Operating System Project #1 Report

myshell Program

12191656 이채연

I. 개발환경

Myshell 프로그램을 구현하기 위해서 Window 내에서 가상머신을 이용하여 Linux를 설치하였습니다. VirtualBox 라는 가상머신을 이용하였고, VirtualBox 6.1.22 platform packages 버전을 사용하였습니다. 그 다음, Ubuntu 20.04.2.0 버전의 리눅스 설치 파일을 받은 뒤, 가상머신을 실행하여 "Linux_Ubuntu" 라는 이름의 새로운 가상머신을 실행하여 리눅스를 설치하였습니다.
이렇게 실행한 Linux Os에서의 CPU 정보, memory 정보, 그리고 Linux 배포판 정보 및 커널 정보는 다음과 같습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 61
model name      : Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
stepping        : 4
cpu MHz         : 2194.918
cache size      : 3072 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 20
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
 pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_g
ood nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq monitor ssse3
 cx16 pcid sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor la
hf_lm abm 3dnowprefetch invpcid_single pti fsgsbase avx2 invpcid rdseed md_clea
r flush_l1d
bugs            : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass l1tf mds
 swapgs itlb_multihit srbds
bogomips        : 4389.83
clflush size    : 64
cache_alignment : 64
address sizes   : 39 bits physical, 48 bits virtual
power management:
```

```
chaeyeon@chaeyeon-VirtualBox:~$ cat /proc/meminfo
MemTotal:        1004312 kB
MemFree:           72280 kB
MemAvailable:     252880 kB
Buffers:           14952 kB
Cached:           286568 kB
SwapCached:        12516 kB
Active:           501628 kB
Inactive:         289896 kB
Active(anon):     294400 kB
Inactive(anon):   204804 kB
Active(file):     207228 kB
Inactive(file):    85092 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:        459260 kB
SwapFree:         383444 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:        483936 kB
Mapped:           163844 kB
Shmem:              9200 kB
KReclaimable:      31584 kB
Slab:              88276 kB
SReclaimable:      31584 kB
SUnreclaim:        56692 kB
KernelStack:        6268 kB
PageTables:        11232 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      961416 kB
Committed_AS:    3354328 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       30524 kB
VmallocChunk:          0 kB
Percpu:              644 kB
HardwareCorrupted:     0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
FileHugePages:         0 kB
FilePmdMapped:         0 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:               0 kB
DirectMap4k:      139200 kB
DirectMap2M:      909312 kB
```

```
chaeyeon@chaeyeon-VirtualBox:~$ cat /proc/version
Linux version 5.8.0-50-generic (buildd@lgw01-amd64-030) (gcc (Ubuntu 9.3.0-17ub
untu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #56~20.04.1-Ubuntu S
MP Mon Apr 12 21:46:35 UTC 2021
```

II. 쉘 프로그램 설계 및 구현 내용

이번 프로젝트는 터미널에서 사용자로부터 명령어(command)를 입력 받아 상호작용하는 myshell 프로그램을 구현하는 것입니다.

먼저 리눅스에서 myshell.c 와 myshell.h 라고 하는 c 소스파일과 c 헤더파일을 만들었습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ vi myshell.h
chaeyeon@chaeyeon-VirtualBox:~$ vi myshell.c
chaeyeon@chaeyeon-VirtualBox:~$ ls
a.out     Documents  Makefile  myshell.c  myshell.o    Pictures  Templates
Desktop   Downloads  Music     myshell.h  myshell.out  Public    Videos
```

그리고 make 라고 하는 파일 관리 유틸리티를 사용하기 위해서 Makefile을 만들었습니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ vi Makefile
```

Makefile 코드는 다음과 같습니다.

```
CC = gcc
TARGET = myshell.out
OBJS = myshell.o

CFLAGS = -Wall
LDFLAGS = -lc

all : $(TARGET)

$(TARGET) : $(OBJS)
	$(CC) $(LDFLAGS) -o $@ $^

.c.o :
	$(CC) $(CFLAGS) -c -o $@ $<

clean :
	rm -f $(OBJS) $(TARGET)
```

Target 파일을 myshell.out으로 정하고 이 바이너리 파일을 만들기 위한 목적파일인 myshell.o을 만들어주었습니다.
맨 위에는 all option을 통해 최종으로 만들고자 하는 것이 myshell.out이라는 것을 표시합니다.

myshell.h 코드는 다음과 같습니다.

```
#include<stdio.h>
#include<string.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 10

void tokenize(char*);
int parse_background();
void cmd_quit();
void cmd_history();
void cmd_help();
```

myshell.h 헤더파일에는 다양한 헤더파일과 함수의 헤더(Header)가 선언되어 있습니다.

제가 구현한 5가지 함수의 바디(Body)와 main 함수는 myshell.c 소스파일에 구현되어 있습니다.

1) tokenize(char*)
사용자에게 입력 받은 명령어 라인을 읽고 명령어 토큰으로 파싱하는 함수입니다.
공백문자, tab 문자, 개행 문자를 구분자로 지정하여 strtok 함수를 호출합니다. ptr 변수에는 구분자가 나타나기 전까지의 문자열이 저장됩니다. arg 배열에 이 파싱된 명령어를 넣고 파싱된 아이템의 개수를 담고 있는 argv 변수의 값을 하나 올려줍니다.

```
void tokenize(char *command){
        char *ptr = strtok(command, " \t\n");
        while (ptr != NULL){
                arg[argv++] = ptr;
                ptr = strtok(NULL, " \t\n");
        }
        arg[argv] = (char*)0;
        return;
}
```

2) parse_background()
명령어 라인 끝에 백그라운드를 표현하는 &기호가 있을 때 이 기호를 명령어 토큰에 포함시키지 않고, 백그라운드 실행이면 1을 반환하고 백그라운드 실행이 아니면 0을 반환하는 함수입니다. command 라는 char 배열 변수에서 개행 문자가 들어있는 인덱스의 바로 전 인덱스에 '&' 문자가 들어가 있는지 확인하고 '&'문자가 들어가 있으면 그 인덱스에 ' '라는 공백 문자를 저장하고 백그라운드 실행이므로 1을 리턴합니다. '&'문자가 들어가있지 않으면 백그라운드 실행이 아니므로 0을 리턴합니다.

```
int parse_background(){
        int end;
        for(int i = 0; i <256 ; i++){
                if(command[i] == '\n'){
                        end = i;
                        break;
                }
        }
        if(command[end-1] == '&'){
                command[end-1] = ' ';
                return 1;
        }
        else{
                return 0;
        }
}
```

3) cmd_quit()
내장형 명령어인 quit이 입력으로 들어왔을 때 적절한 메시지를 출력하고 쉘프로그램을 종료하는 함수입니다.

```
void cmd_quit(){
        printf("myshell developed by ChaeYeonLee(12191656).\n");
        exit(1);
}
```

4) cmd_history()
내장형 명령어인 history이 입력으로 들어왔을 때 사용자로부터 최근 입력 받은 10개까지의 명령어 라인을 출력하는 함수입니다. command_list[10] 배열에 저장 되어있는 최근 입력 받은 명령어들을 for문을 통해 출력합니다.

```
void cmd_history(){
        if(full==false){
                for(int i = 0; i <p-1;i++){
                        printf("%d       %s\n", i+1, command_list[i]);
                }
                printf("%d       %s", p, command_list[p-1]);
        }
        else{
                for(int i = 0; i <9; i++){
                        printf("%d       %s\n", i+1, command_list[p]);
                        p = (p+1)%10;
                }
                printf("10       %s", command_list[p]);
        }
}
```

5) cmd_help()
내장형 명령어인 help가 입력으로 들어왔을 때 3가지 내장형 명령어에 대한 간단한 매뉴얼을 출력하는 함수입니다.

```
void cmd_help(){
        printf("DESCRIPTION\nMyshell is a simple shell.\nThe following are buil
t-in commands.\nquit : Shut down the shell program.\"myshell developed by ChaeY
eonLee(12191656)\" is displayed at the end.\nhistory : It outputs up to 10 comm
and lines that were recently entered by the user.\nhelp : Print out a simple ma
nual.\n");
}
```

다음은 main 함수에 대한 설명입니다.

사용자가 입력한 명령어 라인들을 저장하는 배열인 command_list[10]를 동적할당 해줍니다.

```
int main(){

        for(int i = 0; i < 10; i++){
                command_list[i] = malloc(256);
        }
```

그리고 while문을 돌면서 프롬프트(12191656_myshell$)을 먼저 출력하고 사용자로부터 명령어를 대기합니다. quit 명령어가 오기 전까지 while문을 돌면서 사용자로부터 명령어를 계속 입력 받습니다. 명령어를 입력 받으면 먼저 parse_background()함수를 호출하여 이 명령어가 백그라운드 실행인지 확인합니다. 그리고 tokenize()함수를 호출하여 입력받은 명령어를 파싱하여 arg[MAX] 배열에 저장합니다.
파싱된 첫번째 아이템이 내장형 명령어인 quit, history, help 중 하나이면 fork없이 각각 cmd_quit(), cmd_history(), cmd_help() 함수를 호출합니다.

```
    while(1){

            printf("12191656_shell$ ");
            argv = 0;
            fgets(command,256,stdin);

            strcpy(command_list[p], command);
            if(p==9){full = true;}
            p = (p+1)%10;

            isBackground = parse_background();
            tokenize(command);

            if(strcmp(arg[0], "quit")==0){
                    cmd_quit();
            }
            else if(strcmp(arg[0], "history")==0){
                    cmd_history();
                    continue;
            }
            else if(strcmp(arg[0], "help")==0){
                    cmd_help();
                    continue;
            }
```

만약 내장형 명령어가 아니라면 fork 시스템 호출을 통해 새로운 프로세스를 생성하고, parent process는 wait system call을 통해 child process가 execvp를 실행하는 동안 child process의 종료를 대기합니다. 백그라운드 실행(&) 인 경우에는 waitpid 함수의 WNOHANG 명령어를 사용하여 parent process가 child process를 기다리지 않도록 합니다. Parent process는 실행이 끝나면 “[pid] : done!” 을 출력합니다.
그리고 새로 생성된 child process는 입력 받은 명령어를 실행합니다.

```
            else{
                    if((pid=fork())== -1){
                            perror("fork failed");
                    }
                    else if (pid != 0){
                            printf("[Parent]pid = %d\n", pid);
                            if(isBackground==1){
                                    waitpid(pid, &status, WNOHANG);
                            }
                            else{
                                    wait(&status);
                            }
                            printf("[%d] : done!\n", pid);
                    }
                    else{
                            execvp(arg[0], arg);
                    }
            }
    }
```

마지막으로 동적할당 받은 command_list[10] 배열을 해제시켜주면서 main 함수가 끝나게 됩니다.

```
        for(int i = 0; i < 10; i++){
                free(command_list[i]);
        }
}
```

III. 동작 과정(GNU gdb 명령어 설명 및 캡처사진)

myshell Program 실행 결과입니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ make
make: Nothing to be done for 'all'.
chaeyeon@chaeyeon-VirtualBox:~$ make clean
rm -f myshell.o myshell.out
chaeyeon@chaeyeon-VirtualBox:~$ make
gcc -Wall -c -o myshell.o myshell.c
gcc -lc -o myshell.out myshell.o
chaeyeon@chaeyeon-VirtualBox:~$ ./myshell.out
12191656_shell$ help
DESCRIPTION
Myshell is a simple shell.
The following are built-in commands.
quit : Shut down the shell program."myshell developed by ChaeYeonLee(12191656)"
 is displayed at the end.
history : It outputs up to 10 command lines that were recently entered by the u
ser.
help : Print out a simple manual.
12191656_shell$ sleep 5
[Parent]pid = 2165
[2165] : done!
12191656_shell$ sleep 5 &
[Parent]pid = 2166
[2166] : done!
12191656_shell$ ls -l&
[Parent]pid = 2168
[2168] : done!
```

```
12191656_shell$ total 96
-rwxrwxr-x 1 chaeyeon chaeyeon 21744  5월  7 06:19 a.out
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Desktop
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Documents
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Downloads
-rw-rw-r-- 1 chaeyeon chaeyeon   214  5월  7 04:41 Makefile
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Music
-rw-rw-r-- 1 chaeyeon chaeyeon  2157  5월  7 14:22 myshell.c
-rw-rw-r-- 1 chaeyeon chaeyeon   255  5월  7 13:38 myshell.h
-rw-rw-r-- 1 chaeyeon chaeyeon  6576  5월  7 14:25 myshell.o
-rwxrwxr-x 1 chaeyeon chaeyeon 17744  5월  7 14:25 myshell.out
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  7 02:59 Pictures
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Public
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Templates
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Videos
history
1       help
2       sleep 5
3       sleep 5 &
4       ls -l&
5       history
12191656_shell$ quit
myshell developed by ChaeYeonLee(12191656).
chaeyeon@chaeyeon-VirtualBox:~$
```

동작 과정을 설명하겠습니다.
컴파일 시에 디버깅 정보를 담기 위해 디버깅 옵션인 -g으로 컴파일합니다. 그리고 실행시킵니다.

```
chaeyeon@chaeyeon-VirtualBox:~$ vi myshell.c
chaeyeon@chaeyeon-VirtualBox:~$ gcc -g myshell.c
chaeyeon@chaeyeon-VirtualBox:~$ gdb a.out
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 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 a.out...
```

gdb의 명령어 l은 소스의 내용을 출력합니다. 그리고 명령어 b를 통해 70번째 줄에 breakpoint를 설정합니다.

```
(gdb) l
50              }
51              else{
52                      for(int i = 0; i <9; i++){
53                              printf("%d      %s", i+1, command_list[p]);
54                              p = (p+1)%10;
55                      }
56                      printf("10      %s", command_list[p]);
57              }
58      }
59
(gdb) l
60      void cmd_help(){
61              printf("DESCRIPTION\nMyshell is a simple shell.\nThe following
are built-in commands.\nquit : Shut down the shell program.\"myshell developed
by ChaeYeonLee(12191656)\" is displayed at the end.\nhistory : It outputs up to
 10 command lines that were recently entered by the user.\nhelp : Print out a s
imple manual.\n");
62      }
63
64      int main(){
65
66              for(int i = 0; i < 10; i++){
67                      command_list[i] = malloc(256);
68              }
69
(gdb) b 70
Breakpoint 1 at 0x159e: file myshell.c, line 72.
```

첫번째로 help 라는 명령어 라인을 읽습니다. 명령어 p를 통해 command 변수의 정보를 보면

help\n 라는 문자열이 저장 되어있고 이 문자열을 command_list 배열의 첫번째 인덱스에 저장합니다. 이 명령어 라인은 백그라운드 실행이 아니기 때문에 parse_background() 함수를 실행하고 나면 isBackground 변수에 0이 저장되어 있음을 확인할 수 있습니다.

```
(gdb) run
Starting program: /home/chaeyeon/a.out

Breakpoint 1, main () at myshell.c:72
72                      printf("12191656_shell$ ");
(gdb) n
73                      argv = 0;
(gdb) n
74                      fgets(command,256,stdin);
(gdb) n
12191656_shell$ help
76                      strcpy(command_list[p], command);
(gdb) n
77                      if(p==9){full = true;}
(gdb) p command
$1 = "help\n", '\000' <repeats 250 times>
(gdb) p command_list
$2 = {0x5555555592a0 "help\n", 0x5555555593b0 "", 0x5555555594c0 "",
  0x5555555595d0 "", 0x5555555596e0 "", 0x5555555597f0 "", 0x555555559900 "",
  0x555555559a10 "", 0x555555559b20 "", 0x555555559c30 ""}
(gdb) n
78                      p = (p+1)%10;
(gdb) n
80                      isBackground = parse_background();
(gdb) n
81                      tokenize(command);
(gdb) p isBackground
$3 = 0
```

이 명령어 라인이 tokenize() 함수가 호출되고 나면 띄어쓰기를 구분자로 하여 파싱되어 arg 배열에 저장됩니다. 그리고 arg의 첫번째 인덱스의 문자열이 help 이므로 cmd_help() 함수를 호출합니다. 간단한 매뉴얼이 출력됩니다.

```
(gdb) n
83                      if(strcmp(arg[0], "quit")==0){
(gdb) p arg
$4 = {0x5555555580c0 <command> "help", 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
  0x0, 0x0}
(gdb) n
86                      else if(strcmp(arg[0], "history")==0){
(gdb) n
90                      else if(strcmp(arg[0], "help")==0){
(gdb) n
91                              cmd_help();
(gdb) s
cmd_help () at myshell.c:60
60      void cmd_help(){
(gdb) n
61              printf("DESCRIPTION\nMyshell is a simple shell.\nThe following
are built-in commands.\nquit : Shut down the shell program.\"myshell developed
by ChaeYeonLee(12191656)\" is displayed at the end.\nhistory : It outputs up to
 10 command lines that were recently entered by the user.\nhelp : Print out a s
imple manual.\n");
(gdb) n
DESCRIPTION
Myshell is a simple shell.
The following are built-in commands.
quit : Shut down the shell program."myshell developed by ChaeYeonLee(12191656)"
 is displayed at the end.
history : It outputs up to 10 command lines that were recently entered by the u
ser.
help : Print out a simple manual.
62      }
(gdb) n
main () at myshell.c:92
92                              continue;
(gdb) n
72                      printf("12191656_shell$ ");
(gdb) n

Breakpoint 1, main () at myshell.c:72
72                      printf("12191656_shell$ ");
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) n
73                      argv = 0;
(gdb) n
74                      fgets(command,256,stdin);
(gdb) n
12191656_shell$ sleep 5
76                      strcpy(command_list[p], command);
(gdb) n
77                      if(p==9){full = true;}
(gdb) n
78                      p = (p+1)%10;
(gdb) n
80                      isBackground = parse_background();
(gdb) n
81                      tokenize(command);
```

다음 명령어 라인으로 sleep 5 가 입력됩니다. 이 명령어 라인이 tokenize() 함수 호출을 통해 arg에 sleep 과 5로 파싱되어 들어가있음을 확인할 수 있습니다. sleep은 내장형 명령어가 아니고 백그라운드 실행이 아니기 때문에 child process를 생성하고 child process가 명령어를 실행하는 동안 parent process는 wait system call을 통해 child process를 기다립니다. 그래서 실제로 [parent_pid] : done! 이라는 문자열이 5초 후에 출력됩니다.

```
(gdb) n
83                      if(strcmp(arg[0], "quit")==0){
(gdb) p arg
$5 = {0x5555555580c0 <command> "sleep", 0x5555555580c6 <command+6> "5", 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
(gdb) n
86                      else if(strcmp(arg[0], "history")==0){
(gdb) n
90                      else if(strcmp(arg[0], "help")==0){
(gdb) n
95                              if((pid=fork())== -1){
(gdb) n
[Detaching after fork from child process 2221]
98                              else if (pid != 0){
(gdb) n
99                                      printf("[Parent]pid = %d\n", pid);
(gdb) n
[Parent]pid = 2221
100                                     if(isBackground==1){
(gdb) n
104                                             wait(&status);
(gdb) n
106                                     printf("[%d] : done!\n", pid);
(gdb) n
[2221] : done!
```

다음 명령어 라인으로 sleep 5 &가 들어옵니다. 이 명령어 라인은 백그라운드 실행이기 때문에 parent process가 child process를 기다리지 않습니다. 그래서 실제로 [parent_pid] : done! 이라는 문자열이 5초 후가 아닌 바로 출력됩니다.

```
72                      printf("12191656_shell$ ");
(gdb) n
73                      argv = 0;
(gdb) n
74                      fgets(command,256,stdin);
(gdb) n
12191656_shell$ sleep 5 &
76                      strcpy(command_list[p], command);
(gdb) n
77                      if(p==9){full = true;}
(gdb) n
78                      p = (p+1)%10;
(gdb) n
80                      isBackground = parse_background();
(gdb) n
81                      tokenize(command);
(gdb) n
83                      if(strcmp(arg[0], "quit")==0){
(gdb) n
86                      else if(strcmp(arg[0], "history")==0){
(gdb) n
90                      else if(strcmp(arg[0], "help")==0){
(gdb) n
95                              if((pid=fork())== -1){
(gdb) n
[Detaching after fork from child process 2222]
98                              else if (pid != 0){
(gdb) n
99                                      printf("[Parent]pid = %d\n", pid);
(gdb) n
[Parent]pid = 2222
100                                     if(isBackground==1){
(gdb) n
101                                             waitpid(pid, &status, WNOHANG);
(gdb) n
106                                     printf("[%d] : done!\n", pid);
(gdb) n
[2222] : done!
72                      printf("12191656_shell$ ");
(gdb) n
73                      argv = 0;
(gdb) n
74                      fgets(command,256,stdin);
(gdb) n
12191656_shell$ ls -l&
76                      strcpy(command_list[p], command);
(gdb) n
77                      if(p==9){full = true;}
(gdb) n
78                      p = (p+1)%10;
(gdb) n
80                      isBackground = parse_background();
```

다음 명령어 라인으로 ls -l&이 들어옵니다. parse_background 함수를 호출하면 command 배열에서 ‘&’가 들어가 있는 인덱스를 찾아서 그 인덱스에 ‘ ’를 저장합니다. 그래서 parse_background 함수 호출 후에는 command 변수에 “ls -l \n” 문자열이 저장됩니다.

```
(gdb) s
parse_background () at myshell.c:22
22      int parse_background(){
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
```

```
(gdb) n
24              for(int i = 0; i <256 ; i++){
(gdb) n
25                      if(command[i] == '\n'){
(gdb) n
26                              end = i;
(gdb) n
27                              break;
(gdb) n
30              if(command[end-1] == '&'){
(gdb) n
31                      command[end-1] = ' ';
(gdb) n
32                      return 1;
(gdb) n
37      }
(gdb) n
main () at myshell.c:80
80                      isBackground = parse_background();
(gdb) p command
$6 = "ls -l \n\000 \n", '\000' <repeats 245 times>
(gdb) p isBackground
$7 = 1
(gdb) n
81                      tokenize(command);
```

그리고 tokenize 함수가 호출되면 “ls” 와 “-l” 로 파싱되어 arg 배열에 들어갑니다. 내장형 명령어가 아니기 때문에 fork system call을 하게 됩니다.

```
(gdb) s
tokenize (
    command=0x5555555557a0 <__libc_csu_init> "\363\017\036\372AWL\215=\253%")
    at myshell.c:12
12      void tokenize(char *command){
(gdb) n
13              char *ptr = strtok(command, " \t\n");
(gdb) n
14              while (ptr != NULL){
(gdb) n
15                      arg[argv++] = ptr;
(gdb) p ptr
$8 = 0x5555555580c0 <command> "ls"
(gdb) n
16                      ptr = strtok(NULL, " \t\n");
(gdb) n
14              while (ptr != NULL){
(gdb) n
15                      arg[argv++] = ptr;
(gdb) p ptr
$9 = 0x5555555580c3 <command+3> "-l"
(gdb) n
16                      ptr = strtok(NULL, " \t\n");
(gdb) n
14              while (ptr != NULL){
(gdb) n
18              arg[argv] = (char*)0;
(gdb) p ptr
$10 = 0x0
```

```
(gdb) n
19              return;
(gdb) n
20      }
(gdb) n
main () at myshell.c:83
83                      if(strcmp(arg[0], "quit")==0){
(gdb) p arg
$11 = {0x5555555580c0 <command> "ls", 0x5555555580c3 <command+3> "-l", 0x0,
  0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
(gdb) n
86                      else if(strcmp(arg[0], "history")==0){
(gdb) n
90                      else if(strcmp(arg[0], "help")==0){
(gdb) n
95                              if((pid=fork())== -1){
(gdb) n
[Detaching after fork from child process 2225]
98                              else if (pid != 0){
(gdb) total 96
-rwxrwxr-x 1 chaeyeon chaeyeon 21776  5월  7 14:28 a.out
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Desktop
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Documents
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Downloads
-rw-rw-r-- 1 chaeyeon chaeyeon   214  5월  7 04:41 Makefile
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Music
-rw-rw-r-- 1 chaeyeon chaeyeon  2157  5월  7 14:28 myshell.c
-rw-rw-r-- 1 chaeyeon chaeyeon   255  5월  7 13:38 myshell.h
-rw-rw-r-- 1 chaeyeon chaeyeon  6576  5월  7 14:25 myshell.o
-rwxrwxr-x 1 chaeyeon chaeyeon 17744  5월  7 14:25 myshell.out
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  7 02:59 Pictures
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Public
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Templates
drwxr-xr-x 2 chaeyeon chaeyeon  4096  5월  5 17:46 Videos
n
99                                      printf("[Parent]pid = %d\n", pid);
(gdb) n
[Parent]pid = 2225
100                                     if(isBackground==1){
(gdb) n
101                                             waitpid(pid, &status, WNOHANG);
(gdb) n
106                                     printf("[%d] : done!\n", pid);
(gdb) n
[2225] : done!
```

다음 명령어 라인으로 history가 입력되면 cmd_history 함수가 호출됩니다.
Gdb의 p 명령어를 통해 command_list 배열 변수의 정보를 보면 그동안 입력되었던 명령어 라인의 문자열이 순서대로 저장되어 있습니다. 그래서 cmd_history 함수가 호출되면 최근에 입력받은 명령어라인들이 출력됩니다.

```
72                      printf("12191656_shell$ ");
(gdb) n
73                      argv = 0;
(gdb) n
74                      fgets(command,256,stdin);
(gdb) n
12191656_shell$ history
76                      strcpy(command_list[p], command);
(gdb) n
77                      if(p==9){full = true;}
(gdb) p command_list
$12 = {0x5555555592a0 "help\n", 0x5555555593b0 "sleep 5\n",
  0x5555555594c0 "sleep 5 &\n", 0x5555555595d0 "ls -l&\n",
  0x5555555596e0 "history\n", 0x5555555597f0 "", 0x555555559900 "",
  0x555555559a10 "", 0x555555559b20 "", 0x555555559c30 ""}
(gdb) n
78                      p = (p+1)%10;
(gdb) n
80                      isBackground = parse_background();
(gdb) n
81                      tokenize(command);
(gdb) n
83                      if(strcmp(arg[0], "quit")==0){
(gdb) n
86                      else if(strcmp(arg[0], "history")==0){
(gdb) n
87                              cmd_history();
```

```
(gdb) s
cmd_history () at myshell.c:44
44      void cmd_history(){
(gdb) n
45              if(full==false){
(gdb) n
46                      for(int i = 0; i <p-1;i++){
(gdb) n
47                              printf("%d      %s", i+1, command_list[i]);
(gdb) n
1       help
46                      for(int i = 0; i <p-1;i++){
(gdb) n
47                              printf("%d      %s", i+1, command_list[i]);
(gdb) n
2       sleep 5
46                      for(int i = 0; i <p-1;i++){
(gdb) n
47                              printf("%d      %s", i+1, command_list[i]);
(gdb) n
3       sleep 5 &
46                      for(int i = 0; i <p-1;i++){
(gdb) n
47                              printf("%d      %s", i+1, command_list[i]);
(gdb) n
4       ls -l&
46                      for(int i = 0; i <p-1;i++){
(gdb) n
49                      printf("%d      %s", p, command_list[p-1]);
(gdb) n
5       history
58      }
(gdb) n
main () at myshell.c:88
88                              continue;
```

마지막으로 쉘 프로그램을 종료하는 명령어인 quit이 명령어 라인으로 입력되면 cmd_quit 함수가 호출되고 exit함수를 통해 쉘 프로그램을 종료합니다.

```
(gdb) n
72                      printf("12191656_shell$ ");
(gdb) n
73                      argv = 0;
(gdb) n
74                      fgets(command,256,stdin);
(gdb) n
12191656_shell$ quit
76                      strcpy(command_list[p], command);
(gdb) n
77                      if(p==9){full = true;}
(gdb) n
78                      p = (p+1)%10;
(gdb) p command_list
$13 = {0x5555555592a0 "help\n", 0x5555555593b0 "sleep 5\n",
  0x5555555594c0 "sleep 5 &\n", 0x5555555595d0 "ls -l&\n",
  0x5555555596e0 "history\n", 0x5555555597f0 "quit\n", 0x555555559900 "",
  0x555555559a10 "", 0x555555559b20 "", 0x555555559c30 ""}
(gdb) n
80                      isBackground = parse_background();
(gdb) n
81                      tokenize(command);
(gdb) n
83                      if(strcmp(arg[0], "quit")==0){
(gdb) n
84                              cmd_quit();
(gdb) s
cmd_quit () at myshell.c:39
39      void cmd_quit(){
(gdb) n
40              printf("myshell developed by ChaeYeonLee(12191656).\n");
(gdb) n
myshell developed by ChaeYeonLee(12191656).
41              exit(1);
(gdb) n
[Inferior 1 (process 2229) exited with code 01]
(gdb) n
The program is not being run.
(gdb) quit
chaeyeon@chaeyeon-VirtualBox:~$
```