Lab 5: Reading Material

Implementing a Command Interpreter (Shell)

You should read the introductory material from the task descriptions as part of the reading material.
Download the attached code in the task file and learn how to use it.

Thoroughly understand the following:

1. All the material from previous labs related to the C language, especially using pointers and structures in C.
2. Read the man pages and understand how to use the following system calls and library functions: getcwd, fork, execv, execvp, perror, waitpid, pipe, dup, fopen, close.

Basic introduction to the notion of fork can be found here and here.

Attached code documentation

LineParser:

This package supports parsing of a given string to a structure which holds all the necessary data for a shell program.
For instance, parsing the string **"cat file.c > output.txt &"** results in a cmdLine struct, consisting of arguments = {"cat", "file.c"}, outputRedirect = "output.txt", blocking = 0 etc.

```
typedef struct cmdLine
{
  char * const arguments[MAX_ARGUMENTS];
  int argCount;
  char const *inputRedirect;
  char const *outputRedirect;
  char blocking;
  int idx;
  struct cmdLine* next;
} cmdLine;
```

Included functions:

1. **cmdLine* parseCmdLines(const char *strLine)**
   Returns a parsed structure cmdLine from a given strLine string, NULL when there was

nothing to parse. If the strLine indicates a pipeline (e.g. "ls | grep x"), the function will return a linked list of cmdLine structures, as indicated by the **next** field.

2. **void freeCmdLines(cmdLine *pCmdLine)**
   Releases the memory that was allocated to accomodate the linked list of cmdLines, starting with the head of the list pCmdLine.

3. **int replaceCmdArg(cmdLine *pCmdLine, int num, const char *newString)**
   Replaces the argument with index *num* in the arguments field of pCmdLine with a given string.
   If successful returns 1, otherwise - returns 0.

Job Control

Processes belonging to a single command are referred to as a job or a process group. For example, `ls|wc -l` both the process that runs `ls` and the process that runs `wc -l` of this command belong to the same job/process group. Only one process group can run in the foreground and control the shell, the terminal's process group is set using tcsetpgrp. The rest of the process groups run in the background. Deciding which jobs run in the background and which run in the foreground is called job control. You can read more about job control in the following links: link1 and link2(chapter 9.8). An example of job control in Linux.

To save the shell's terminal attribute and restore it back to it's state after running a process in the foreground, we use tcgetattr(3) and tcsetattr(3)respectively. You can read about terminal attributes/modes here.

Signals

Signals are a used to send asynchronous events to a program such as ctrl-c, and ctrl-z. You can read more about signals here and here.

MAN: Relevant Functions and Utilities

fork(2), wait(2), waitpid(2), _exit, exit(3), getpid(2), setpgid(2), tcsetpgrp(3), tcgetpgrp(3), tcgetattr(3), tcsetattr(3), kill(2), signal(2).