Project 2 Phase 1: Standalone Shell-Shell

The usual OS project is to build a simple shell, which is the user interface to the operating system. Instead, we will build a shell on shell, which takes user commands and pass to shell for processing. We call this program Shsh and the executable name should be "shsh.exe".

Also, in Unix, everything in execution is a process, including those OS programs. After the system boots up, the first process is created, which forks many other processes to offer operating system and other system services. When you login, a process executing the shell program is created. In the shell process, when you execute a program, a new process is created. In Unix, this process creation operation is also offered as a system call to the users. We will use "fork" system call to create processes and use other system calls to enable communication between the processes we have created.

In the Shsh program, you need to process 3 types of commands, "cmd", "pipe" and "exit". We call these the shshcmd. You first print the command prompt "cmd> " and then read the command from stdin and process it.

The cmd command starts with the keyword "cmd" and followed by any shell command. You need to retrieve the shell command (let's call it shcmd) following cmd and use the system call "system()" to execute it. For example, if Shsh reads the following shell command

cmd> cmd cd testdir; ./a.out
cmd>

it should execute system("cd testdir; ./a.out"). Note that the "system()" system call will result in the creation of a new subprocess executing the shell program. Note that you have to distinguish which commands will only cause effects internal to the process and which ones will result in side effects external to the process. For example, if you execute cd in process p, which changes p's cwd, but the parent Shsh process will not get the effect of cd. On the other hand, if a subprocess creates a file, which is an external effect, the parent process of course will be able to see the file. Thus, we need to assume that the commands issued to Shsh are independent, as though being issued in different sessions. The pipe command starts from keyword "pipe", followed by a sequence of shell commands (shcmd) separated by ";". An example pipe command is given below. Your Shsh should fork one subprocess for each shell command in the pipe command and pass the output from one process to the next using the pipes created by the "pipe()" system call. You need to create and use the pipes carefully to achieve correct data flow. In order to force the "system()" system call to use the proper pipes as input and output, you need to map stdin and stdout to the pipes by using dup2() system call for the corresponding subprocess.

cmd> pipe cat proj2.txt; sed -e "s/fork/create-process/g"; cat > out
cmd>

The exit command simply asks Shsh to terminate. When Shsh starts, it should obtain the process ids of its own and its parent's using the getpid() and getppid() system calls, respectively. Then, it prints "Shsh process $\boldsymbol{pid}$ has been created by process $\boldsymbol{ppid}$". When it terminates, it prints "Shsh process $\boldsymbol{pid}$ terminates". (Italic font indicates that the to-beprinted item is a variable.) When receiving a shshcmd, Shsh should print out "Shsh processing $\boldsymbol{shshcmd}$", where *shshcmd* is the entire shshcmd command Shsh has received. For each pipe command, Shsh should print out the entire configuration in multiple lines. Each line should be "Shsh forked $\boldsymbol{pid}$ for $\boldsymbol{shcmd}$, with in-pipe $\boldsymbol{pipedesc}$ and out-pipe $\boldsymbol{pipedesc}$", where the two *pipedesc*s are different and they are the file descriptors of the input and output pipes for the corresponding process.