Project 2

리눅스 프로그래밍 004분반
컴퓨터공학과 12181611 박재민

project1에서 가독성을 위해 코드를 일부 수정했지만 기능은 정상 작동

1. &의 문제점 해결

project1에서 이미 해결했기 때문에 그때 코드를 조금 수정해 사용
main에서 한 커맨드가 끝나고 while문을 다시 시작할때 자식 프로세스를 wait한다 이때 block되는걸 방지하기 위해 WNOHANG option을 사용한다.

main.c

```
30  int main(int argc, char* argv[]) {
31      int status;
32      char* prompt = get_dir(getcwd(cwd, 255));
33      act.sa_handler = int_handler;
34      sigsetjmp(position, 1);
35      while (sigaction(SIGINT, &act, NULL), userin(prompt) != EOF) {
36          waitpid(-1, &status, 0 | WNOHANG);
37          procline();
38          prompt = get_dir(getcwd(cwd, 255));
39      }
40      return 0;
41  }
```

```
~/submission/project2$ ./sleep &
[Process id] 1307893
~/submission/project2$ ps
    PID TTY          TIME CMD
1258242 pts/37   00:00:00 bash
1307523 pts/37   00:00:00 shell
1308650 pts/37   00:00:00 ps
~/submission/project2$
```

10초간 잠들었다 종료하는 sleep 프로그램을 백그라운드로 실행한 다음, 10초후에 ps로 확인해 봤을 때 sleep프로세스가 없는것을 볼 수 있다.

2. SIGINT 처리

2-1. SIGINT가 들어왔을 때 smallsh가 닫히지 않도록 처리

```
12    void int_handler(int signo) {
13        putchar('\n');
14        siglongjmp(position, 1);
15    }
```

```
 6  int main(int argc, char* argv[]) {
 7      int status;
 8      char* prompt = get_dir(getcwd(cwd, 255));
 9      act.sa_handler = int_handler;
10      sigsetjmp(position, 1);
11      while (sigaction(SIGINT, &act, NULL)  userin(prompt) != EOF) {
12          waitpid(-1, &status, 0 | WNOHANG);
13          procline();
14          prompt = get_dir(getcwd(cwd, 255));
15      }
16      return 0;
17  }
```

SIGINT가 들어왔을 때 줄바꿈만 하고 다시 prompt를 띄우는 줄로 돌아가는 int_handler를 만들고 sigaction을 통해 등록한다

2-2. foreground 프로세스만 종료되도록 처리

smallsh.c의 runcommand 함수

```
201     switch (pid = fork()) {
202         case -1:
203             perror("smallsh");
204             return -1;
205         case 0:
206             if (where == BACKGROUND) setpgid(getpid(), 0);
207             else {
208                 act.sa_handler = SIG_DFL;
209                 sigaction(SIGINT, &act, NULL);
210             }
211             if (types.type2 == REDIRECTION)
212                 cline = run_redirection(cline, narg);
213             if (types.type2 == PIPE) {
214                 run_pipe(cline, narg);
215                 perror("pipe run");
216                 exit(0);
217             }
218             execvp(*cline, cline);
219             perror(*cline);
220             exit(1);
221     }
222     if (where == BACKGROUND) {
223         printf("[Process id] %d\n", pid);
224         return 0;
225     }
```

프로세스가 background인 경우 프로세스가 리더인 새로운 그룹을 생성하게 해 SIGINT를 받지 않게 했고 foreground인 경우 핸들러에 시그널 기본동작을 하게 만드는 SIG_DFL을 설정하고 등록해 SIGINT를 받았을 때 종료되게 한다.

```
~/submission/project2$ ./sleep &
[Process id] 1318120
~/submission/project2$ ./sleep2
^C
~/submission/project2$ ps
    PID TTY          TIME CMD
1258242 pts/37   00:00:00 bash
1315301 pts/37   00:00:00 shell
1318120 pts/37   00:00:00 sleep
1318152 pts/37   00:00:00 ps
~/submission/project2$ ^C
~/submission/project2$
```

10초동안 sleep한 후 종료되는 sleep, sleep2을 만들고 sleep은 background로, sleep2는 foreground로 실행한다
이후 ctrl + C를 눌러 SIGINT를 보내 foreground 프로세스를 종료한 후 ps를 확인해 보면 sleep 프로세스는 잘 작동하는 것을 볼 수 있다
더불어 smallsh도 ctrl + C를 했을 때 줄바꿈 후 정상작동하는걸 볼 수 있다.

3. |(pipe) 처리

struct tok_types

```
struct tok_types {
    int type1;
    int type2;
};
```

gettok함수에서 redirection, pipe 유무를 체크하기 위해 사용

type1은 기존 타입들(ARG, EOF, …) type2는 redirection이나 pipe용으로 사용

smallsh.c의 gettok 함수

```
77          case '|':
78              types.type2 = PIPE;
79          default:
80              types.type1 = ARG;
81              while (inarg(*ptr)) *tok++ = *ptr++;
```

pipe 처리를 위해 명령어 토큰을 나눌 때 명령어 중 |가 존재하는지 확인하고 존재한다면 type2를 PIPE로 변경

break가 없기 때문에 default로 내려가게 되고 |도 토큰 배열에 들어가게 된다.

smallsh.c의 runcommand 함수

```
212             if (types.type2 == PIPE) {
213                 run_pipe(cline, narg);
214                 perror("pipe run");
215                 exit(0);
216             }
```

types.type2가 PIPE라면 run_pipe 함수로 이동

run_pipe 함수는 return이 없기 때문에 return되는 경우 에러 출력

smallsh.c의 run_pipe 함수

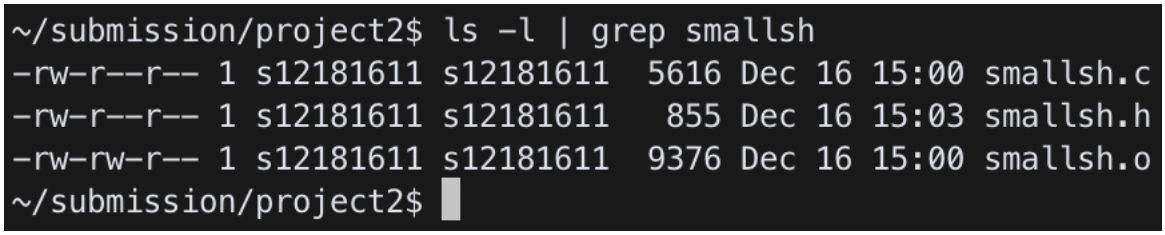
```
158   void run_pipe(char** cline, int narg) {
159       char *cmd1[MAXBUF] = {NULL}, *cmd2[MAXBUF] = {NULL};
160       int i = 0, j = 0;
161       int p[2];
162
163       for (i; strcmp(cline[i], "|"); i++) cmd1[i] = cline[i];
164       for (i = i + 1; i < narg; i++) cmd2[j++] = cline[i];
165       if ((pipe(p) == -1)) perror("pipe");
166       else {
167           switch (fork()) {
168               case -1:
169                   perror("pipe fork");
170               case 0:
171                   dup2(p[0], 0);
172                   close(p[0]);
173                   close(p[1]);
174                   execvp(*cmd2, cmd2);
175                   perror(*cmd2);
176                   exit(1);
177               default:
178                   dup2(p[1], 1);
179                   close(p[0]);
180                   close(p[1]);
181                   execvp(*cmd1, cmd1);
182                   perror(*cmd1);
183                   exit(1);
184           }
185       }
186   }
```

for문을 돌며 | 직전까지 명령어는 cmd1에 | 이후 명령어는 cmd2에 저장
그리고 pipe를 생성후 fork
부모 프로세스는 pipe out을 stdout으로 사용하고 자식 프로세스는 pipe in을 stdin으로 사용
부모와 자식 프로세스 모두 pipe를 닫은 후 부모 프로세스는 cmd1, 자식 프로세스는 cmd2를 실행

smallsh.c의 procline

```
112                 if (narg != 0) {
113                     arg[narg] = NULL;
114                     runcommand(arg, where, narg, types);
115                     types.type2 = 0;
116                 }
```

다음 명령어가 정상적으로 처리되도록 runcommand가 실행된 후 types.type2를 0으로 변경



ls -l의 출력값 중에서 smallsh가 포함된 라인만 출력된 것을 확인 가능