|  |
| --- |
|  |
| [ MiniShell ] |
|  |
| [minishell 구현 리포트] |

**컴퓨터 공학부 3학년**

2017 10월 22일

작성자: 20153208 이소영

[ MiniShell ]

[minishell 구현 리포트]

1.자신의 접근 방법, 구현 방법 등을 기술, UNIX System Call (fork,exec,wait)의 study 결과

**접근 방법 및 구현방법** : 우선적으로 입력을 받는 메소드를 만들고 그 메소드에서 입력받은prompt를 읽어서 buffer에 저장한다. 버퍼에 넣을 때 변수로 하나씩 읽으면서 버퍼의 사이즈를 넘지 않는지 등을 확인해서 넣고, 만일 줄이 너무 길 경우 길다고 출력해주고 다시 입력을 받는다.

그렇게 버퍼배열에 입력받은 문자를 보통 인수의 일부가 될 수 있는지 특별한 문자인지 조사하는 메소드를 만든다.

또한, 토큰을 가져와서 토큰버퍼에 넣기위해 토큰을 포함하고 있는 버퍼로부터 여백을 제거하고 버퍼내의 토큰에 따라 유형 변수를 지정하여 구현한다. 그리고 interrupt을 위해 ctrl+c 일 경우 처리해주는 메소드도 만들어줘야한다.

Wait을 선택사항으로 하여 명령을 수행하기 위해 signal을 무시해주도록 구현하고 pid를 fork하고 만일 백그라운드 프로세스이면 프로세스 식별자를 프린트하고 퇴장한다.

마지막으로 입력줄을 처리하는 메소드를 만들어 토큰 유형에 따라 case를 나누어 처리한다.

main에서는 입력이 EOF가 아닐때까지 loop를 돌리고 계속 입력받은 것을 처리해주는 메소드를 호출해준다.

**UNIX System Call의 (fork,exec,wait)Study 결과**

Fork : fork 시스템 호출은 기본적인 프로세스 생성 프리미티브이다.

* 이 기법을 사용하여 UNIX기 다중 태스킹 시스템으로 전환된다.

Fork가 성공적으로 수행되면 커널은 호출하는 프로세스와 (거의) 똑같은 복사본을 새로운 프로세스로서 생성한다. 여기서 새로 생성된 프로세스를 자식 프로세스라고 하고 fork를 호출한 프로세스를 부모 프로세스라고한다.

Fork의 호출로 자식 프로세스가 생성된 후에는 부모와 자식 프로세스가 동시에 수행된다.

부모 프로세스에서 pid는 0이 아닌 양의 정수값을 갖고 자식 프로세스에서는 0을 갖는다.

부모와 자식의 리턴값이 서로 다르기 때문에 이를 이용하여 서로 다른 동작을 지정할 수 있다.

Exec : 호출 프로세스의 기억 공간에 새로운 프로그램을 적재함으로써 호출 프로세스를 변형시킨다. Exec가 성공적으로 수행되면 호출 프로그램은 완전히 새로운 프로그램으로 대치되고 이어 그 프로그램의 처음부터 수행되기 시작한다. (단,식별번호를 그대로 유지한다.)

-exec; : 첫 번째 인수인 path의 새로이 수행될 프로그램이 들어있는 파일의 이름을 가리킨다.

Execl과 유사기능의 호출들이 복귀할 때는 항상 -1을 돌려준다.(호출 프로그램이 살아남아 execl이 복귀하면 오류가 발생하기 때문)

여기서 fork와 exec를 함께 사용하면 fork로 자식 프로세스를 만들고, 그 자식 프로세스안에서 exec을 이용하면, 한 프로그램은 자신을 죽이지 않고도 전혀 다른 프로그램을 부프로세스 내에서 수행할 수 있다.

Wait : wait은 자식 프로세스가 수행되고 있는 동안 부모 프로세스의 수행을 일시적으로 중단시킨다. 자식이 수행을 마치면, 기다리던 부모는 수행을 재개한다. 하나 이상의 자식이 수행되고 있으면 wait은 자식 프로세스들 중 하나가 퇴장하기만 하면 바로 복귀한다.

Wait은 하나의 인수,status를 가지며 이것은 정수를 가리키는 포인터이다. 만일 포인터가 NULL이면 인수는 단순히 무시된다. 그러나 유효한 포인터가 인수로 전달되면 이 포인터가 가리키는 정수는 wait가 복귀될 때 유용한 상태 정보를 나타낸다.

## 2. TESTCASE

Smallsh.h 라는 헤더파일을 만들어서 명령 처리를 위한 정의를 한다.

Mini\_shell.c로 구현을 했음으로 gcc -o mini\_shell.out mini\_shell.c 로 하여 실행파일을 만든다.

./mini\_shell.out으로 실행을 해준다.

Command> //prompt이다.

**Command > ls**

현재 디렉토리에 있는 파일 및 디렉토리의 목록이 보여진다. Ex) mini\_shell.c minshell.out smallsh.h

**Command> ls -l**

Ls에 -l이라는 명령어를 붙여 현재 디렉토리가 가진 파일 및 디렉토리의 접근권한 등의 상세한 정보의 목록을 보여준다.

Ex) total 72

-rwxr-xr-x 1 pi pi 8308 Sep 20 07:59 copyfile

….등

**Command> pwd**

현재 디렉토리를 출력해준다. Ex) /home/pi

**Command> echo Hello!**

Hello! 를 출력해준다.

**Command> cat hello.txt**

Hello! (지정한 파일의 내용을 출력해준다.)

### 3. 부록: 소스 코드

1. mini\_shell.c : mini\_shell을 실제 구현

2.smallsh.h : mini\_shell에서 명령 처리기를 위한 정의들

<smallsh.h>

#include <stdio.h>

#include <unistd.h>

#define EOL 1 /\* 줄의 끝 \*/

#define ARG 2 /\* 정상적 인수 \*/

#define AMPERSAND 3

#define SEMICOLON 4

#define INTERRUPT 5

#define MAXARG 512 /\* 명령인수의 최대수 \*/

#define MAXBUF 512 /\* 입력줄의 최대길이\*/

#define FOREGROUND 0

#define BACKGROUND 1

#define END\_CMD "goodbye"

<Mini\_shall.c>

#include "smallsh.h"

#include <stdlib.h>

#include <signal.h>

/\* 프로그램 버퍼 및 작업용 포인터들 \*/

static char inpbuf[MAXBUF], tokbuf[2\*MAXBUF], \*ptr = inpbuf, \*tok = tokbuf;

int intr\_p = 0;

char \*prompt = "Command>"; /\* prompt \*/

int fg\_pid = 0;

int userin(char \*p) //프롬프트를 프린트하고 한줄을 읽는다.

{

int c, count;

/\* 초기화 \*/

ptr = inpbuf;

tok = tokbuf;

/\* 프롬프트 표시 \*/

printf("%s ", p);

for(count = 0;;){

if((c = getchar()) == EOF)

return(EOF);

if(count < MAXBUF)

inpbuf[count++] = c;

if(c == '\n' && count < MAXBUF){

inpbuf[count] = '\0';

return(count);

}

/\* 줄이 너무 길면 재시작 \*/

if(c == '\n'){

printf("smallsh: input line too long\n");

count = 0;

printf("%s ", p);

}

}

}

static char special[] = {' ', '\t', '&', ';', '\n', '\0'};

int inarg(char c) /\* 한 문자가 보통 인수의 일부가 될 수 있는지를 결정 \*/

{

char \*wrk;

for(wrk = special; \*wrk != '\0'; wrk++)

if(c == \*wrk)

return(0);

return(1);

}

int gettok(char \*\*outptr) /\* 토큰을 가져와서 tokbuf에 넣는다. \*/

{

int type;

/\* outptr 문자열을 tok로 지정한다.\*/

\*outptr = tok;

/\* 토큰을 포함하고 있는 버퍼로부터 여백을 제거한다. \*/

for(;\*ptr == ' ' || \*ptr == '\t'; ptr++)

;

/\*토큰을 포함하고 있는 버퍼로부터 여백을 제거한다.\*/

\*tok++ = \*ptr;

/\*버퍼내의 토큰에 따라 유형 변수를 지정한다.\*/

switch(\*ptr++){

case '\n':

type = EOL; break;

case '&':

type = AMPERSAND; break;

case ';':

type = SEMICOLON; break;

default:

type = ARG;

while(inarg(\*ptr))

\*tok++ = \*ptr++;

}

\*tok++ = '\0';

return(type);

}

void handle\_int(int s) {

int c;

if(!fg\_pid) {

/\* ctrl-c hit at shell prompt \*/

return;

}

if(intr\_p) {

printf("\ngot it, signalling\n");

kill(fg\_pid, SIGTERM);

fg\_pid = 0;

} else {

printf("\nignoring, type ^C again to interrupt\n");

intr\_p = 1;

}

}

/\* wait을 선택사항으로 하여 명령을 수행한다. \*/

int runcommand(char \*\*cline,int where)

{

int pid, exitstat, ret;

/\* signal을 무시한다.(linux) \*/

struct sigaction sa\_ign, sa\_conf;

sa\_ign.sa\_handler = SIG\_IGN;

sigemptyset(&sa\_ign.sa\_mask);

sa\_ign.sa\_flags = SA\_RESTART;

sa\_conf.sa\_handler = handle\_int;

sigemptyset(&sa\_conf.sa\_mask);

sa\_conf.sa\_flags = SA\_RESTART;

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

perror("smallsh");

return(-1);

}

if(pid == 0){

sigaction(SIGINT, &sa\_ign, NULL);

sigaction(SIGQUIT, &sa\_ign, NULL);

execvp(\*cline, cline);

perror(\*cline);

exit(1);

} else {

fg\_pid = pid;

}

/\* 부모의 코드 \*/

/\* 만일 백그라운드 프로세스이면 프로세스 식별자를 프린트하고 퇴장한다. \*/

if(where == BACKGROUND){

fg\_pid = 0;

printf("[Process id %d]\n", pid);

return(0);

}

/\* 프로세스 pid가 퇴장할 때까지 기다린다. \*/

sigaction(SIGINT, &sa\_conf, NULL);

sigaction(SIGQUIT, &sa\_conf, NULL);

while( (ret=wait(&exitstat)) != pid && ret != -1)

;

fg\_pid = 0;

return(ret == -1 ? -1 : exitstat);

}

void procline() /\* 입력줄을 처리한다. \*/

{

char \*arg[MAXARG+1]; /\*runcommand를 위한 포인터 배열 \*/

int toktype; /\* 명령내의 토큰의 유형 \*/

int narg; /\* 지금까지의 인수 수 \*/

int type; /\* FOREGROUND or BACKGROUND \*/

/\* intr\_p 초기화\*/

intr\_p = 0;

for(narg = 0;;){ /\* 무한 루프 \*/

/\* 토큰 유형에 따라 case를 나눈다. \*/

switch(toktype = gettok(&arg[narg])){

case ARG:

if(narg < MAXARG)

narg++;

break;

case EOL:

case SEMICOLON:

case AMPERSAND:

type = (toktype == AMPERSAND) ? BACKGROUND : FOREGROUND;

if(narg != 0){

arg[narg] = NULL;

runcommand(arg, type);

}

if(toktype == EOL)

return;

narg = 0;

break;

}

}

}

int main()

{

/\* sigaction struct (linux) \*/

struct sigaction sa;

sa.sa\_handler = SIG\_IGN;

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = SA\_RESTART;

sigaction(SIGINT, &sa, NULL);

sigaction(SIGQUIT, &sa, NULL);

while(userin(prompt) != EOF){

procline();

}

}