# CSC415 – Windows/Linux Shell

## Aleksandr Kibis

## 9/30/2014

This project was the hardest one assigned to us yet, but very useful since I learned quite a lot of different functions. As with the previous projects, the Linux version was much easier to implement than the Windows counterpart. However, this time around, the Windows version was actually less complex. In the Linux version, an infinite loop was created that created a child process and waited for it to finish executing before spawning another. Various if-statements were used in order to check for process id and whether the user issued the “exit” command or not. The biggest difference that I noticed is that the users input had to be tokenized manually for POSIX, in Win32, the CreateProcess functions took care of this automatically. This difference also led to a need for a two-dimensional array in POSIX.

While the Windows version was eventually less complicated, it took me a lot longer to figure out exactly how to pass the correct arguments to CreateProcess. This is something that really bothers me about the Win32 API. There are just so many different variable types and most are not capable of being cast into anything simple. For example, in order to properly pass arguments to CreateProcess, I had to send it a TCHAR array instead of a char array. Char\* cannot be cast to TCHAR, instead the MultiByteToWideChar function has to be used. It’s nice that CreateProcess takes care of everything but chasing types in Win32 is like going down a rabbit hole.

## POSIX

/\*

\* File: linux\_shell.c

\* Author: netdom

\*

\* Created on September 28, 2014, 7:40 PM

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define EXIT\_SUCCESS 0

#define EXIT\_FAILURE 1

#define BUFFER\_SIZE 2048

#define MAX\_ARGS 20

/\*

\*

\*/

int main() {

char input[BUFFER\_SIZE]; // single string of user args

const char s[2] = "\n "; // string delimiter

char\* token; // single token from string

int myargc = 0; // number of args

char\*\* myargv;

pid\_t cpid = 0;

int status = 0, error;

while (1) {

// reset argument memory

myargv = (char\*\*)calloc(MAX\_ARGS, sizeof(char\*));

myargc = 0;

// start shell gui and get user input

printf("netdom> ");

fgets(input, BUFFER\_SIZE, stdin);

token = strtok(input, s); // tokenize user input

// fork process for each new input

while (token != NULL) {

myargv[myargc++] = token;

token = strtok(NULL, s);

if (strcmp(myargv[0], "exit") == 0) { // exit shell upon keyword

printf("<3\n");

exit(EXIT\_SUCCESS);

}

if ((cpid = fork()) < 0) {

perror("Error sporking process o\_O\n");

exit(EXIT\_FAILURE);

}

else if (cpid == 0) { // limit to only 1 shell process

if (error = execvp(\*myargv, myargv) < 0) {

perror("EXEC ERROR\n");

exit(EXIT\_FAILURE);

}

} else {

while (wait(&status) != cpid); // have parent wait for child to finish

}

//printf("myargv[%d]: %s\n", myargc - 1, myargv[myargc - 1]);

}

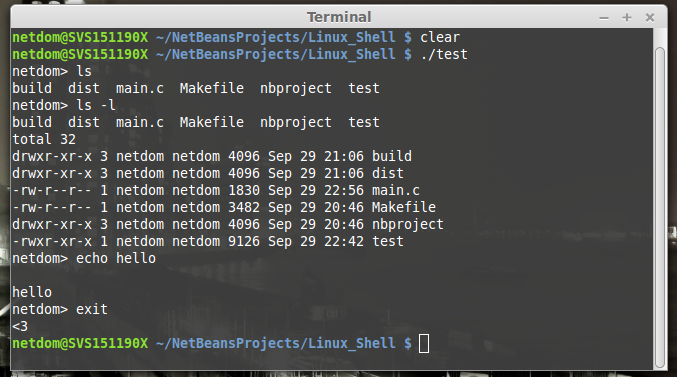
free(myargv); // free memory

}

return (EXIT\_SUCCESS);

}

## Output



## Win32

/\*

File: windows\_shell.c

Author: Aleksandr Kibis

Date: 9/29/2014

This program mimicks the windows command prompt

\*/

#include <Windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <tchar.h>

#define EXIT\_SUCCESS 0

#define EXIT\_FAILURE 1

#define BUFFER\_SIZE 2048

int main()

{

char\* input;

TCHAR command[BUFFER\_SIZE];

STARTUPINFO si;

PROCESS\_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );

si.cb = sizeof(si);

ZeroMemory( &pi, sizeof(pi) );

while(1){

// allocate memory and add cmd preprocessor

char args[BUFFER\_SIZE + 4] = "/c ";

input = (char\*)calloc(BUFFER\_SIZE, sizeof(char));

printf("netdom> ");

// get user input and concatinate with preprocessor

fgets(input, BUFFER\_SIZE, stdin);

if (strcmp(input, "exit\n") == 0) { // exit shell upon keyword

printf("<3\n");

exit(EXIT\_SUCCESS);

} strcat(args, input);

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

// convert args from char\* to TCHAR\*

MultiByteToWideChar(CP\_UTF8, 0, args, -1, command, BUFFER\_SIZE + 4);

if( !CreateProcess(TEXT("C:\\Windows\\System32\\cmd.exe"), // Use Windows Command Prompt

command, // Pass command line args

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

0, // default flags - Normal Priority

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi ) // Pointer to PROCESS\_INFORMATION structure

)

{

printf( "CreateProcess failed (%d).\n", GetLastError() );

exit(EXIT\_FAILURE);

}

//printf("check\n");

// Wait until child process exits.

WaitForSingleObject( pi.hProcess, INFINITE );

// Close process and thread handles.

CloseHandle( pi.hProcess ); CloseHandle( pi.hThread );

//printf("you typed: %s\n", input);

free(input);

}

return 0;

}

## Output

