mysh.h

#ifndef \_\_MYSH\_H\_\_

#define \_\_MYSH\_H\_\_

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <pwd.h>

#include <signal.h>

#include <termio.h>

#include <limits.h>

#include <sys/wait.h>

#include <setjmp.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#define MAXSIZE 1024

#define MAXBUF 256

char shell[9][MAXSIZE];

char \*curpath;

int background;

int work\_num;

sigset\_t blockmask;

pid\_t childpid;

struct sigaction defhandler;

char abspath[MAXSIZE];

struct passwd \*userinfo;

int getcommand(char \*);

int ispipe(char \*\*);

int forkproc(char \*\*);

void extrapipeproc(char \*\*);

int makeargv(const char \*, const char \*, char \*\*\*);

void pipeproc(char \*\*);

int signalsetup(struct sigaction \*def, sigset\_t \*mask, void (\*handler)(int));

static void jumphd(int signalnum);

static sigjmp\_buf jumptoprompt;

static volatile sig\_atomic\_t oktojump = 0;

static int server();

static int client();

#endif

main.c

#include "mysh.h"

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

char command[MAXSIZE]= {0};

char \*com\_tmp;

char \*buff;

char \*buff\_tmp;

char \*\*argvp;

char \*token;

char c;

int status;

int com\_count= 1;

int buff\_count = 0;

int i=0, n=0;

if (signalsetup(&defhandler, &blockmask, jumphd)==-1){

perror("set up shell signal handling");

return 1;

}

memset(abspath, 0, sizeof(abspath));

background = 0;

work\_num = 1;

if(argc!=1){

perror("usage : mysh");

exit(-1);

}

userinfo = getpwuid(getuid());

getcwd(abspath, MAXSIZE);

memset(shell, 0, sizeof(shell));

while(1){

if(sigsetjmp(jumptoprompt,1))

continue;

wait(NULL);

oktojump = 1;

if(getcommand(command)==0)

continue;

//command com\_tmp 저장 에러시 다시 프롬프트

if((com\_tmp = (char \*)malloc(strlen(command)+1))==NULL){

perror("Can't allocate command's temp string variable");

continue;

} else {

strcpy(com\_tmp, command);

}

//저장된 명령어 개수 ;구분자

if(strtok(com\_tmp, ";") != NULL)

for(com\_count = 1; strtok(NULL, ";") !=NULL ; com\_count++);

//command 개수 만큼 구행

for(i=1; i<=com\_count; i++){

//각 command 공백으로 구분 argvp 이중 포인터 배열저장

strcpy(com\_tmp, command);

//token으로 분리

token = strtok(com\_tmp, ";");

for(n=1; n<i;n++)

token = strtok(NULL, ";");

//분리한 token을 buff와 buff\_tmp저장

if((buff = (char \*)malloc(strlen(token)+1))==NULL){

perror("Can't malloc");

continue;

} else {

if((buff\_tmp = (char \*)malloc(strlen(token)+1))==NULL){

perror("Cant malloc buff\_tmp");

continue;

}

strcpy(buff, token);

strcpy(buff\_tmp, token);

}

//buff\_tmp 저장 토큰 수

if(strtok(buff\_tmp, " ") !=NULL)

for(buff\_count = 1;strtok(NULL, " ") !=NULL; buff\_count++);

if((argvp = (char \*\*)malloc((buff\_count+1)\*sizeof(char \*))) == NULL){

free(buff);

free(buff\_tmp);

perror("Cant malloc argvp String");

continue;

}

//토큰포인트 인자를 배열 삽입

\*argvp = strtok(buff, " ");

for(n=1; n< buff\_count;n++){

\*(argvp + n) = strtok(NULL, " ");

//인자중 background 체크

//argvp에 포함안함

if(strcmp(\*(argvp+n), "&")==0){

background=1;

buff\_count-=1;

}

}

\*(argvp+buff\_count)=NULL;

if(forkproc(argvp)==0)

continue;

else break;

}

}

return 0;

}

int getcommand(char \*command){

int n=0, space\_chk=0;

char path[MAXSIZE] = {0};

char c;

fflush(NULL);

curpath = (char \*)getenv("PWD");

printf("[%s@%s]$ ","mysh", curpath);

sprintf(path, "[%s@%s]$ ", "mysh", curpath);

while((c=fgetc(stdin))){

//enter 종료

if(c==10){

fputc('\n',stdout);

break;

}

//Ctrl+D 종료

if(c==4){

fputc('\n', stdout);

return 0;

}

//backspace 제외 입력무시

if((c<=26)&&(c!=8))

break;

switch(c){

//backspace key 입력처리

case 8:

if(n==0)

break;

fputc('\b', stdout);

fputc(' ', stdout);

fputc('\b', stdout);

command[--n] = (char)0;

break;

default:

fputc(c, stdout);

command[n++] = (char)c;

break;

}

}

command[n]='\0';

//입력 없을때 다시 프롬프트

if(strlen(command)==0)

return 0;

//공백만 포함 다시 프롬프트

for(n=0; n<strlen(command);n++)

if(command[n] != ' ' && command[n] != '\t')

return 1;

return 0;

}

int ispipe(char \*\*argvp){

int n, check =0;

for(n=0; argvp[n]!=NULL;n++){

if(strcmp(argvp[n], "<")==0)

{

check = 1;

break;

} else if(strcmp(argvp[n],">")==0){

check=1;

break;

} else if(strcmp(argvp[n], ">>")==0){

check=1;

break;

} else if(strcmp(argvp[n], "|")==0){

check=1;

break;

} else {

check = 0;

}

}

return check;

}

static int server(){

int ssock, csock;

int clen;

struct sockaddr\_in client\_addr, server\_addr;

char like[MAXBUF]="좋아!!";

char buf[MAXBUF];

if((ssock=socket(PF\_INET, SOCK\_STREAM,IPPROTO\_TCP))<0){

perror("socket error :");

exit(1);

}

clen = sizeof(client\_addr);

//구조체 주소 지정

memset(&server\_addr,0,sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr=htonl(INADDR\_ANY);

server\_addr.sin\_port = htons(3317);

//바인드를 통해서 서버의 주소 설정

if(bind(ssock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr))<0){

perror("bind error :");

exit(1);

}

//위의 주소로 지정한 클라이언트 접속을 기다림

if(listen(ssock,8)<0){

perror("listen error : ");

exit(1);

}

while(1){

//클라이언트가 접속하면 접속허용 클라이언트 소켓생성

csock = accept(ssock, (struct sockaddr\*)&client\_addr,&clen);

if(read(csock,buf,MAXBUF)<0)

perror("read error");

strcat(buf,like);

//클라이언트로 buf에 전송

if(write(csock, buf, MAXBUF)<=0)

perror("write error :");

close(csock);

}

return 0;

}

static int client(){

int ssock;

int clen;

struct sockaddr\_in server\_addr;

char buf[MAXBUF];

//소켓 생성

if((ssock=socket(PF\_INET, SOCK\_STREAM, IPPROTO\_TCP))<0){

perror("socket error : ");

exit(1);

}

clen = sizeof(server\_addr);

//서버 부분 초기화 및 값 세팅

memset(&server\_addr,0,sizeof(server\_addr));

server\_addr.sin\_family = AF\_INET;

server\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

server\_addr.sin\_port = htons(3317);

//서버에 연결

if(connect(ssock,(struct sockaddr\*)&server\_addr,clen)<0){

perror("connent error : ");

exit(1);

}

//값을 읽어서 서버에 전송

scanf("%s",buf);

if(write(ssock,buf,MAXBUF)<0)

perror("write error");

memset(buf,0,MAXBUF);

//서버의 문자열을 받아서 출력

if(read(ssock,buf,MAXBUF)<0){

perror("read error");

exit(1);

}

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

close(ssock);

return 0;

}

int forkproc(char \*\*argvp){

int pid;

int status;

int pipefd[2];

char tmppath[MAXSIZE] = {0};

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

perror("cant fork()");

return -1;

} else if(pid == 0){

if(background==1){

printf("[%d] %d\n",work\_num++,getpid());

if((sigaction(SIGTSTP, &defhandler,NULL)==-1)||

(sigprocmask(SIG\_UNBLOCK,&blockmask,NULL)==-1)){

perror("set up signal failed");

return 1;

}

} else {

signal(SIGINT, SIG\_DFL);

}

//pipe이면 pipe수행

if(ispipe(argvp))

pipeproc(argvp);

//명령어가 server이면 server 수행

if(strcmp(\*argvp, "server")==0){

background = 1;

server();

}

//명령어가 client이면 client 수행

if(strcmp(\*argvp, "client")==0){

client();

}

else {

execvp(\*argvp, argvp);

}

//수행끝 종료

exit(0);

}

//부모 프로세스 수행

else {

//자식프로세스가 백그라운드 수행

//부모는 다음 명령수행

if(background==1){

background = 0;

return 0;

}

//자식프로세스 백그라운드 수행아니면 기다림

else {

if((pid==waitpid(pid, &status, 0))<0){

perror("wait pid error");

return -1;

}

usleep(10);

return 0;

}

}

}

int makeargv(const char \*s, const char \*delimiters, char \*\*\*argvp){

int i;

int numtokens;

const char \*snew;

char \*t;

if((s==NULL)||(delimiters==NULL)||(argvp==NULL)){

return -1;

}

\*argvp = NULL;

//들어온 명령어에 구분자를 찾아서 할당한다.

snew = s + strspn(s, delimiters);

if((t=malloc(strlen(snew)+1))==NULL)

return -1;

//임시 저장소에 아까 구분자로 나눈 명령어를 할당한다.

strcpy(t,snew);

numtokens = 0;

//토큰을 센다.

if(strtok(t, delimiters)!=NULL)

for(numtokens=1;strtok(NULL,delimiters)!=NULL;numtokens++);

//리턴할 배열에 토큰으로 나누어진 숫자로 할당한다.

if((\*argvp=malloc((numtokens+1)&sizeof(char \*)))==NULL){

free(t);

return -1;

}

//토큰이 없을 때에는 할당을 임시 문자열 해제한다.

if(numtokens == 0)

free(t);

else{

//토큰으로 나눈 부분을 배열로 만든다.

strcpy(t,snew);

\*\*argvp = strtok(t,delimiters);

for(i=1;i<numtokens;i++)

\*((\*argvp)+i)=strtok(NULL,delimiters);

}

//마지막의 명령어 문자배열에 널을 할당한다.

\*((\*argvp)+numtokens)=NULL;

//토큰의 개수를 반환한다.

return numtokens;

}

//명령어를 처리한다.

void extraproc(char \*s){

char \*\*chargv;

makeargv(s, "\t", &chargv);

execvp(chargv[0], chargv);

exit(1);

}

void pipeproc(char \*\*argvp){

int child;

int count;

int fds[2];

int i;

char \*\*pipelist;

count = makeargv(\*argvp, "|", &pipelist);

//'|'만큼 반복한다.

for(i=0;i<count-1;i++){

if(pipe(fds)==-1)

perror("pipe error");

else if((child=fork())==-1)

perror("fork error");

//자식 수행 표준 출력으로 보냄

else if(child){

dup2(fds[1], STDOUT\_FILENO);

close(fds[0]);

close(fds[1]);

extraproc(pipelist[i]);

exit(1);

}

//부모의 경우 자식의 결과를 표준입력으로

dup2(fds[0], STDIN\_FILENO);

close(fds[0]);

close(fds[1]);

}

//처리하고 남은부분 처리

extraproc(pipelist[i]);

exit(1);

}

int signalsetup(struct sigaction \*def, sigset\_t \*mask, void (\*handler)(int)){

//시그널에 관한 설정을 한다.

struct sigaction catch;

catch.sa\_handler = handler;

def->sa\_handler = SIG\_DFL;

catch.sa\_flags = 0;

if ((sigemptyset(&(def->sa\_mask)) == -1) ||

(sigemptyset(&(catch.sa\_mask))==-1)||

(sigaction(SIGTSTP,&catch,NULL)==-1)||

(sigemptyset(mask)==-1)||

(sigaddset(mask,SIGTSTP)))

return -1;

return 0;

}

//시그널 입력시에 지정한 곳으로 이동한다.

static void jumphd(int signalnum){

if(!oktojump) return;

oktojump=0;

siglongjmp(jumptoprompt,1);

}