# mysh 작성 과제

과목명 | 시스템프로그래밍

담당교수 | 최종무 교수님

학과 | 소프트웨어학과

학번 | 32191197

이름 | 김채은

제출일 | 2020. 10. 29.

### 1. gdb를 이용한 디버깅

```
sys32191197@embedded:~$ vi myshell.c
sys32191197@embedded:~$ gcc -g myshell.c -o mysh
sys32191197@embedded:~$ gdb mysh
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.5) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from mysh...done.
(gdb) run
Starting program: /home/7-class/sys32191197/mysh
 /home/7-class/sys32191197 : mysh $ ls &
 /home/7-class/sys32191197 : mysh $ ls: cannot access '&'$'\n': No such file
or directory
```

백그라운드 프로세싱을 할 때 & 문자가 인자로 인식되어 No such file or directory라는 오류가 발생했습니다. 그래서 수행 전에 line에서 & 문자가 발견된다면 token에서 마지막 인자인 &을 NULL 값으로 바꾸는 코드를 추가했습니다.

### 2. 주요 코드 분석

전체 코드는 압축파일로 묶어 조교님께 전송했습니다.

```
bool cd(int argc, char* argv[]);
bool quit(int argc, char* argv[]);
bool help(int argc, char* argv[]);
struct cmdSet {
	char *name;
	char *info;
	bool (*cmdFunc) (int argc, char *argv[]);
};
struct cmdSet cmds[5] = {
	{"cd", "Change directory", cd},
	{"help", "Show help info.", help},
	{"quit", "Quit my shell", quit},
	{">", "Redirection"},
	{"&", "Background processing"}
};
```

처음에는 name, info, cmdFunc 각각 배열을 선언했다가 cmdFunc에서 함수 포인터 배열 선언을 하고 초기화하는데에서 계속 에러가 발생하여 구조체로 묶어서 선언했다. 프로그램이 훨씬 간결해졌다.

```
bool cd(int argc, char* argv[]) {
	if (argc == 2) {
		if (chdir(argv[1]))
			printf("%s is no directory\n", argv[1]);
	}
	else {
		printf("[!] Enter correct argument: cd [directory name]\n");
	}
	return true;
}
```

cd(change directory 명령어)는 내부 명령어이기 때문에 fork 하지 않고 명령을 실행한 후 종료한다. cd 함수를 사용하면 run 함수에서 fork가 수행되기 전에 true를 리턴하기 때문에 바로 run 함수가 종료되고 fork가 수행되지 않는다.

```
bool quit(int argc, char* argv[]) {
	return false;
}
bool help(int argc, char* argv[]) {
	int i;
	printf("------------------------------------------------------\n");
	printf("Simple shell by. Kimce\n");
	printf("Here's some commands.\n");
	for (i = 0; i < 5; i++) {
		printf("%s: %s\n", cmds[i].name, cmds[i].info);
	}
	printf("------------------------------------------------------\n");
	return true;
}
```

quit 함수는 run 함수가 false를 리턴하게 해서 while문에서 빠져나와 mysh 프로그램을 종료시킨다.

help 함수는 이 쉘에서 사용할 수 있는 명령어를 보여준다.

```
void redir(int argc, char* argv[]) {
	int fd;
	if (argc == 4) {
		fd = open(argv[3], O_WRONLY | O_CREAT, 0664);
		if (fd < 0) {
			printf("[!] file open error: no. %d\n", argv[3], errno);
			exit(-1);
		}
		dup2(fd, STDOUT_FILENO);
		argv[2] = NULL;
	}
	close(fd);
	return;
}
```

argc를 체크하고 dup2를 이용해 output이 표준출력이 아닌 인자로 주어진 파일로 입력되도록 리다이렉션한다.

```
int tokenize(char* buf, char* delims, char* tokens[], int maxTokens) {
	int count = 0;
	char* t;
	t = strtok(buf, delims);
	while (t != NULL && count < maxTokens) {
		tokens[count++] = t;
		t = strtok(NULL, delims);
	}
	tokens[count] = NULL;
	return count;
}
```

토큰화하는 함수이다. 강의자료에 나온 코드를 참고했다.

```
bool run(char* line)
{
	char delims[] = " \n";
	char* tokens[80];
	int i;
	int t_count;
	int status;
	pid_t pid;
	bool doBack = false;
	bool doRedir = false;
	char* isRedir = strchr(line, '>');
	char* isBack = strchr(line, '&');
	if (isRedir != NULL) doRedir = true;
	if (isBack != NULL) doBack = true;
	t_count = tokenize(line, delims, tokens, sizeof(tokens) / sizeof(char*));
	if (t_count == 0)
		return true;
	if(doBack == true)
		tokens[t_count-1] = NULL;
	for (i = 0; i < 3; i++) {
		if (strcmp(cmds[i].name, tokens[0]) == 0)
			return cmds[i].cmdFunc(t_count, tokens);
	}
	if ((pid = fork()) < 0) {
		perror("fork error");
		exit(-1);
	}
	else if (pid == 0) {
		if (doRedir == true) {
			redir(t_count, tokens);
		}
		execvp(tokens[0], tokens);
	}
	if (doBack == false)
		wait(&status);
	return true;
}
		execvp(tokens[0], tokens);
	}
	if (doBack == false)
		wait(&status);

	return true;
}
```

실행함수이다. 전반적인 프로그램 수행이 모두 이루어진다. for문에서 strcmp로 tokens의 첫 문자와 명령어들을 확인하고 겹치는 게 있는지 확인한다. 있다면 특별한 명령어이므로 그에 따른 함수를 호출해주고 특별한 함수에서 true를 리턴해 run을 종료한다. 겹치는 게 없으면 밑으로 내

려와 fork와 exec으로 쉘 명령을 구현한다.

line에서 '&' 문자열이 확인되어 doBack이 true가 되면 tokens 중 맨 뒤에 있는 문자, 즉 &를 NULL로 바꿔준다. 그렇지 않으면 명령어 실행 시 인자에 &이 포함되어 오류가 생긴다. background 수행은 부모 프로세스에서 wait하지 않으면 명령과 shell이 각자 동작하므로 구현이 간단하다. doBack이 false일 때만 wait해주면된다.

redirection은 line에서 '>' 문자열이 확인되어 doRedir이 true가 되면 redir 함수가 호출되며 수행된다. 수행 후에는 true가 리턴 되어 run 함수가 종료된다. false일 때는 동작하지 않는다.

### 3. 실행 화면

(1) help

```
sys32191197@embedded:~$ vi myshell.c
sys32191197@embedded:~$ gcc -o mysh myshell.c
sys32191197@embedded:~$ ./mysh
 /home/7-class/sys32191197 : mysh $ help
------------------------------------------------------
Simple shell by. Kimce
Here's some commands.
cd: Change directory
help: Show help info.
quit: Quit my shell
>: Redirection
&: Background processing
------------------------------------------------------
```

(2) cd

```
 /home/7-class/sys32191197 : mysh $ cd ..
 /home/7-class : mysh $ cd ./sys32191197
 /home/7-class/sys32191197 : mysh $ ps
```

(3) background processing / ps

```
sys32191197@embedded:~$ cat whiletest.c
#include<stdio.h>

int main(){
        while(1);
}
sys32191197@embedded:~$ gcc -o while whiletest.c
sys32191197@embedded:~$ ./mysh
 /home/7-class/sys32191197 : mysh $ ./while &
 /home/7-class/sys32191197 : mysh $ ps
  PID TTY          TIME CMD
 1629 pts/45   00:00:00 bash
 9512 pts/45   00:00:00 mysh
 9846 pts/45   00:00:04 while
 9934 pts/45   00:00:00 ps
```

(4) redirection

```
 /home/7-class/sys32191197 : mysh $ cat my_poem.txt > new.txt
 /home/7-class/sys32191197 : mysh $ cat new.txt
난 내가 오늘 죽었다고 생각했어
무대에서 내려와
하늘에서 열리는 공연을 향해
구름 사이로 올라갔지

그곳에 도착하자
천사들이 이 노래를 부르고 있었어
그래, 네가 아는 그 노래야
너도 따라 부르고 있니?

난 내가 하는 모든 일들이
어떻게든 원하는 대로 풀릴 거라 생각했어
한 황소와 달이 내게 손을 건넸지
그러니 그들과 함께 하는 수밖에
천둥과 번개를 불러 노래하게 해
폭풍의 눈에는 옳고 그름 따윈 없으니까

안녕, 안녕
언젠간 온세상이 내 노래를 부르게 될 거야
여전히 삶은 계속되고
내 심장 속 어딘가에선 박동이 계속되고 있어

The beat goes on, Beady eye
```

(5) quit

```
 /home/7-class/sys32191197 : mysh $ quit
sys32191197@embedded:~$ cat myshell.c
```

**4. Discussion**

처음 수업에서 과제를 들었을 때는 fork(), exec..() 명령어, 심지어는 3장의 리다이렉션에 대한 개념도 완전히 머릿속에 정립되지 않은 상태였어서 굉장히 막막한 기분이었다. 무작정 강의자료를 따라 시도해보려고 해도 이해가 안되는 부분이 너무 많아서 일단 시험 공부를 겸하면서 개념 공부를 다시 하기로 했다. 1장부터 수업 때 휘갈겨둔 필기를 보며(강의 속도 느리게 기능도 필요하지 않을까 싶다...) 개념 공부를 다시 했다. 1장을 들을 때는 이전에 배웠던 컴파일레이션 시스템을 제외하고는 이해가 가는 내용이 정말 없이 그냥 받아적기만 했는데, 다시 돌아가서 보니 2, 3, 4, 5장에 나온 내용들이 똑같은 말로 한 번씩 모두 언급이 되어있어서 신기했다. 3장의 리다이렉션 부분은 mysh 과제에서 사용해야 되는 것을 떠올리며 더 정확히 이해하려고 노력했다. 사실 전에 리다이렉션과 관련한 출석 문제를 풀 때는 디바이스 파일이나 터미널에 대해 이해를 제대로 못해서 틀렸는데(...) 다시 공부해보니 왜 틀렸지 싶었다. 앞으로는 처음 강의를 들을 때 주의를 조금 더 기울여야겠다.

개념을 이해한 뒤에도 쉘 구현이 마냥 쉽지는 않았다. 먼저 인터넷에 나와있는 코드 자료들을 훑어보면서 감을 잡았다. 단국대 선배들의 과제 코드가 굉장히 많았다. 대충 감을 잡은 후에 강의에서 소개된 선경문 선배의 코드를 참고하며 틀을 잡았다. 리다이렉션과 백그라운드 프로세싱은 일단 무시하고 cd, tokenize, run 함수를 제대로 구현하는 걸 우선으로 삼았다. 괜찮은 쉘이 완성되고나서 비교적 쉬운 백그라운드 프로세싱을 구현했는데, 막히는 부분은 gdb로 코드를 돌려보고 해결했다. 리다이렉션은 3장 내용을 계속 들여다보면서 코드를 작성했다. run을 제외하고 가장 오래 걸렸던 것 같다.

이번 프로그램을 만들면서 가장 힘들었던 점은 사실 코드 작성이 아니라 putty 오류였다... 로그인 창이 안 뜰 때는 그동안 편집해둔 게 다 날아간 건가...? 하는 생각이 들어서 정말 충격이었고, vi 편집기로 코드를 수정하는 도중에도 한 번 같은 상황이 벌어져 코드를 날려먹기도 했다... 선배들 말로는 좀비 프로세스 때문이라고 한다. 결국 중간쯤부터 다시 시작했지만 다시 코드를 짜면서 더 좋은 방법을 찾았다는 사실로 위안을 삼았다. (^^) 과제 난이도부터 putty 문제까지... 아주 다사다난한 과제였다. 하지만 결과물을 보니(완벽하진 않을지라도) 내가 만든 쉘이 이렇게 잘 동작한다는 게 굉장히 신기했고 뿌듯했다.