CSC415[01] – Homework 7 – Extended Shell

By: Aleksandr Kibis

12/8/2014

This assignment is an addition to Homework 3 which had us implement a POSIX and Windows shells. Surprisingly, this last project was tremendously complicated, to the point where I wasn't even able to complete the Windows version after a week of working on the assignment. Part of the reason for having such a tough time implementing the new functionality was because I had to completely rebuild the entire program from the ground up. For some reason, the way I was tokenizing my user input in the initial program completely refused to behave properly when more complicated functions were added. Standard output and error streams were not being written to files correctly, and commands were truncated. After fiddling around with the code, I came to the realization that it was due to the way strtok formats text (it adds a null terminating character after every token). This forced my commands to be cut abruptly. The only way to circumvent this problem was to change my approach completely by using pointers and strstr which returns the index of the first occurrence of a string. Using this approach, user input was then split into redirection operators and input.

Overall it was almost like writing 3 different programs/algorithms. It could be split into background processes, stream redirection, and piping. Background processes were almost identical to the original program, with the exception that the parent did not wait anymore. Stream redirection has the dup2 function at its heart which copies file descriptors from one and points it to the other. Finally the pipe was where things got really tricky. Here we have 2 processes depending on each other. One command is run and the output is written to a buffer which is then redirected as the standard input to the right hand side of the pipe. The rest finishes as a standard command. I wish I had more time to work on this project, but eventually had to call it quits because there is just so much other work that has to be done. Thankfully, I finished the Linux version since that is where I focus my studies in general.

CODE:

Posix:

```
/*
* File:   posix_shell_extended.c
* Author: Aleksandr Kibis
*
* Compile: gcc -o run posix_shell_extended.c
* Run: ./run
*
* This program runs a basic Linux shell with added functionality
*
* Added functions:
* redirect standard input, output, error
* piping
*
* Created on September 28, 2014, 7:40 PM
*/
```

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

#define EXIT_SUCCESS 0
#define EXIT_FAILURE 1
#define BUFFER_SIZE 2048
#define WORD_LENGTH 1024

int main() {
	char buf[BUFFER_SIZE];

	while (1) {
		// reset
		// stdout
		dup2(1, 1);

		// stderr
		dup2(2, 2);

		// stdin
		dup2(0, 0);

		// grab user input
		printf("netdom> ");
		fgets(buf, BUFFER_SIZE, stdin);

		// strip new line char
		buf[strlen(buf) - 1] = '\0';

		// exit on keyword
		if (strcmp(buf, "exit") == 0) {
			exit(EXIT_SUCCESS);
		}

		// shell function flags
		int isBackgroundProc = 0;
		int stdout_redir = 0;
		int stderr_redir = 0;
		int stdin_redir = 0;
		int isPipe = 0;


		char* tokenLocation;

		// FILE HANDLERS

		// single command
		int fd;

		// pipe
		int pipe_fd[2];
		// END HANDLERS
		// run background process
		if (strstr(buf, "&")) {
			tokenLocation = strstr(buf, "&");
			*tokenLocation = 0;
			isBackgroundProc = 1;
		}

		// stdout truncate
		if (strstr(buf, " > ")) {
			tokenLocation = strstr(buf, " > ");
			char* file = tokenLocation + 3;

			// create new file if it doesn't exist, truncate it, write-only
			fd = open(file, O_CREAT | O_TRUNC | O_WRONLY, S_IRWXU);
			if (fd < 0) {
				perror("Error opening file!\n");
				exit(EXIT_FAILURE);
			}
			stdout_redir = 1;
			*tokenLocation = 0;
```

```
}

// stdout append
if (strstr(buf, " >> ")) {
          tokenLocation = strstr(buf, " >> ");
          char* file = tokenLocation + 4;

          // create new file if it doesn't exist, append it, write-only
          fd = open(file, O_CREAT | O_APPEND | O_WRONLY, S_IRWXU);
          if (fd < 0) {
                    perror("Error opening file!\n");
                    exit(EXIT_FAILURE);
          }
          stdout_redir = 1;
          *tokenLocation = 0;
}

// stderr truncate
if (strstr(buf, " 2> ")) {
          tokenLocation = strstr(buf, " 2> ");
          char* file = tokenLocation + 4;

          // create new file if it doesn't exist, truncate it, write-only
          fd = open(file, O_CREAT | O_TRUNC | O_WRONLY, S_IRWXU);
          if (fd < 0) {
                    perror("Error opening file!\n");
                    exit(EXIT_FAILURE);
          }
          stderr_redir = 1;
          *tokenLocation = 0;
}

// stderr append
if (strstr(buf, " 2>> ")) {
          tokenLocation = strstr(buf, " 2>> ");
          char* file = tokenLocation + 5;

          // create new file if it doesn't exist, append it, write-only
          fd = open(file, O_CREAT | O_APPEND | O_WRONLY, S_IRWXU);
          if (fd < 0) {
                    perror("Error opening file!\n");
                    exit(EXIT_FAILURE);
          }
          stderr_redir = 1;
          *tokenLocation = 0;
}

// stdin
if (strstr(buf, " < ")) {
          tokenLocation = strstr(buf, " < ");
          char* file = tokenLocation + 3;

          // open file read-only
          fd = open(file, O_RDONLY);
          if (fd < 0) {
                    perror("Error opening file!\n");
                    exit(EXIT_FAILURE);
          }
          stdin_redir = 1;
          *tokenLocation = 0;
}


/*START PIPE*/

/*
* Pipe creates two processes which handle their own data but are connected
* via input/output. Have to handle memory individually
*/
int pipeMade = 0;
char* threadName;


// FIRST CHUNK
char* myargv[BUFFER_SIZE];
char temp_argv[BUFFER_SIZE][WORD_LENGTH];
int myargc = 0;
```

```
if (strstr(buf, " | ")) {
	isPipe = 1;
	pipeMade = 1;

	tokenLocation = strstr(buf, " | ");
	//printf("check: %d\n", tokenLocation);

	pipe(pipe_fd);
	threadName = tokenLocation + 3;

	*tokenLocation = 0;

	// tokenize input
	int j = 0;
	while (threadName[j]) {
		int k = 0;

		// delimeters: '\0' and ' '
		while (threadName[j] && threadName[j] != ' ') {
			temp_argv[myargc][k] = threadName[j];
			j++;
			k++;
		}
		temp_argv[myargc][k] = 0;

		myargv[myargc] = temp_argv[myargc];
		myargc++;

		//printf("myargv: %s\n", myargv);

		if (threadName[j])
			j++;
	}

	// end of input
	myargv[myargc] = NULL;
}

// SECOND CHUNK

char* myargv2[BUFFER_SIZE];
char temp_argv2[BUFFER_SIZE][WORD_LENGTH];

int myargc2 = 0;
int j = 0;

// delimeters: '\0' and ' '
while (buf[j]) {
	int k = 0;
	while (buf[j] && buf[j] != ' ') {
		temp_argv2[myargc2][k] = buf[j];
		j++;
		k++;

	}
	temp_argv2[myargc2][k] = 0;

	myargv2[myargc2] = temp_argv2[myargc2];
	myargc2++;

	//printf("myargv2: %s\n", myargv2);

	if (buf[j]) j++;
}

// end of input
myargv2[myargc2] = NULL;

// create child process
pid_t cpid;

cpid = fork();
if (cpid == 0) {
	if (stdout_redir) {
		if (dup2(fd, 1) == -1) {
			perror("Error redirecting stdout!\n");
			exit(EXIT_FAILURE);
		}
	}
```

```
                        if (stderr_redir) {
                                if (dup2(fd, 2) == -1) {
                                        perror("Error redirecting stderr!\n");
                                        exit(EXIT_FAILURE);
                                }
                        }

                        if (stdin_redir) {
                                if (dup2(fd, 0) == -1) {
                                        perror("Error redirecting stdin!\n");
                                        exit(EXIT_FAILURE);
                                }
                        }

                        if (isPipe) {
                                if (dup2(pipe_fd[1], 1) == -1) {
                                        perror("Error creating pipe!\n");
                                        exit(EXIT_FAILURE);
                                }
                        }

                        execvp(myargv2[0], myargv2);
                        if (isPipe) {
                                close(pipe_fd[1]);
                        }
                        exit(EXIT_FAILURE);

                }
                else {
                        // parent
                        if (cpid < 0) {
                                perror("Error sporking process o_O\n");
                                exit(EXIT_FAILURE);
                        }
                        else if (!isBackgroundProc) {
                                waitpid(cpid, 0, 0);
                        }
                        if (isPipe) {
                                close(pipe_fd[1]);
                        }
                }
                if (pipeMade) {
                        cpid = fork();

                        // is child process
                        if (cpid == 0) {
                                if (dup2(pipe_fd[0], fileno(stdin)) == -1) {
                                        perror("Error redirecting stdin!\n");
                                        exit(EXIT_FAILURE);
                                }
                                execvp(myargv[0], myargv);
                                exit(EXIT_FAILURE);

                                // else parent process
                        }
                        else {
                                if (cpid < 0) {
                                        printf("Error sporking parent o_O\n");
                                        exit(EXIT_FAILURE);
                                }
                                else {
                                        waitpid(cpid, 0, 0);
                                        close(pipe_fd[0]);

                                }
                        }
                }

                if (fd > 0) {
                        close(fd);
                }
        }

        return EXIT_SUCCESS;
}
```

OUTPUT:

Background Process

```
netdom> firefox &

netdom>

(process:58193): GLib-CRITICAL **: g_slice_set_config: assertion 'sys_page_size == 0' failed

(firefox:58193): GLib-GObject-WARNING **: Attempt to add property GnomeProgram::sm-connect after class was initialised

(firefox:58193): GLib-GObject-WARNING **: Attempt to add property GnomeProgram::show-crash-dialog after class was initialised

(firefox:58193): GLib-GObject-WARNING **: Attempt to add property GnomeProgram::display after class was initialised

(firefox:58193): GLib-GObject-WARNING **: Attempt to add property GnomeProgram::default-icon after class was initialised

netdom>
```

Redirect STDOUT

```
netdom> ls build > test

netdom> cat test

Debug

netdom> ls -l build >> test

netdom> cat test

Debug

total 4

drwxr-xr-x 3 netdom netdom 4096 Dec  7 21:45 Debug

netdom>
```

Redirect STDERR

```
netdom> ls asdkgnkasdgn 2> error.log

netdom> cat error.log

ls: cannot access asdkgnkasdgn: No such file or directory

netdom> ls kjnsadgiohw 2>> error.log

netdom> cat error.log
```

ls: cannot access asdkgnkasdgn: No such file or directory

ls: cannot access kjnsadgiohw: No such file or directory

netdom>

Redirect STDIN

netdom> cat test

here is some test text.

and another line below it.

plus one more for good measure.

netdom> grep below < test

and another line below it.

netdom>

Pipe

netdom> df

Filesystem 1K-blocks Used Available Use% Mounted on

/dev/sda1 16381864 12680840 2845832 82% /

none 4 0 4 0% /sys/fs/cgroup

udev 2005036 4 2005032 1% /dev

tmpfs 404108 1540 402568 1% /run

none 5120 0 5120 0% /run/lock

none 2020524 39488 1981036 2% /run/shm

none 102400 24 102376 1% /run/user

netdom> df | grep tmpfs

tmpfs 404108 1540 402568 1% /run

netdom>