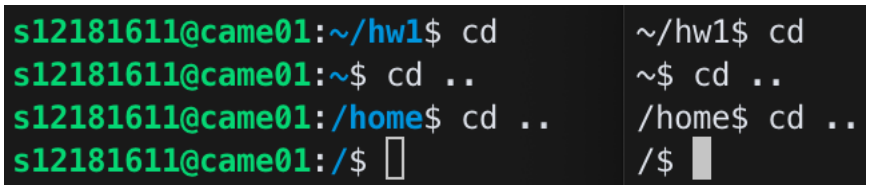
Project 1

리눅스 프로그래밍 004분반
컴퓨터공학과 12181611 박재민

1. prompt에서 현재 디렉토리 표시

shell 실행시 /home/"username"인 주소는 ~로 치환해 출력



main.c의 get_dir 함수

```
4   static struct passwd *pwd;
5
6   char* get_dir(char* dir) {
7       if((pwd = getpwuid(getuid())) == NULL){
8           perror("getpwuid error");
9           return dir;
10      }
11
12      char root_home[255] = "/home/";
13      char* home = strcat(root_home, pwd->pw_name);
14      char* ptr;
15
16      if ((ptr = strstr(dir, home))) {
17          ptr += strlen(home) - 1;
18          dir = ptr;
19          strncpy(dir, "~", 1);
20      }
21      strcat(dir, "$ ");
22      return dir;
23  }
```

4: getpwuid가 사용할 구조체를 전역변수로 선언

12, 13: "/home/"인 내용을 가지는 char 배열을 선언하고 username과 합쳐 "/home/username""을 만듦

7~9: getpwuid 함수를 에러처리 코드와 함께 작성

16~19: 입력으로 들어온 dir에 home 디렉토리의 주소가 존재하는지 strstr함수를 사용해 찾고 있다면 그 시작위치를 ptr에 저장, 없다면 ptr에 0이 저장됨
시작위치가 존재한다면 if문으로 들어가 home 디렉토리의 주소길이 -1 만큼 ptr을 이동시키고 ptr을 새로운 dir로 변경 그 후 dir의 맨 처음 문자를 ~로 변경

21, 22: dir의 마지막에 "& "를 붙인 후 리턴

main.c의 main함수

```
3    static char cwd[255];
```

```
24   int main(int argc, char* argv[]) {
25       char* prompt = get_dir(getcwd(cwd, 255));
26       while (userin(prompt) != EOF) {
27           procline();
28           prompt = get_dir(getcwd(cwd, 255));
29       }
30       return 0;
31   }
```

3: 주소를 저장할 char 배열 cwd를 선언, $를 strcat하기 위해 char배열로 선언함

25, 28: getcwd함수를 이용해 현재 working directory를 얻어온 후 get_dir함수로 디렉토리 전처리후 사용

2. cd기능 구현

cd만 입력하거나 주소를 ~만 주었을 때 때 홈 디렉토리로 이동

~과 같이 사용해 홈 디렉토리에 있는 주소로 이동 가능

```
s12181611@came01:~/hw1$ cd                  ~/hw1$ cd
s12181611@came01:~$ cd submission/          ~$ cd submission/
s12181611@came01:~/submission$ cd ~/hw1     ~/submission$ cd ~/hw1
s12181611@came01:~/hw1$ cd ~                ~/hw1$ cd ~
s12181611@came01:~$                         ~$
```

smallsh.c의 runcommand 함수

```
int runcommand(char** cline, int where, int narg, int redirection);
```

```
127     if (strstr(*cline, "cd")) {
128         if(narg > 2){
129             printf("cd: too many arguments\n");
130             return -1;
131         }
132         run_cd(cline, narg);
133         return 0;
134     }
```

argument의 수를 확인하기 위해 narg도 입력받도록 함수의 파라미터를 변경

127: strstr을 이용해 첫번째 cmd가 cd인지 확인하고 if문 진입

128~130: argument가 3개 이상이라면 에러메세지 출력 후 함수 종료

132: argument가 2개 이하라면 run_cd 실행 후 함수 종료

smallsh.c의 run_cd 함수

```
96   void run_cd(char** cline, int narg) {
97       if((pwd = getpwuid(getuid())) == NULL){
98           perror("getpwuid error");
99           return ;
100      }
101
102      if (narg > 1) {
103          char home[255] = "/home/";
104          char* dir = *(cline + 1);
105          strcat(home, pwd->pw_name);
106
107          if (strstr(dir, "~")){
108              strcat(home, dir + 1);
109              dir = home;
110          }
111          if(chdir(dir) == -1) {
112              if(errno == EACCES){
113                  printf("%s: Permission denied\n", dir);
114              }
115          }
116      } else if(chdir(pwd->pw_dir) == -1) {
117          printf("chdir error\n");
118      }
119  }
```

get_dir함수 처럼 getpwuid를 사용해 유저 정보를 pwd에 저장

102~115: 추가 argument가 있는 경우는 주소 전처리를 위해 if문으로 이동

116: 추가 argument가 없는 경우는 유저의 홈 디렉토리로 이동

103, 105: "/home/'username'" 만들기

104: 두번째 cmd를 dir 변수에 저장

107~109: dir에 ~가 존재한다면 home디렉토리에 ~이후 부분을 연결하고 그 주소를 dir로 변경

111~113: 전처리된 dir로 이동, 이동에 실패한 경우 이동하지 않고 errno가 EACCESS인지 확인 후 권한이 없다고 출력

3. exit명령어 구현

shell상에서 exit를 입력하면 shell 종료

```
~/hw1$ exit
s12181611@came01:~/hw1$
```

smallsh.c의 runcommand 함수

```
135         if(strstr(*cline, "exit")){
136             exit(0);
137         }
```

cmd에 exit이 존재하는지 확인하고 존재한다면 exit함수 호출

4. redirection(>) 처리

```
~/hw1$ date
Sun 13 Nov 2022 08:30:23 AM UTC
~/hw1$ date > a.txt
~/hw1$ cat a.txt
Sun 13 Nov 2022 08:30:29 AM UTC
~/hw1$
```

```
~/hw1$ ls > a.txt;date
Sun 13 Nov 2022 08:31:40 AM UTC
~/hw1$ cat a.txt
a.txt
main.c
main.o
Makefile
shell
smallsh.c
smallsh.h
smallsh.o
test.c
~/hw1$
```

```
~/hw1$ date > a.txt;ls -a; ls -l;cat a.txt
.  ..  a.txt  main.c  main.o  Makefile  shell  smallsh.c  smallsh.h  smallsh.o  test.c
total 56
-rw-r--r-- 1 s12181611 s12181611    32 Nov 13 08:33 a.txt
-rw-r--r-- 1 s12181611 s12181611   484 Nov 12 17:08 main.c
-rw-rw-r-- 1 s12181611 s12181611  2256 Nov 12 17:36 main.o
-rw-rw-r-- 1 s12181611 s12181611   226 Nov 12 17:36 Makefile
-rwxrwxr-x 1 s12181611 s12181611 18024 Nov 12 17:36 shell
-rw-r--r-- 1 s12181611 s12181611  3594 Nov 12 17:20 smallsh.c
-rw-r--r-- 1 s12181611 s12181611   594 Nov 12 17:18 smallsh.h
-rw-rw-r-- 1 s12181611 s12181611  6440 Nov 12 17:36 smallsh.o
-rw-rw-r-- 1 s12181611 s12181611   361 Nov 11 13:58 test.c
Sun 13 Nov 2022 08:33:22 AM UTC
~/hw1$
```

```
14    #define REDIRECTION 5
```

```
22    struct gettok_pair{
23        int type;
24        int redirection;
25    };
```

```
29    struct gettok_pair gettok(char** outptr);
```

```
31    int runcommand(char** cline, int where, int narg, int redirection);
```

redirection상태를 표현하기 위해 REDIRECTION을 새로 정의

gettok에서 type과 redirection 유무를 전달하기 위해 새로운 구조체를 만들고 gettok함수가 구조체를 반환하도록 변경

runcommand함수가 redirection유무를 파라미터로 받게 변경

smallsh.c의 gettok함수

```
36  struct gettok_pair gettok(char **outptr) {
37      *outptr = tok;
38      while (*ptr == ' ' || *ptr == '\t') ptr++;
39      *tok++ = *ptr;
40      switch (*ptr++) {
41          case '\n':
42              pair.type = EOL;
43              break;
44          case '&':
45              pair.type = AMPERSAND;
46              break;
47          case ';':
48              pair.type = SEMICOLON;
49              break;
50          case '>':
51              pair.redirection = REDIRECTION;
52          default:
53              pair.type = ARG;
54              while (inarg(*ptr)) *tok++ = *ptr++;
55      }
56      *tok++ = '\0';
57      return pair;
58  }
```

전역변수로 struct gettok_pair pair를 선언하고 그 구조체를 수정하도록 함수를 변경
50~54: 문자 중 >가 존재하면 redirection 상태를 변경하고 cmd에 포함시키기 위해 break 사용 안함

smallsh.c의 runcommand 함수

```
122         char *temp[MAXARG + 1];
```

```
126       int i;
```

```
143               if(redirection) {
144                   for(i = 0; !strstr(*(cline + i), ">"); i++) temp[i] = *(cline + i);
145                   fd = open(*(cline + i + 1), O_WRONLY|O_CREAT|O_TRUNC, 0644);
146                   fcntl(fd, F_SETFD, 1);
147                   dup2(fd, 1);
148                   cline = temp;
149               }
```

143: cmd에 >가 존재하면 if문으로 진입

144: for문과 strstr을 사용해 cline중 >가 있는곳까지 i를 증가시키고 temp에 명령어 저장

145: > 다음 cmd는 파일 이름이므로 open을 사용해 열고 열때는 없으면 만들고 있으면 비움

146: exec를 하면 파일을 닫을 수 없으므로 close_on_exec flag를 켬

147: dup2를 이용해 STDOUT의 file descriptor를 fd로 복제

148: > 전까지 저장된 명령어를 cline으로 변경

smallsh.c의 procline함수

```
74              gettok(&arg[narg]);
75              switch (toktype = pair.type) {
```

```
86                      runcommand(arg, type, narg, pair.redirection);
87                      pair.redirection = 0;
```

74: gettok으로 전역변수 pair 수정

75: pair의 type을 toktype으로 사용

86, 87: runcommand의 redirection을 pair.redirection으로 사용하고 사용 후 0으로 변경

5. 기본 구현 코드에 있는 '&'의 문제점 분석

```
s12181611@came01:~/ex8$ ./ex_test &
[1] 1548800
s12181611@came01:~/ex8$ ps -o pid,ppid,pgid,sid,command
    PID    PPID    PGID     SID COMMAND
1548578 1548577 1548578 1548578 -bash
1548800 1548578 1548800 1548578 ./ex_test
1548801 1548800 1548800 1548578 ./ex_test
1548802 1548800 1548800 1548578 ./ex_test
1548803 1548801 1548800 1548578 ./ex_test
1548826 1548578 1548826 1548578 ps -o pid,ppid,pgid,sid,command
s12181611@came01:~/ex8$
```

```
~/ex8$ ./ex_test &
[Process id] 1548870
~/ex8$ ps -o pid,ppid,pgid,sid,command
    PID    PPID    PGID     SID COMMAND
1545064 1545063 1545064 1545064 -bash
1547175 1545064 1547175 1545064 ./shell
1548870 1547175 1547175 1545064 ./ex_test
1548871 1548870 1547175 1545064 ./ex_test
1548872 1548870 1547175 1545064 ./ex_test
1548873 1548871 1547175 1545064 ./ex_test
1548912 1547175 1547175 1545064 ps -o pid,ppid,pgid,sid,command
~/ex8$
```

기존 shell은 shell의 그룹이 따로 존재하지만 기본 구현 shell은 shell이 실행한 파일과 같은 그룹으로 존재한다

따라서 8주차 실습의 ex_kill로 ex_test의 그룹 아이디 전체에 kill signal을 보내게 되면 shell도 같이 종료되게 된다.

이러한 문제를 방지하려면 fork시 자식 프로세스는 자식프로세스를 리더로 하는 그룹을 만들도록 한다.

그리고 자식이 좀비가 되는걸 막기 위해 main에서 한 커맨드가 끝나면 자식 프로세스를 wait한다 이때 block되는걸 방지하기 위해 WNOHANG option을 사용한다.

```
138         switch (pid = fork()) {
139             case -1:
140                 perror("smallsh");
141                 return -1;
142             case 0:
143                 setpgid(getpid(), 0);
```

```
32              waitpid(-1, &status, 0|WNOHANG);
```

```
~/ex8$ ./ex_test &
[Process id] 1603727
~/ex8$ ps -o pid,ppid,pgid,sid,command
    PID    PPID    PGID     SID COMMAND
1592398 1592397 1592398 1592398 -bash
1602769 1592398 1602769 1592398 ./shell
1603727 1602769 1603727 1592398 ./ex_test
1603728 1603727 1603727 1592398 ./ex_test
1603729 1603727 1603727 1592398 ./ex_test
1603730 1603728 1603727 1592398 ./ex_test
1603827 1602769 1603827 1592398 ps -o pid,ppid,pgid,sid,command
~/ex8$ ./ex_kill 9 -g 1603727
~/ex8$ ps -o pid,ppid,pgid,sid,command
    PID    PPID    PGID     SID COMMAND
1592398 1592397 1592398 1592398 -bash
1602769 1592398 1602769 1592398 ./shell
1604129 1602769 1604129 1592398 ps -o pid,ppid,pgid,sid,command
~/ex8$
```