#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <fcntl.h>

char \*home;

int main(int argc, char \*argv[], char \*env[])

{

char input[1024], \*commandArray[1024], \*ask = "Type Something> ";

int i = 0, j = 1;

home = getenv("HOME"); // get home directory, searching the env array manually caused a crash on env[8] using Ubuntu

printf("\033[0;32m");

printf("%s", ask);

printf("\033[0m;");

// get input each cycle

while (fgets(input, 1024, stdin) != NULL)

{

if (strcmp(input, "") != 0 && strcmp(input, "\n") != 0) // do we have a valid input string?

{

// tokenize input into pipes

commandArray[0] = strtok(input, "|");

while (commandArray[j] = strtok(0, "|"))

j++;

// process the various (potentially) commands

if (j == 1) // single command

{

processCommand(commandArray[0]);

}

else // multiple commands (currently only handles two commands)

{

for (i = 0; i < j - 1; i++)

processPipe(commandArray[i], commandArray[i + 1]);

}

}

// reset variables and repeat input qufestion

memset(input, 0, sizeof(input));

j = 1;

i = 0;

printf("%s", ask);

}

return(0);

}

// process piped commands

//create a pipe

//fork a child process to share the pipe

//arrange one process as the pipe writer, and the other one as the pipe reader.

//fork a child process to share the pipe

//arrange one process as the pipe writer, and the other one as the pipe reader.

void processPipe(char commandA[512], char commandB[512])

{

char \*Aargs[512], \*Bargs[512], \*cmd, \*arg;

int pd[2], pid, i = 1, statusA, statusB;

int redirA = 0, rediriA = 0, redirB = 0, rediriB = 0;

// Breakup commandA

cmd = strtok(commandA, " ");

Aargs[0] = cmd;

arg = strtok(0, " ");

while (arg)

{

if (arg[strlen(arg) - 1] == '\n')

arg[strlen(arg) - 1] = '\0';

Aargs[i] = arg;

// handle redirects

if (strcmp(arg, "<") == 0) // input

{

rediriA = i;

redirA = 1;

}

else if (strcmp(arg, ">") == 0) // output

{

rediriA = i;

redirA = 2;

}

else if (strcmp(arg, ">>") == 0) // output and append

{

rediriA = i;

redirA = 3;

}

arg = strtok(0, " ");

i++;

}

if (!rediriA)

Aargs[i] = NULL;

else

Aargs[rediriA] = NULL;

// Breakup commandB

cmd = strtok(commandB, " ");

Bargs[0] = cmd;

arg = strtok(0, " ");

i = 1;

while (arg)

{

if (arg[strlen(arg) - 1] == '\n')

arg[strlen(arg) - 1] = '\0';

Bargs[i] = arg;

// handle redirects

if (strcmp(arg, "<") == 0) // input

{

rediriB = i;

redirB = 1;

}

else if (strcmp(arg, ">") == 0) // output

{

rediriB = i;

redirB = 2;

}

else if (strcmp(arg, ">>") == 0) // output and append

{

rediriB = i;

redirB = 3;

}

arg = strtok(0, " ");

i++;

}

if (!redirB)

Bargs[i] = NULL;

else

Bargs[rediriB] = NULL;

// do the commands

pipe(pd);

if (fork() == 0)

{

dup2(pd[1], 1);

close(pd[0]);

close(pd[1]);

if (redirA == 1) // infile

{

close(0);

open(Aargs[i - 1], O\_RDONLY);

}

else if (redirA == 2) // outfile

{

close(1);

open(Aargs[i - 1], O\_WRONLY | O\_CREAT, 0644);

}

else if (redirA == 3) // outfile append

{

close(1);

open(Aargs[i - 1], O\_WRONLY | O\_APPEND);

}

execvp(Aargs[0], Aargs);

}

if (fork() == 0)

{

dup2(pd[0], 0);

close(pd[0]);

close(pd[1]);

if (redirB == 1) // infile

{

close(0);

open(Bargs[i - 1], O\_RDONLY);

}

else if (redirB == 2) // outfile

{

close(1);

open(Bargs[i - 1], O\_WRONLY | O\_CREAT, 0644);

}

else if (redirB == 3) // outfile append

{

close(1);

open(Bargs[i - 1], O\_WRONLY | O\_APPEND);

}

execvp(Bargs[0], Bargs);

}

close(pd[0]);

close(pd[1]);

wait(&statusA);

wait(&statusB);

printf("Command 1 Exit: %d, Command 2 Exit: %d\n", statusA, statusB); // print exit status codes

}

// process a single command

void processCommand(char command[512])

{

char \*args[512], \*cmd, \*arg;

int pid, status, redir = 0, rediri = 0, i = 1, builtin = 0;

command[strlen(command) - 1] = 0;

// get actual command

cmd = strtok(command, " ");

arg = strtok(0, " ");

// built in commands or not?

if (!builtInCommands(cmd, arg))

{

// get arguments

args[0] = cmd;

while (arg != NULL)

{

args[i] = arg;

// handle redirects

if (strcmp(arg, "<") == 0) // input

{

rediri = i;

redir = 1;

}

else if (strcmp(arg, ">") == 0) // output

{

rediri = i;

redir = 2;

}

else if (strcmp(arg, ">>") == 0) // output and append

{

rediri = i;

redir = 3;

}

arg = strtok(0, " ");

i++;

}

// add null pointer

if (!rediri)

args[i] = NULL;

else

args[rediri] = NULL;

// fork off new process

pid = fork();

if (!pid)

{

if (redir == 1) // infile

{

close(0);

open(args[i - 1], O\_RDONLY);

}

else if (redir == 2) // outfile

{

close(1);

open(args[i - 1], O\_WRONLY | O\_CREAT, 0644);

}

else if (redir == 3) // outfile append

{

close(1);

open(args[i - 1], O\_WRONLY | O\_APPEND);

}

execvp(cmd, args);

printf("couldn't execute: %s\n", cmd);

}

else if (pid)

{

pid = wait(&status);

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

}

}

}

// do a built-in command

int builtInCommands(char \*cmd, char \*arg)

{

int done = 0;

// cd command

if (strcmp(cmd, "cd") == 0)

{

done = 1;

// no directory specified and if home directory is specified

if (!arg)

{

if (home)

{

chdir(home);

}

}

// directory specified

else

chdir(arg);

}

// pwd command

else if (strcmp(cmd, "pwd") == 0)

{

printf("%s\n", get\_current\_dir\_name());

done = 1;

}

// exit command

else if (strcmp(cmd, "exit") == 0)

{

exit(1);

done = 1;

}

return done;

}