/\*

\* pipe\_command.c : deal with pipes

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <errno.h>

#include <signal.h>

#include "shell.h"

#define STD\_OUTPUT 1

#define STD\_INPUT 0

void pipe\_and\_exec(char \*\*myArgv) {

int pipe\_argv\_index = pipe\_present(myArgv);

int pipefds[2];

char \*\*left\_argv;

char \*\*right\_argv;

signal(SIGINT,SIG\_DFL);

signal(SIGQUIT,SIG\_DFL);

switch (pipe\_argv\_index) {

case -1: /\* Pipe at beginning or at end of argv; See pipe\_present(). \*/

fputs ("Missing command next to pipe in commandline.\n", stderr);

errno = EINVAL; /\* Note this is NOT shell exit. \*/

break;

case 0: /\* No pipe found in argv array or at end of argv array.

\*See pipe\_present(). Exec with whole given argv array.

\*

\* fill in code

\*/

execvp(myArgv[0],myArgv);

break;

default:/\* Pipe in the middle of argv array. See pipe\_present(). \*/

/\* Split arg vector into two where the pipe symbol was found.

\* Terminate first half of vector.

\*

\* Fill in code. \*/

left\_argv = (char\*\*)malloc(pipe\_argv\_index\* sizeof(char\*));

for(int i=0;i < pipe\_argv\_index ;i++){

left\_argv[i] = (char\*)malloc(strlen(myArgv[i])\*sizeof(char));

strcpy(left\_argv[i],myArgv[i]);

}

/\* Create a pipe to bridge the left and right halves of the vector.

\*

\* Fill in code. \*/

if (pipe(pipefds) == -1) {

perror("pipe");

exit(1);

}

/\* Create a new process for the right side of the pipe.

\* (The left side is the running "parent".)

\*

\* Fill in code to replace the underline. \*/

//pid\_t pid;

//pid = fork();

switch(fork()) {

case -1 :

//perror("fork");

break;

/\* Talking parent. Remember this is a child forked from shell. \*/

default :

/\* - Redirect output of "parent" through the pipe.

\* - Don't need read side of pipe open. Write side dup'ed to stdout.

\* - Exec the left command.

\*

\* Fill in code. \*/

close(pipefds[0]);

dup2(pipefds[1],STD\_OUTPUT);

execvp(left\_argv[0],left\_argv);

//之後有空+free()

close(pipefds[1]);

break;

/\* Listening child. \*/

case 0 :

/\* - Redirect input of "child" through pipe.

\* - Don't need write side of pipe. Read side dup'ed to stdin.

\* - Exec command on right side of pipe and recursively deal with other pipes

\*

\* Fill in code. \*/

close(pipefds[1]);

dup2(pipefds[0],STD\_INPUT);

close(pipefds[0]);

pipe\_and\_exec(&myArgv[pipe\_argv\_index+1]);

}

}

perror("Couldn't fork or exec child process");

exit(errno);

}