**MY SHELL**

오퍼레이팅 시스템

학 번 12191659

이 름 이화연

소속학과 컴퓨터공학과

교 수 님 정진만 교수님

제출일자 2021.05.07

1. **개발환경**

리눅스 배포판은 ubuntu-20.04.2.0을 사용하였고 가상머신은 Oracle VM VirtualBOX를 사용하였다.

1. **쉘 프로그램 설계 및 구현 내용**

설계 및 구현 내용은 하단의 코드와 주석으로 설명하겠다.

#include <unistd.h>

#include <stdio.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <string.h>

#include <stdlib.h>

#include <stdbool.h>

#define MAX 1024

// history 버퍼의 크기를 10으로 한다.

char\* history[10];

int now\_hist = 0;

// history 버퍼를 초기화시켜준다.

void history\_init(char\*\* buf\_history) {

for(int i=0; i<10; ++i) {

buf\_history[i] = NULL;

}

}

// history 버퍼를 차례로 출력한다.

void history\_print(char\*\* buf\_history, int cur) {

int n = cur;

do {

if(buf\_history[n]) {

fprintf(stdout, "%s\n", buf\_history[n]);

}

n = (n+1) % 10;

}while(n != cur);

}

// history 버퍼를 비워준다.

void history\_clear(char\*\* buf\_history) {

for(int i=0;i<10;++i) {

free(buf\_history[i]);

buf\_history[i] = NULL;

}

}

// 프롬포트에 "12191659\_shell$"을 출력한 다음 문자를 입력받는다.

// 만약 엔터가 입력된다면 prompt 함수를 종료하고 다른 문자가 입력된다면 cmd 배열에 문자 하나하나 저장한다.

// cmd에는 명령어 라인이 입력될 것이다.

int prompt(char\* cmd) {

char ch;

int n = 0;

fflush(NULL);

fprintf(stdout, "12191659\_shell$ ");

while(ch = fgetc(stdin)) {

if(ch == 10) { // ENTER key

break;

}

else {

cmd[n++] = ch;

}

}

}

// cmd에 저장되어 있는 명령어를 공백이나 개행(" "또는 "\n")을 기준으로 잘라서 arg배열에 차례로 넣는다.

void parse(char\* cmd, char\*\* arg, int\* arg\_cnt) {

int n = 0;

char\* tmp = strtok(cmd, " \n");

while(tmp != NULL) {

arg[n] = tmp;

tmp = strtok(NULL, " \n");

n++;

}

arg[n] = NULL;

\*arg\_cnt = n;

}

// 내장형 명령어가 아닌 명령어들을 처리하기 위한 함수이다.

// fork 시스템 호출을 통해 새로운 프로세스를 생성하고 wait 시스템 호출을 통해 프로세스의 종료를 기다린다.

// 백그라운드 실행이라면 wait하지 않는다.

bool execute(char\*\* arg,int arg\_cnt) {

int status;

pid\_t pid;

bool background = (strcmp(arg[arg\_cnt-1],"&")==0) ? 1 : 0;

pid = fork();

if(pid == -1) {

fprintf(stdout, "fork error\n");

return 0;

}

else if(pid == 0) {

if(execvp(arg[0], arg) == -1) {

fprintf(stdout, "execvp error\n");

return 0;

}

exit(0);

}

else {

if(background == 1) {

return 1;

}

else {

wait(&status);

return 1;

}

}

}

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

char cmd[MAX]; // 입력받은 명령어를 저장하는 문자열

history\_init(history);

while(1) {

char\* arg[MAX];

int arg\_cnt = 0;

// 입력받은 명령어를 저장하는 cmd를 초기화시켜준다.

cmd[0] = '\0';

// 프롬포트를 출력하고 명령어를 입력받는다.

prompt(cmd);

// 입력받은 명령어를 공백과 개행을 기준으로 나누어서 arg배열에 저장한다.

parse(cmd, arg, &arg\_cnt);

// 입력받은 명령어를 history에 복제한다.

history[now\_hist] = strdup(arg[0]);

now\_hist = (now\_hist+1) % 10;

if(arg\_cnt == 0) continue;

// 내장형 명령어 quit는 쉘프로그램을 종료시킨다.

if(strcmp(arg[0],"quit") == 0 || strcmp(cmd,"quit\0") == 0) {

fprintf(stdout, "myshell developed by leehwayeon(12191659)\n");

exit(0);

}

// 내장형 명령어 history는 사용자로부터 최근 입력받은 10개까지의 명령어 라인을 출력한다.

else if(strcmp(arg[0],"history") == 0 || strcmp(cmd,"history\0") == 0) {

history\_print(history, now\_hist);

}

// 내장형 명령어 help는 간단한 메뉴얼을 출력한다.

else if(strcmp(arg[0], "help") == 0 || strcmp(arg[0], "help\0") == 0) {

fprintf(stdout, "my shell is simple shell.\n");

fprintf(stdout, "12191659 Lee hwa Yeon\n");

}

// 내장형 명령어가 아니라면 execute함수를 호출한다.

else {

execute(arg, arg\_cnt);

}

}

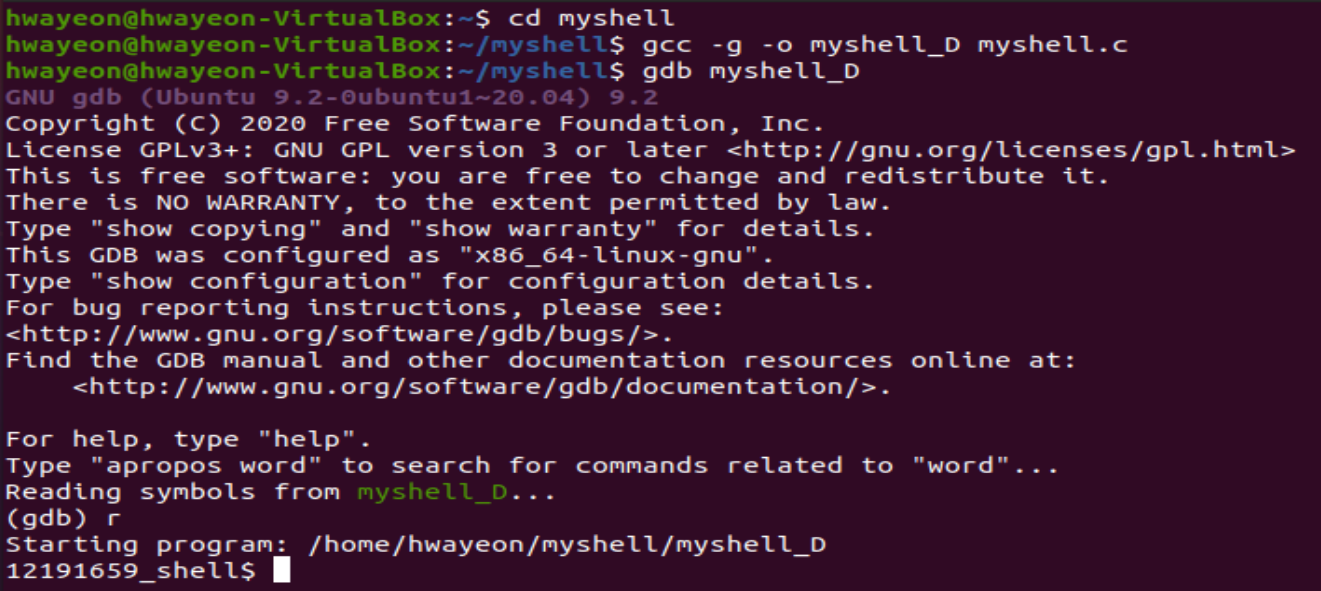
history\_clear(history);

return 0;

}

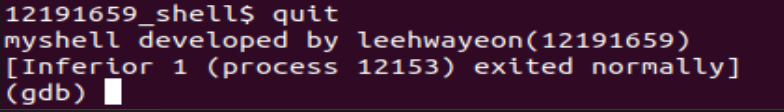
1. **동작과정**

**3-1. 쉘 실행**



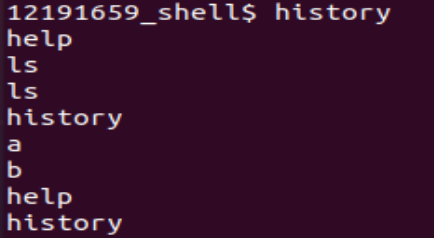
쉘이 시작되면 12191659\_shell$의 프롬포트를 출력한다.

**3-2. quit 명령어**

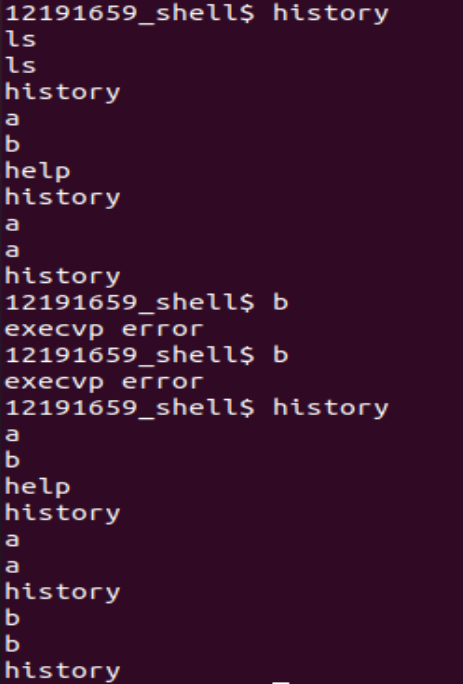


quit 명령어는 쉘프로그램을 종료한다. 종료 시 “myshell developed by leehwayoen(12191659)”를 출력한다.

**3-3. history 명령어**

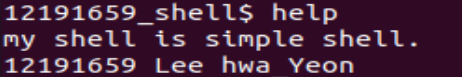


history 명령어는 사용자로부터 최근 입력받은 10개까지의 명령어라인을 출력한다. 10개가 넘으면 처음부터 지워가면서 히스토리를 갱신한다.



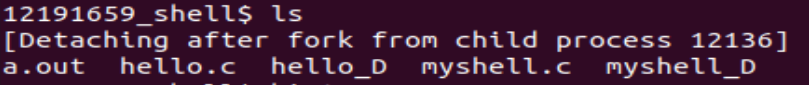
히스토리 명령을 확인해보면 새로운 명령어 b b history를 입력하자 제일 오래 된 명령어는 사라지고 새로운 명령어가 히스토리에 들어온 것을 볼 수 있다. 이는 FIFO(first in first out) 구조이다.

**3-4. help 명령어**



help 명령어는 myshell에 대한 간단한 매뉴얼을 출력한다.

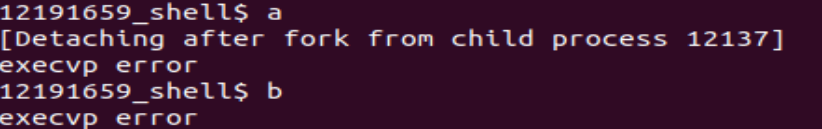
**3-5. 내장형 명령어가 아닌 명령어 ex)ls**



fork 시스템호출을 통해 새로운 프로세스를 생성하고 wait 시스템호출을 통해 프로세스의 종료를 대기한다.

ls 명령어는 디렉토리에 있는 내용을 출력한다.

**3-6. 잘못된 명령어**



잘못된 명령어를 입력하면 에러가 어느 부분에서 났는지 출력한다.

1. **잡지 못한 에러**

4-1. 명령어를 다 입력하고 뒤에 “ “ 공백을 하나 더 입력해야 명령어가 정상적으로 작동한다.

명령어를 파싱할 때 사용하는 parse 함수에서 strtok의 이해도가 낮아 이러한 에러가 발생하는 것 같다.

4-2. 만약 history 명령어 다음 help 명령어를 입력한다면 cmd변수에 helpory와 같이 저장된다. 즉, 명령어를 새로 입력할 때 cmd변수의 내용이 초기화 되지 않아 에러가 발생하는 것이다. char형 배열을 다루는 법이 미숙하여 이러한 에러를 발생시킨 것 같다.