Shell Pseudo-Code

Bradley Juma

Monday - February 18, 2019

Introduction

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <errno.h> // perrors
#include <dirent.h>

#define TOK_BUFSIZE 128
#define TOK_DELIM " \t\r\n\a”

void my_cd(char **);       //change directory
void my_clr(char **);      //clear the screen
void my_dir(char **);      //display contents of target directory
void my_environ(char **);  //display environment strings
void my_echo(char **);     //repeat input back to stdout
void my_pause(char **);    //busy-wait
void my_helper(char **);   //display readme with more filter
int  my_quit(char **);     //quit

// Array of characters to hold builtin commands.
char *builtin_cmd[] = {
    "cd",
    "clr",
    "echo",
    "environ",
    "export",
    "help",
    "pause",
    "quit"
};

// Array of pointer to function that takes char ** as input and
return int
int (*builtin_func[]) (char **) = {
    &sh_cd,
    &sh_clr,
    &sh_echo,
    &sh_environ,
    &sh_export,
    &sh_help,
    &sh_pause,
```

```
    &sh_quit
};


int main(){


    if (Pipe is detected) {
        read or parse then pipe or dup2 will be used.
    }
    launch();
}
```

- Function - split_line
  - INPUT - char *line, char **ptr
  - OUTPUT - Character containing each arguments.
    - This function will split-up the command line into arguments so it knows which builtin ruction it'll be using.

****ASSUME****

```
void split_line(char *line, char **ptr) {
    char *tokens[100];
    char *token;
    int pos = 0;

    char out [100];
    int i = 0;

    while (1) {
        out[i] = line[i];
        if (out[i] == '\0' || out[i] == '\n') {
            out[i] = '\0';
            break;
        }
        i++;
    }

    token = strtok(out, TOK_DELIM);
    //printf("Bye\n");
    while (token != NULL) {
        //printf("%s %d \n",token, pos);
        tokens[pos++] = strdup(token);

        token = strtok(NULL, TOK_DELIM);
    }
    //printf("%s %d \n",token, pos);
    int j;
```

```
    for (j  = 0; j<pos; j++) {
        ptr[j] = tokens[j];
    }
    //tokens[p]
    //tokens[pos] = NULL;

    return;
}
```

- Function -my_clear
  - INPUT - char **argv
  - RETURN - void
    - System function does the work for you to clear stdout screen.

```
void my_clr(char **args) {
    system("clear");
    return;
}
```

- Function -my_pause
  - INPUT - char **argv
  - RETURN - void
    - System function does the work for you to clear stdout screen.

```
void my_pause(char **argv){
    printf("press 'Enter' to continue");
    while (getchar() != '\n');

    return;

}
```

- Function -my_quit
  - INPUT - char **argv
  - RETURN - 0 == Success rate (END of the program)
    - Pauses the operation for the shell until \n is detected.

```
// Quits the shell
int my_quit(char **argv){
    exit(EXIT_FAILURE);
}
```

- Function -my_echo
  - INPUT - char **argv
  - RETURN - void
    - Prints all the arguments from the command back to stdout.

```
void my_echo(char **argv){
    if (argv == NULL)
    fprintf(stdout, "\n");
     for loop agrv not equal to null
```

```
          printf("%s\n", argv[i]);
    return;
}
```

- Function -my_environ
  - INPUT - char **argue
  - OUTPUT - void - Does it's job.
    - Prints the environment variable to stdout.

```
void my_environ(char **argv){

    if (redirected) {
        Apply changed to the FD for stdout
            if (0) {
                perror("Error");

            }
            else{

                duplicate the FD --> stdout //1
            }
    }
    printf("Dir,Pwd,Shell");

    return;
}
```

- Function - my_dir
  - INPUT - char **argv
  - OUTPUT - void
    - Lists all the current items in working directory just like (ls) function.

```
void my_dir(char **argv){
    DIR *dp; // '#include <dirent.h>'; declaration of 'DIR' must be
imported from module 'Darwin.POSIX.dirent'
    if (redirections) {
        open file for appending // adding
            }else{
                open file for trunc
            }
    if fail
        perror
    else
        dup file descriptor to redirect output

        while dir reaches = readdir()
            print to dir;
}
```

- Function - my_helper
  - INPUT - char **argv
  - OUTPUT - void
    - Lists all the current items in working directory just like (ls) function.

```
void my_helper(char **argv){
    FILE *file pointer;
    char array for read me.
    if (redirections) {
        open file for appending // adding
            }else{
                open file for trunc
                    }
    if fail
        perror
        else
            dup file descriptor to redirect output

            while buffer reaches EOF
                print to stdout;
}
```