/\* Author(s): Sam Lee

\*

\* This is lab9.c the csc60mshell

\* This program serves as a skeleton for doing labs 9, 10, 11.

\* Student is required to use this program to build a mini shell

\* using the specification as documented in direction.

\* Date: Spring 2018

\*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

#include <fcntl.h>

#include <errno.h>

#define MAXLINE 80

#define MAXARGS 20

#define MAX\_PATH\_LENGTH 50

#define TRUE 1

/\* function prototypes \*/

int parseline(char \*cmdline, char \*\*argv);

//The two function prototypess below will be needed in lab10.

//Leave them here to be used later.

void process\_input(int argc, char \*\*argv);

void handle\_redir(int count, char \*argv[]);

/\* ----------------------------------------------------------------- \*/

/\* The main program starts here \*/

/\* ----------------------------------------------------------------- \*/

int main(void)

{

char cmdline[MAXLINE];

char \*argv[MAXARGS];

char \*cwd;

int argc;

int status;

pid\_t pid;

int i;

/\* Loop forever to wait and process commands \*/

while (TRUE) {

/\* Print your shell name: csc60mshell (m for mini shell) \*/

printf("csc60mshell> ");

/\* Read the command line \*/

fgets(cmdline, MAXLINE, stdin);

/\* Call parseline to build argc/argv \*/

argc = parseline(cmdline,argv);

/\* If user hits enter key without a command, continue to loop \*/

/\* again at the beginning \*/

/\* Hint: if argc is zero, no command declared \*/

/\* Hint: look up for the keyword "continue" in C \*/

/\* Handle build-in command: exit, pwd, or cd \*/

/\* Put the rest of your code here \*/

if (argc == 0) {

continue;

}

printf("Argc = %i\n",argc);

for (i = 0; i < argc; i++) {

printf("Argv %i = %s\n",i,argv[i]);

}

if (strncmp(argv[0],"exit",4) == 0) {

exit(EXIT\_SUCCESS);

}

else if (strncmp(argv[0],"pwd",3) == 0) {

char variable[MAX\_PATH\_LENGTH];

cwd = getcwd (variable, MAX\_PATH\_LENGTH);

if(cwd == NULL) {

perror("error, no such directroy exists");

}

else {

printf("%s\n",cwd);

continue;

}

}

else if (strncmp(argv[0],"cd",2) ==0){

char \*dir;

if (argc ==1) {

dir = getenv("HOME");

}

else {

dir = argv[1];

}

if (chdir(dir) == -1) {

perror("error changing directry");

continue;

}

}

//.......................IGNORE........................

/\* Else, fork off a process \*/

else {

pid = fork();

switch(pid)

{

case -1:

perror("Shell Program fork error");

exit(EXIT\_FAILURE);

case 0:

/\* I am child process. I will execute the command, \*/

/\* and call: execvp \*/

process\_input(argc, argv);

break;

default:

/\* I am parent process \*/

if (wait(&status) == -1)

perror("Parent Process error");

else

printf("Child returned status: %d\n",status);

break;

} /\* end of the switch \*/

//...end of the IGNORE above.........................

} /\* end of the if-else-if \*/

} /\* end of the while \*/

} /\* end of main \*/

/\* ----------------------------------------------------------------- \*/

/\* parseline \*/

/\* ----------------------------------------------------------------- \*/

/\* parse input line into argc/argv format \*/

int parseline(char \*cmdline, char \*\*argv)

{

int count = 0;

char \*separator = " \n\t"; /\* Includes space, Enter, Tab \*/

/\* strtok searches for the characters listed in separator \*/

argv[count] = strtok(cmdline, separator);

while ((argv[count] != NULL) && (count+1 < MAXARGS))

argv[++count] = strtok((char \*) 0, separator);

return count;

}

/\* ----------------------------------------------------------------- \*/

/\* process\_input \*/

/\* ----------------------------------------------------------------- \*/

void process\_input(int argc, char \*\*argv) {

/\* Step 1: Call handle\_redir to deal with operators: \*/

/\* < , or >, or both \*/

handle\_redir(argc,argv);

int returned\_value = execvp(argv[0], argv);

/\* Step 2: perform system call execvp to execute command \*/

/\* Hint: Please be sure to review execvp.c sample program \*/

if (returned\_value == -1) {

fprintf(stderr, "Error on the exec call\n");

\_exit(EXIT\_FAILURE);

}

}

/\* ----------------------------------------------------------------- \*/

void handle\_redir(int count, char \*argv[]) {

int out\_redir = 0;

int in\_redir = 0;

int i;

for (i = 0; i < count; i++) {

if (strcmp(argv[i], ">") == 0) {

if (out\_redir != 0) {

fprintf(stderr,"Cannot output to more than one file.\n");

\_exit(EXIT\_FAILURE); // shoulf this be \_exitfailure

}

else if (i == 0) {

fprintf(stderr,"No command entered.\n");

\_exit(EXIT\_FAILURE);

}

out\_redir = i;

}

else if (strcmp(argv[i], "<") == 0) {

if (in\_redir != 0) {

fprintf(stderr,"Cannot input from more than one file.\n");

\_exit(EXIT\_FAILURE);

}

else if (i == 0) {

fprintf(stderr,"No command entered.\n");

\_exit(EXIT\_FAILURE);

}

in\_redir = i;

}

}

int fd;

if (out\_redir != 0) {

if (argv[out\_redir + 1] == NULL) {

fprintf(stderr,"There is no file.");

\_exit(EXIT\_FAILURE);

}

fd = open(argv[out\_redir+1], O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

if (fd == -1) {

fprintf(stderr,"Error opening out\_redir file\n");

\_exit(EXIT\_FAILURE);

}

dup2(fd,1);

close(fd);

argv[out\_redir] = NULL;

}

int fd1;

if (in\_redir != 0) {

if(argv[in\_redir + 1] == NULL) {

fprintf(stderr,"There is no file.");

\_exit(EXIT\_FAILURE);

}

fd1 = open(argv[in\_redir+1], O\_RDONLY,S\_IRUSR | S\_IWUSR);

if(fd1 == -1) {

fprintf(stderr,"Error opening at in\_redir file\n");

\_exit(EXIT\_FAILURE);

}

dup2(fd1,0);

close(fd1);

argv[in\_redir] = NULL;

}

}

/\* ----------------------------------------------------------------- \*/