Myshell Project.1
readme file

과목	오퍼레이팅시스템
학과	컴퓨터공학과
학번	12161095
이름	강석진
제출 날짜	2021.05.04

1) 개발환경

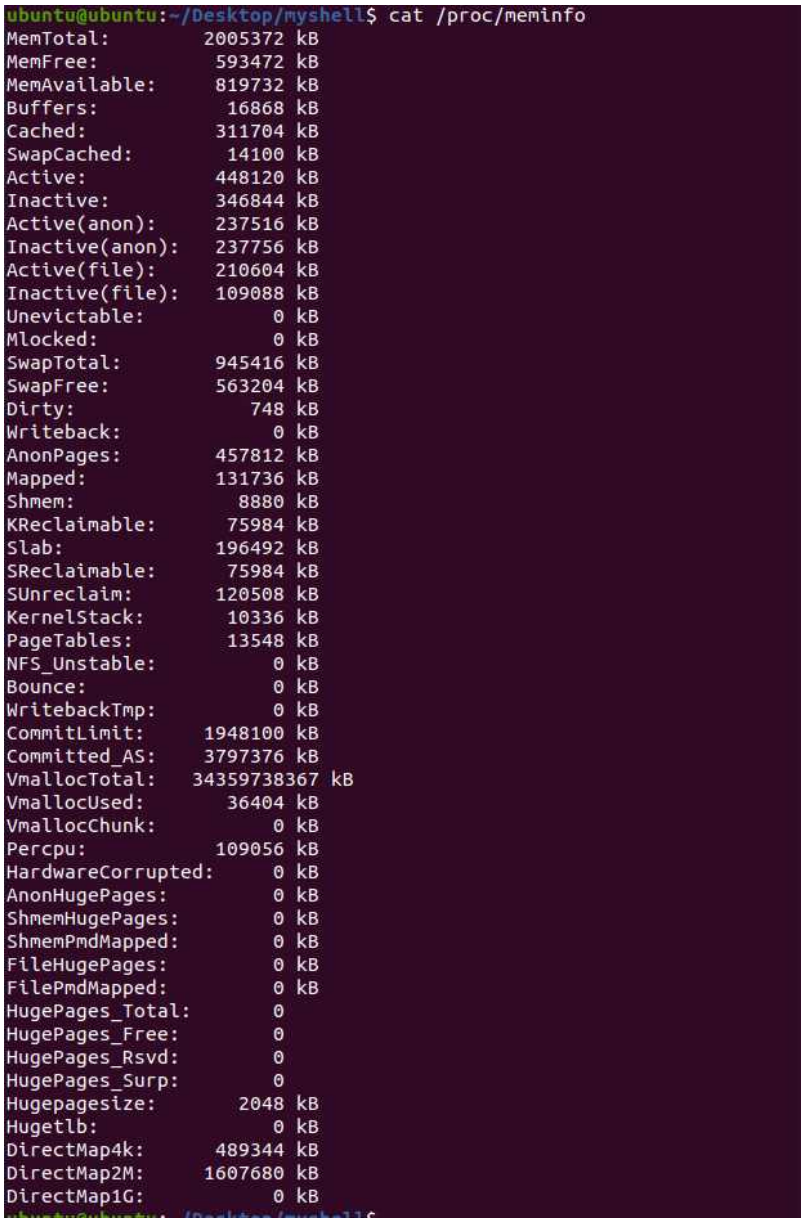
1-1) cpuinfo

```
ubuntu@ubuntu:~/Desktop/myshell$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 126
model name      : Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
stepping        : 5
microcode       : 0x78
cpu MHz         : 1497.602
cache size      : 8192 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 27
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 ss syscall nx pdpe1gb r
dtscp lm constant_tsc arch_perfmon rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x
2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single ssbd ibrs i
bpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt avx512c
d sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves arat avx512vbmi umip pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx5
12_bitalg avx512_vpopcntdq rdpid fsrm md_clear flush_l1d arch_capabilities
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
bogomips        : 2995.20
clflush size    : 64
cache_alignment : 64
address sizes   : 45 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 126
model name      : Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz
stepping        : 5
microcode       : 0x78
cpu MHz         : 1497.602
cache size      : 8192 KB
physical id     : 2
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 27
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 ss syscall nx pdpe1gb r
dtscp lm constant_tsc arch_perfmon rep_good nopl xtopology tsc_reliable nonstop_tsc cpuid pni pclmulqdq ssse3 fma cx16 pcid sse4_1 sse4_2 x
2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand hypervisor lahf_lm abm 3dnowprefetch cpuid_fault invpcid_single ssbd ibrs i
bpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid avx512f avx512dq rdseed adx smap avx512ifma clflushopt avx512c
d sha_ni avx512bw avx512vl xsaveopt xsavec xgetbv1 xsaves arat avx512vbmi umip pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512_vnni avx5
12_bitalg avx512_vpopcntdq rdpid fsrm md_clear flush_l1d arch_capabilities
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit
bogomips        : 2995.20
clflush size    : 64
cache_alignment : 64
address sizes   : 45 bits physical, 48 bits virtual
power management:
```

1-2) version

```
ubuntu@ubuntu:~/Desktop/myshell$ cat /proc/version
Linux version 5.8.0-50-generic (buildd@lgw01-amd64-030) (gcc (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0, GNU ld (GNU Binutils for Ubuntu) 2.34) #
56~20.04.1-Ubuntu SMP Mon Apr 12 21:46:35 UTC 2021
```

1-3) meminfo



```
ubuntu@ubuntu:~/Desktop/myshell$ cat /proc/meminfo
MemTotal:        2005372 kB
MemFree:          593472 kB
MemAvailable:     819732 kB
Buffers:           16868 kB
Cached:           311704 kB
SwapCached:        14100 kB
Active:           448120 kB
Inactive:         346844 kB
Active(anon):     237516 kB
Inactive(anon):   237756 kB
Active(file):     210604 kB
Inactive(file):   109088 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:        945416 kB
SwapFree:         563204 kB
Dirty:               748 kB
Writeback:             0 kB
AnonPages:        457812 kB
Mapped:           131736 kB
Shmem:              8880 kB
KReclaimable:      75984 kB
Slab:             196492 kB
SReclaimable:      75984 kB
SUnreclaim:       120508 kB
KernelStack:       10336 kB
PageTables:        13548 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1948100 kB
Committed_AS:    3797376 kB
VmallocTotal:   34359738367 kB
VmallocUsed:       36404 kB
VmallocChunk:          0 kB
Percpu:           109056 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:      489344 kB
DirectMap2M:     1607680 kB
DirectMap1G:           0 kB
```

2) 쉘 프로그램 설계 및 구현 내용

2-1) myshell.h

```
#include <stdio.h>

void make_history(char command[], char** history, int i);
void print_history(char** history, int count);
void print_help();
void do_exec(char command[]);
void do_background(char command[]);
```

myshell.h파일에서는 myshell.c에서 사용할 함수들을 선언했습니다. 다음은 함수들에 대한 설명입니다.

1. make_history() ; 입력한 명령어를 history에 저장하는 함수입니다.
2. print_history() ; 지금까지 입력한 명령어의 history를 출력하는 함수입니다.
3. print_help() : myshell의 사용법을 알려주는 함수입니다.
4. do_exec() : 사용자가 입력한 명령어를 인자로 받고 해당 명령어를 실행시키는 함수입니다. 이 때 자식프로세스가 끝날 때까지 부모 프로세스가 기다립니다.
5. do_background() : 사용자가 입력한 명령어를 인자로 받고 해당 명령어를 백그라운드로 실행시키는 함수입니다.

2-2) utility.c - make_history()

```
void make_history(char command[], char** history, int i)
{
        char* newstrptr;
        int l = 0;

        l = strlen(command);
        newstrptr = (char*)malloc(sizeof(char*) * (l + 1));
        strcpy(newstrptr, command);
        history[i] = newstrptr;
}
```

history를 만들기 위한 make_history함수를 구현하였습니다. 인자로 저장할 명령어 command와 저장할 공간인 history, 저장할 인덱스 i를 받았습니다. 처음에 입력받은 값을 미리 만들어둔 history배열에 넣어보았는데 마지막에 입력된 값이 모두 복사되어 출력되어서 구현방법을 수정하였습니다. 입력받은 command의 길이만큼 동적할당을 해주고 해당 배열에 입력받은 문자열을 복사해준 후 history에 넣어줌으로써 복사되는 오류를 해결하였습니다.

2-2) utility.c - print_history()

```
void print_history(char** history, int count)
{
        if(count < 9)
        {
                for(int i=0; i<count; ++i)
                {
                        printf("%d : %s\n", i+1, history[i]);
                }
        }
        else
        {
                for(int i=10; i>0; --i)
                {
                        printf("%d : %s\n", count - i + 1, history[count -i]);
                }
        }
}
```

history를 출력하기 위한 함수를 구현하였습니다. history가 저장되어 있는 배열과 저장되어 있는 정보의 개수를 인자로 받았습니다. 만약 history에 저장되어 있는 명령어의 개수가 10개 이하라면 배열에 있는 만큼 출력이 되도록 하였고, 10개 이상이라면 최근 10개의 명령어를 출력하도록 하였습니다.

2-2) utility.c - print_help()

```
oid print_help()
{
        printf("-b\n");
        printf("This flag forces a ``break'' from option processing, causing any further shell arguments to be treated as non-option argume
ts. The remaining arguments will not be interpreted as shell options. This may be used to pass options to a shell script without confusion
 or possible subterfuge. The shell will not run a set-user-ID script without this option.\n\n");
        printf("-c\n");
        printf("Commands are read from the(single) following argument which must be present.Any remaining arguments are placed in argv\n\n"
);
        printf("-e\n");
        printf("The shell exits if any invoked command terminates abnormally or yields a non-zero exit status.\n\n");
        printf("-f\n");
        printf("The shell will start faster, because it will neither search for nor execute commands from the file .cshrc in the invoker's
ome directory. Note: if the environment variable HOME is not set, fast startup is the default.\n\n");
        printf("-i\n");
        printf("The shell is interactive and prompts for its top-level input, even if it appears not to be a terminal. Shells are interacti
e without this option if their inputs and outputs are terminals\n\n");
        printf("-l\n");
        printf("The shell is a login shell (only applicable if -l is the only flag specified).\n\n");
        printf("-m\n");
        printf("Read .cshrc regardless of its owner and group. This option is dangerous and should only be used by su(1).\n\n");
        printf("-n\n");
        printf("Commands are parsed, but not executed. This aids in syntactic checking of shell scripts. When used interactively, the shell
 can be terminated by pressing control-D (end-of-file character), since exit will not work.\n\n");
        printf("-s\n");
        printf("Command input is taken from the standard input.\n\n");
        printf("-t\n");
        printf("A single line of input is read and executed. A backslash (`\' ) may be used to escape the newline at the end of this line a
d continue onto another line.\n\n");
        printf("-V\n");
```

help명령어를 위한 함수를 구현하였습니다. 정보들을 모두 printf로 출력하였습니다.

2-2) utility.c - do_exec()

```
void do_exec(char command[])
{
        char* arg[7];
        char* str;
        char* save;
        int argv;
        int stat;

        char slice[] = " ";
        int pid = 0;

        argv = 0;

        str = strtok_r(command, slice, &save);
        while (str != NULL)
        {
                arg[argv++] = str;
                str = strtok_r(NULL, slice, &save);
        }

        arg[argv] = (char*)0;


        pid = fork();

        // child process
        if(pid == 0)
        {
                execvp(arg[0], arg);
                exit(0);
        }

        // parent process
        if(pid > 0)
        {
//              wait(NULL);
//              waitpid(-1, &stat, 0);
//              sleep(1);
                pid = waitpid(pid, &stat, 0);

                //pid = wait(&stat);
//              if(WIFEXITED(stat))
//              {
//                      printf("exit not back\n");
//              }
//              if(WIFSIGNALED(stat))
//              {
//                      printf("killed not back\n");
//              }
        }

        // error
        if(pid < 0)
        {
                printf("fork error!\n");
                exit(-1);
        }

}
```

exec함수를 구현하였습니다. 명령어를 인자로 받아왔고, 해당 명령어를 " "로 파싱하였습니다. "&"가 붙어있지 않는 명령어를 실행하는 것이기 때문에 waitpid를 사용하여 부모 프로세스가 자식 프로세스를 끝날 때까지 기다리도록 하였습니다. pid = waitpid(pid, &stat, 0); 상위에 주석으로 되어있는 부분은 제대로 동작하지 않았던 wait을 기억하고 싶어 지우지 않았습니다. 또한 아래 주석처리된 if문을 통해 자식프로세스가 exit이 되는지 kill이 되는지 확인할 수 있었습니다.

2-2) utility.c - do_background()

```
void do_background(char command[])
{
        char* arg[7];
        char* str;
        char* save;
        int argv;
        int stat;

        char slice[] = " ";
        int pid= 0;

        argv = 0;

        str = strtok_r(command, slice, &save);
        while(str != NULL)
        {
                arg[argv++] = str;
                str = strtok_r(NULL, slice, &save);
        }

        arg[argv] = (char*)0;


        pid = fork();

        // child process
        if(pid == 0)
        {
                execvp(arg[0], arg);
                exit(0);
        }

        // parent process
        if(pid>0)
        {
                pid = waitpid(pid, &stat, WNOHANG);

                //pid = wait(&stat);
//              if(WIFEXITED(stat))
//              {
//                      printf("exit\n");
//              }
//              if(WIFSIGNALED(stat))
//              {
//                      printf("killed\n");
//              }
        }

        // error
        if(pid < 0)
        {
                printf("fork error!\n");
                exit(-1);
        }
}
```

백그라운드 지원을 위한 함수를 구현하였습니다. 명령어를 인자로 받았고, 해당 명령어를 " "로 파싱하였습니다. "&"가 붙어있는 명령어는 myshell.c파일의 main함수에서 if문을 사용하여 마지막 문자가 '&'인지 아닌지로 판단할 수 있도록 하였습니다. "&"가 붙어있다면 위 함수를 실행하고, 자식프로세스가 끝날 때까지 기다리지 않아도 되기 때문에 waitpid의 마지막 인자인 option을 WNOHANG을 주어 자식프로세스가 끝날 때 까지 기다리지 않도록 하였습니다. do_exec함수와 마찬가지로 주석 처리된 if문을 통해 자식프로세스가 exit되는지 kill되는지 확인할 수 있었습니다.

2-3) myshell.c

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "head.h"

int main(){
        // input string
        char command[30];
        char* history[100];

        // index of history
        int count = 0;

        // shell start
        while(1)
        {
                // print prompt
                printf("12161095_shell$ ");

                // input command
                fgets(command, sizeof(command), stdin);

                // end of command make \n->\0
                command[strlen(command)-1] = '\0';

                // make_history
                make_history(command, history, count);
                count++;

                // command == quit
                if(!strcmp("quit", command))
                {
                        printf("myshell developed by seokjinkang(12161095)\n");
                        break;
                }

                // command == history
                else if(!strcmp("history", command))
                {
                        print_history(history, count);
                }

                // command == help
                else if(!strcmp("help", command))
                {
                        print_help();
                }

                // background exec
                else if(command[strlen(command) -1] == '&')
                {
                        command[strlen(command) - 1] = '\0';
                        do_background(command);
                }

                // exec
                else
                {
                        do_exec(command);
                }
        }
}
```

shell이 시작되면 학번_shell$ 이 출력되고 fgets를 이용하여 사용자에게 명령어를 입력받도록 하였습니다. fgets는 개행문자까지 입력을 받기 때문에 맨 마지막 문자를 '\0'으로 수정해주었습니다. 입력받은 명령어를 history에 저장하는 make_history함수를 호출하여 저장해주었습니다. 해당 명령어가 내장형 명령어인지 판단하여 내장형이라면 fork()를 사용하지 않고 구현해놓은 help, print_history를 실행하거나 quit이라면 break하여 while문을 탈출하도록 작성했습니다. 내장형 명령어가 아니라면 맨 마지막 문자가 '&'인지 판단하여 자식 프로세스가 종료될 때 까지 기다려야 하는지 아닌지 구분하여 exec함수를 실행하도록 하였습니다.

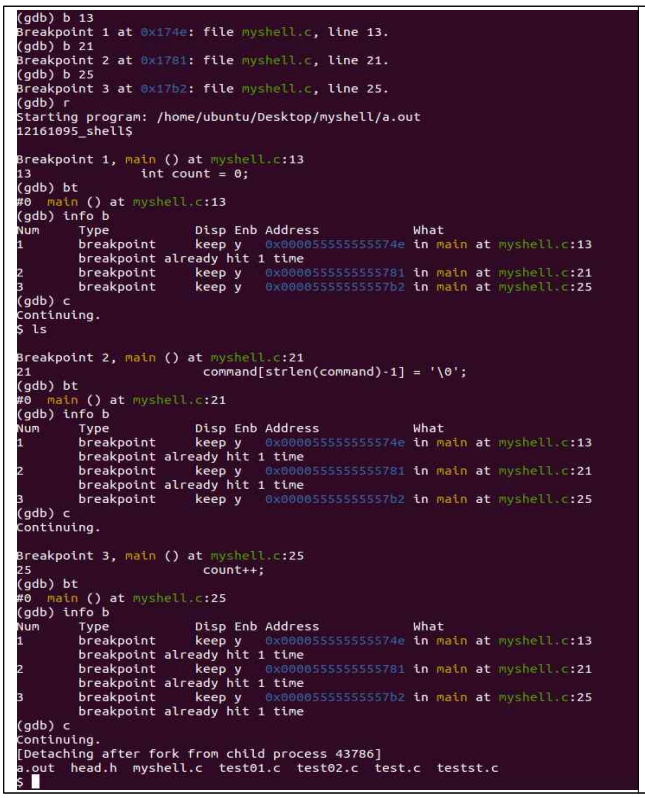
3) 동작과정(GNU gdb 명령어 설정 및 캡춰사진)

3-1) l을 이용하여 전체 소스 출력

```
(gdb) l
1       #include <stdio.h>
2       #include <string.h>
3       #include <stdlib.h>
4       #include "head.h"
5
6       int main(){
7               // input string
8               char command[30];
9               char* history[100];
10
(gdb) list
11              // shell start
12              printf("12161095_shell$\n");
13              int count = 0;
14              while(1)
15              {
16                      printf("$ ");
17                      // input command
18                      fgets(command, sizeof(command), stdin);
19
20                      // end of command make \n->\0
(gdb) list
21                      command[strlen(command)-1] = '\0';
22
23                      // make_history
24                      make_history(command, history, count);
25                      count++;
26
27                      if(!strcmp("quit", command))
28                      {
29                              printf("myshell developed by seokjinkang(12161095)\n");
30                              break;
(gdb) l
31                      }
32                      else if(!strcmp("history", command))
33                      {
34                              print_history(history, count);
35                              //printf("history\n");
36                      }
37                      else if(!strcmp("help", command))
38                      {
39                              print_help();
40                              //printf("help\n");
(gdb) l
41                      }
42                      else if(command[strlen(command) -1] == '&')
43                      {
44                              command[strlen(command) - 1] = '\0';
45                              do_background(command);
46                      }
47                      else
48                      {
49                              do_exec(command);
50                      }
(gdb) l
51              }
52
53      }
```

l을 이용하여 전체 소스코드를 출력하여 라인넘버와 소스를 확인했습니다.

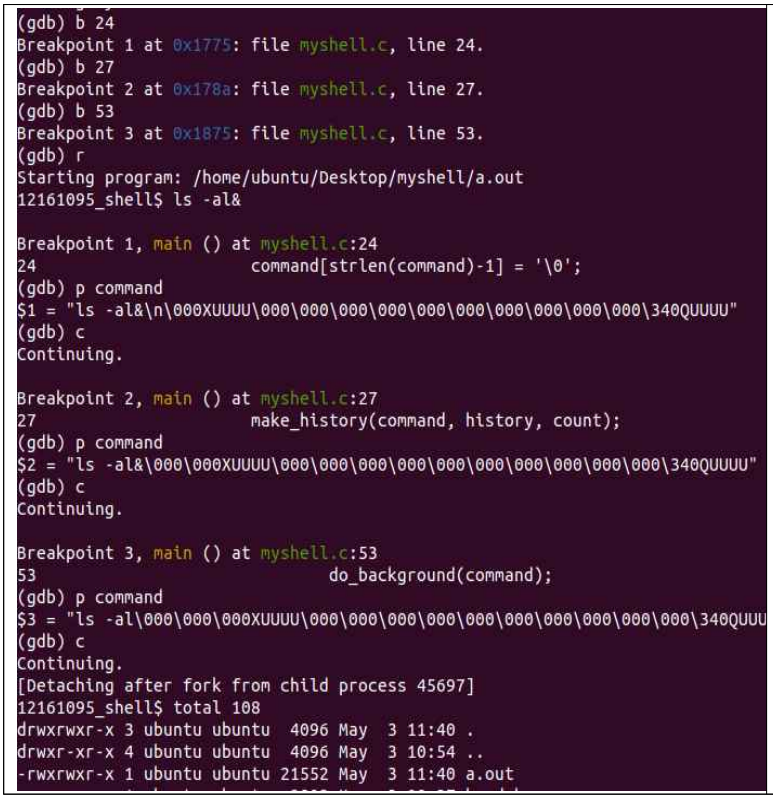
3-2) b, bt, info b, c, r을 이용하여 break point까지 debug



```
(gdb) b 13
Breakpoint 1 at 0x174e: file myshell.c, line 13.
(gdb) b 21
Breakpoint 2 at 0x1781: file myshell.c, line 21.
(gdb) b 25
Breakpoint 3 at 0x17b2: file myshell.c, line 25.
(gdb) r
Starting program: /home/ubuntu/Desktop/myshell/a.out
12161095_shell$

Breakpoint 1, main () at myshell.c:13
13              int count = 0;
(gdb) bt
#0  main () at myshell.c:13
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555574e in main at myshell.c:13
        breakpoint already hit 1 time
2       breakpoint     keep y   0x0000555555555781 in main at myshell.c:21
3       breakpoint     keep y   0x00005555555557b2 in main at myshell.c:25
(gdb) c
Continuing.
$ ls

Breakpoint 2, main () at myshell.c:21
21                      command[strlen(command)-1] = '\0';
(gdb) bt
#0  main () at myshell.c:21
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555574e in main at myshell.c:13
        breakpoint already hit 1 time
2       breakpoint     keep y   0x0000555555555781 in main at myshell.c:21
        breakpoint already hit 1 time
3       breakpoint     keep y   0x00005555555557b2 in main at myshell.c:25
(gdb) c
Continuing.

Breakpoint 3, main () at myshell.c:25
25                      count++;
(gdb) bt
#0  main () at myshell.c:25
(gdb) info b
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000055555555574e in main at myshell.c:13
        breakpoint already hit 1 time
2       breakpoint     keep y   0x0000555555555781 in main at myshell.c:21
        breakpoint already hit 1 time
3       breakpoint     keep y   0x00005555555557b2 in main at myshell.c:25
        breakpoint already hit 1 time
(gdb) c
Continuing.
[Detaching after fork from child process 43786]
a.out  head.h  myshell.c  test01.c  test02.c  test.c  testst.c
$
```

l 을 이용하여 전체 소스코드를 출력해본 것을 바탕으로 해당 라인넘버를 확인할 수 있었습니다. 이를 이용하여 b로 break point를 설정했습니다. r을 이용하여 프로그램을 시작하였고, break point를 만날 때마다 bt를 이용하여 오류가 있는지 체크했습니다. 문제가 없는 것을 확인한 후 info b를 이용하여 현재 설정된 break point의 정보를 확인하였습니다. 현재 위치에서 문제가 없고, 정보를 확인한 후 다음 break point까지 c를 이용하여 진행하였고 만나는 break point마다 문제확인, 정보확인, 계속 진행을 하여 확인했습니다.

3-3) b, p를 이용하여 변수정보 확인



```
(gdb) b 24
Breakpoint 1 at 0x1775: file myshell.c, line 24.
(gdb) b 27
Breakpoint 2 at 0x178a: file myshell.c, line 27.
(gdb) b 53
Breakpoint 3 at 0x1875: file myshell.c, line 53.
(gdb) r
Starting program: /home/ubuntu/Desktop/myshell/a.out
12161095_shell$ ls -al&

Breakpoint 1, main () at myshell.c:24
24                      command[strlen(command)-1] = '\0';
(gdb) p command
$1 = "ls -al&\n\000XUUUU\000\000\000\000\000\000\000\000\000\000\340QUUUU"
(gdb) c
Continuing.

Breakpoint 2, main () at myshell.c:27
27                      make_history(command, history, count);
(gdb) p command
$2 = "ls -al&\000\000XUUUU\000\000\000\000\000\000\000\000\000\000\340QUUUU"
(gdb) c
Continuing.

Breakpoint 3, main () at myshell.c:53
53                              do_background(command);
(gdb) p command
$3 = "ls -al\000\000\000XUUUU\000\000\000\000\000\000\000\000\000\000\340QUUU
(gdb) c
Continuing.
[Detaching after fork from child process 45697]
12161095_shell$ total 108
drwxrwxr-x 3 ubuntu ubuntu  4096 May  3 11:40 .
drwxr-xr-x 4 ubuntu ubuntu  4096 May  3 10:54 ..
-rwxrwxr-x 1 ubuntu ubuntu 21552 May  3 11:40 a.out
```

전체 소스코드를 보면 21번째 라인에서 명령어를 입력받고 24번째 라인에서 '\n'을 '\0'으로 바꿔줍니다. 만약 '&'문자가 있다면 해당 문자를 52번째 라인에서 '\0'으로 바꿔줍니다. 이것을 확인하기 위해 b를 이용하여 24번째, 27번째, 53번째 라인에 break point를 설정해주었습니다. 첫 번째 break point에서 p를 이용하여 command의 정보를 확인해보면 '\n'이 있는 것을 확인할 수 있습니다. 두 번째 break point에서 p를 이용하여 command의 정보를 확인해보면 '\n'이 사라진 것을 확인할 수 있습니다. 마지막 break point에서 p를 이용하여 command의 정보를 확인해보면 '&'이 사라진 것을 확인할 수 있습니다.