**OS Project #1**

**myshell program 에 대한**

**Readme 보고서**

**컴퓨터공학과**

**12193075 이유진**

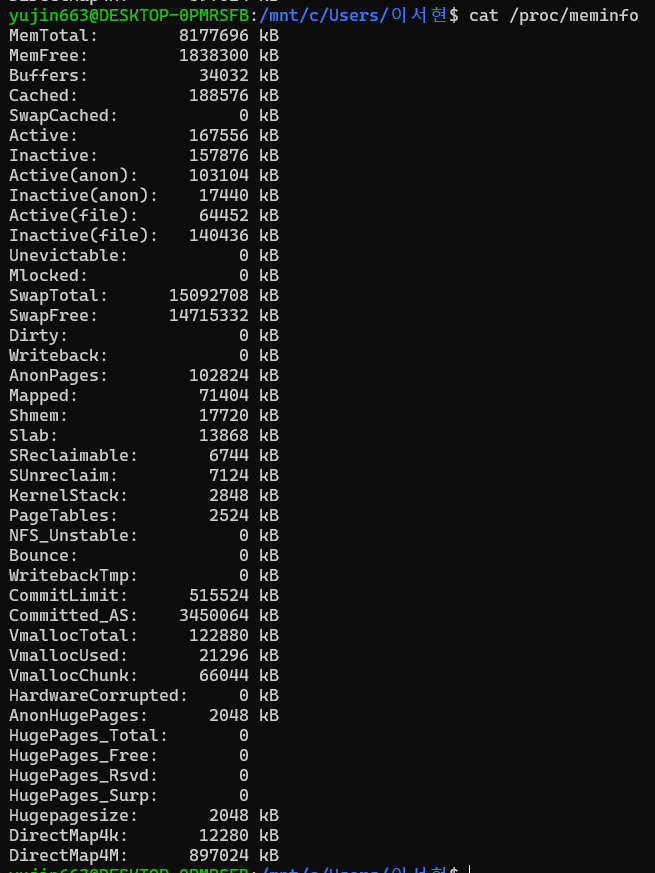
**2021.05.07**

**1) 개발환경**

-CPU 정보

 …

-메모리 관련 정보



-운영체제명, 커널 버전(정보), gcc 컴파일 버전, 생성 날짜



**2) 쉘 프로그램 설계 및 구현 내용 -** 나열한 순서대로 구현하였다.

-변수 정보

instr는 입력 받은 명령어, lines는 입력 받은 모든 명령어를 저장한다.

Count는 입력 받은 명령어의 개수이다.

-내장형 명령어(quit, history, help)

• quit : 쉘 프로그램을 종료한다.

if (strcmp(lines[count], "quit") == 0) {

printf("myshell developed by \"이유진(12193075)\" \n\n");

break; }

strcmp를 통해 quit를 확인하고, break를 통해 프로그램을 종료한다.

• history : 최근 입력 명령어 10개 출력

<history 입력>

else if (strcmp(lines[count], "history") == 0) {

historyp(history, cur);

}

<history 함수>

void seth(char\*\* hist) {

for (int i = 0; i < 10; ++i) hist[i] = NULL;

}

void historyp(char\*\* hist, int c) { //history 출력

int i; int n = 1;

for(i = c-1 ; i>=0; i--){

printf("%d) %s\n",n,hist[i]);

n++;

if(n >10) break;

} printf("\n");

}

history는 따로 함수를 두었다. 배열 history를 통해 명령어를 따로 저장하고 seth 함수로 history배열을 NULL로 두고 historyp 함수로 history를 출력한다. seth는 main 함수 앞부분에 두었다.

• help : 매뉴얼 출력

else if (strcmp(lines[count], "help") == 0) {

printf("NAME \n myshell \n");

printf("DESCRIPTION \n myshell is simple shell which interact by inputting commands.It was made using forkand execvp.There are some bulit - in commands, quit, historyand help. When users input “& ” behind command, it run the process in the background.\n");

}

help 입력시 출력되는 매뉴얼은 myshell에 대한 간단한 설명이다.

-이외의 명령어 처리, 백그라운드 지원

else { char fullpath[20];

char\* tok;

입력받은 명령어 라인을 명령어 토큰으로 파싱한다.

내장형 명령어는 띄어쓰기가 없으므로 여기에 두었다.

tok = strtok(instr, " ");

int n = 0;

while (tok != NULL) { //토큰 나누기

lines[n++] = tok;

tok = strtok(NULL, " ");

}

lines[n] = (char\*)0;

argc = n;

strcpy(fullpath, path);

execvp에 사용할 경로를 만든다.

strcat(fullpath, lines[0]);

for (n = 0; n < strlen(fullpath); n++) {

if (&fullpath[n] == "\n") fullpath[n] = '\0';

}

pid = fork();

int i = 0;

while (lines[i] != NULL) {

if (strcmp(lines[i], "&") == 0) {

&를 입력할 경우 백그라운드 함수로 넘어간다.

background(lines, i);

} i++;

}

Parent process.

if (pid > 0) { //parent

pid = wait(&status);

}

else if (pid == 0) { //child

execvp(fullpath, lines);

Child process.

fork 이후 execvp를 통해 새 프로세스를 만든다.

perror("execvp error");

exit(-1);

}

else {

perror("write error");

exit(-11);

}

printf("\n");

} }

<백그라운드 함수>

void background(char\*\* cmd, int i) {

&는 명령어로 포함시키지 않는다.

cmd[i] = NULL;

int pid2;

pid2 = fork();

자식 프로세스를 기다리지 않는다.

if (pid2 > 0) exit(1);

else if (pid2 == 0) {

execvp(cmd[0], cmd);

}

else {

perror("error");

exit(-11);

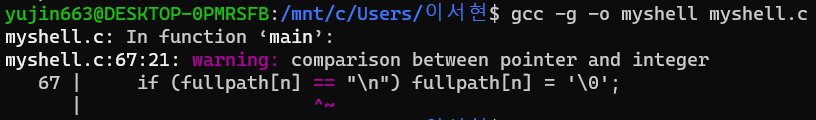
} }

**3) 동작과정**

- GDB를 통한 오류 해결

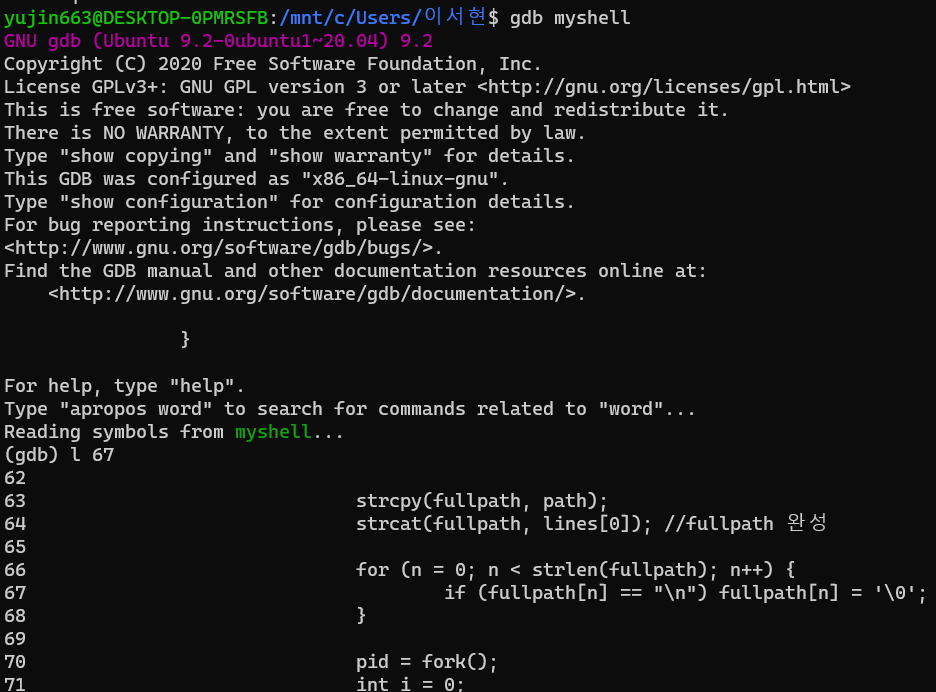
gdb를 통해 오류를 해결한 경우 중 한가지이다.

• 오류 발견



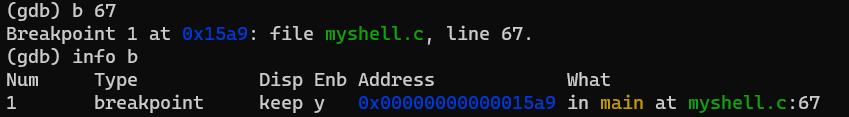
코드를 실행하려 했으나, 67번 줄에서 error가 생긴 것을 발견했다.

• 소스코드 확인



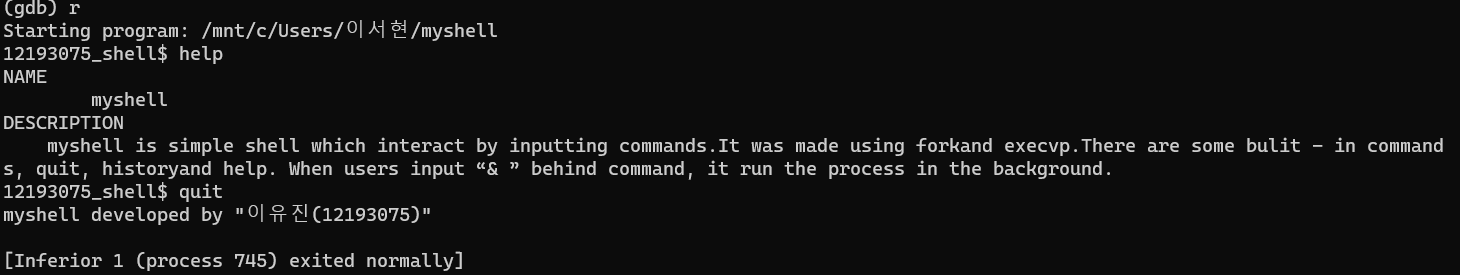
gdb 명령어 ‘l’을 통해 67행 주변의 소스를 확인하였다.

• break point 설정



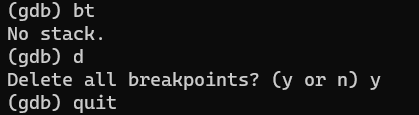
에러가 난 부분에 breakpoint를 설정하였고, 그 breakpoint에 대한 정보이다.

• 프로그램 실행

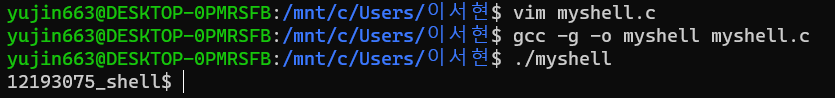


명령어 r을 통해 프로그램을 수행했을 때, 에러가 뜨지 않고 breakpoint 이전의 명령어는 수행되는 것을 확인하였다.

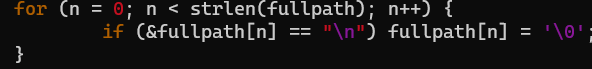
• 에러 정보 확인, breakpoint 삭제



• 소스코드 수정



(vim myshell.c)



fullpath 앞에 &를 두었더니 성공적으로 수행되었다.