand you may have worked in a group on the lab, and so you may have the same solution as your lab partner(s) on that work, you were supposed to type, compile and test your own copy of the code. These extensions to smsh must be individual work. Try your best to do the work using my guidance above, or by using piazza to seek clarifications from me and your classmates. If you rely too much on asking for help from your friends and the web, you will not exercise and understand the concepts and will not be able to pass a test on the topic.