

과제 4: "mysh 작성"

시스템 프로그래밍 7분반
소프트웨어학과 32170578 김산

1. 구현 설명

명령어를 입력받아 처리하는 간단한 셸 프로그램입니다.

- Default command set

ls, cd, help, exit로 구성된 셸에서 미리 실행할 수 있는 명령어를 구현하였습니다.

기본 명령어	설명
ls	dirent.h 헤더파일의 구조체에 있는 readdir함수를 이용하여 현재 디렉토리의 파일 정보를 읽어와 출력하도록 작성하였습니다.
cd	chdir 함수를 이용하여 두 번째 인자에 입력된 주소로 현재 디렉토리를 변경합니다
help	셸의 기본 명령어를 출력하도록 하였습니다.
exit	셸을 종료합니다.

- Shell logic

`exec_handler()`는 입력받은 command line을 tokenizer를 통해 토큰화하여 저장합니다.

`parse_default()`를 통해 명령어가 기본명령어인지 확인하고 기본 명령어 셋에 포함되어 있으면

`exec_default()`을 통해 명령어에 해당하는 함수를 실행합니다

만약 기본명령어가 아니면 fork를 통해 자식프로세스를 생성합니다

`parse_Redirect()`를 통해 redirect를 지시하는 인자(>)가 포함되어 있는지 확인하고 포함되어 있으면

`exec_redirect()`를 실행하여 해당 인자 다음의 인자로 출력을 재지정하여 수행합니다.

`parse_Background()`는 background를 지시하는 인자(&)가 포함되어 있는지 확인하고 포함되어 있으면 부모 프로세스가 wait()를 실행하지 않게 하여 자식 프로세스의 백그라운드 프로세싱을 수행합니다.

2. 수행결과 스냅샷

1. ls, ps, gcc, a.out 실행

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  3: cppdbg: simple_shell v + [] 🗑 ^ ×

/mnt/c/workspace/32170578 $ ls
hello.c loop.c simple_shell simple_shell.c
/mnt/c/workspace/32170578 $ ps
  PID TTY          TIME CMD
  348 ?            00:00:00 simple_shell
  353 ?            00:00:00 ps
/mnt/c/workspace/32170578 $ gcc hello.c
/mnt/c/workspace/32170578 $ ls
a.out  hello.c loop.c simple_shell simple_shell.c
/mnt/c/workspace/32170578 $ ./a.out
hello world!
/mnt/c/workspace/32170578 $
```

- gcc hello.c를 통해 a.out 파일이 생성된 것을 확인할 수 있습니다.

2. 백그라운드 실행

다음과 같은 내용의 loop 프로그램을 백그라운드에서 실행시킵니다.

```
main(){
    while(1){}
}
```

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  3: cppdbg: simple_shell v + [] 🗑 ^ ×

/mnt/c/workspace/32170578 $ ps
  PID TTY          TIME CMD
  282 ?            00:00:00 simple_shell
  287 ?            00:00:00 ps
/mnt/c/workspace/32170578 $ ./loop &
/mnt/c/workspace/32170578 $ ps
  PID TTY          TIME CMD
  282 ?            00:00:00 simple_shell
  288 ?            00:00:01 loop
  289 ?            00:00:00 ps
/mnt/c/workspace/32170578 $
```

./loop &를 입력하면 쉘은 계속 화면에 있지만 loop 프로세스는 실행 중인 것을 확인할 수 있습니다.

3. 리다이렉션 구현

```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  3: cppdbg: simple_shell
/mnt/c/workspace/32170578 $ ls
a.out  hello.c  loop  loop.c  simple_shell  simple_shell.c
/mnt/c/workspace/32170578 $ ./a.out > text.txt
redirect
/mnt/c/workspace/32170578 $ cat text.txt
hello world!
/mnt/c/workspace/32170578 $ ls
a.out  hello.c  loop  loop.c  simple_shell  simple_shell.c  text.txt
/mnt/c/workspace/32170578 $
```

앞서 gcc를 통해 컴파일한 a.out 파일의 출력을 text.txt 로 재지정하여 text.txt 파일을 생성했습니다. open() 시스템 호출의 옵션에 O_CREAT 이 있기 때문에 text.txt파일이 없더라도 생성됨을 확인할 수 있습니다.

4. gdb를 이용한 디버깅

main함수에서는 while문에 의해 같은 분장이 반복되기 때문에 gdb에서 r을 실행하면 다시 gdb로 돌아갈수 없었습니다. 따라서 while문 내부에 breakpoint를 설정하여 디버깅을 했습니다.

```
san@DESKTOP-C6CKGU9: /mnt/c/workspace/32170578
hello world!
/mnt/c/workspace/32170578 $ exit
[Inferior 1 (process 410) exited with code 01]
(gdb) b 39
Breakpoint 1 at 0x8000e00: file /mnt/c/workspace/32170578/simple_shell.c, line 39.
(gdb) n
The program is not being run.
(gdb) r
Starting program: /mnt/c/workspace/32170578/simple_shell
/mnt/c/workspace/32170578 $ ps
  PID TTY          TIME CMD
  358 tty2      00:00:00 bash
  408 tty2      00:00:00 gdb
  415 tty2      00:00:00 simple_shell
  416 tty2      00:00:00 ps
Breakpoint 1, main () at /mnt/c/workspace/32170578/simple_shell.c:39
39     while(1){
(gdb) n
40         fflush(NULL);
(gdb) n
41         printf("%s $ ", get_current_dir_name());
(gdb) n
42         fgets(cmd_line, MAX_CMDLEN, stdin);
(gdb) n
/mnt/c/workspace/32170578 $ ps
  43     cmd_line[strlen(cmd_line) -1] = NULL;
(gdb) p 'simple_shell.c::cmd_line
$1 = "ps\n", 'W000' <repeats 96 times>
(gdb)
```

n을 입력하여 한줄 한줄 실행하면서 실행상황을 확인했고 실수하기 쉬웠던 부분인 명령어를 입력하는 부분을 p 'simple_sheel.c::cmd_line 을 통해 전역변수의 값을 확인하고 'wn'을 제거하는 코드를 작성하였습니다.

```
san@DESKTOP-C6CKGU9: /mnt/c/workspace/32170578
(gdb) b 44
Breakpoint 1 at 0xdeb: file /mnt/c/workspace/32170578/simple_shell.c, line 44.
(gdb) r
Starting program: /mnt/c/workspace/32170578/simple_shell
/mnt/c/workspace/32170578 $ ./a.out > text.txt

Breakpoint 1, main () at /mnt/c/workspace/32170578/simple_shell.c:44
44     exec_handler();
(gdb) n
redirect
45     *tokens = NULL;
(gdb) p tokens
$1 = {0x82020e0 <cmd_line> "./a.out", 0x82020e8 <cmd_line+8> ">", 0x82020ea <cmd_line+10> "text.txt", 0x0, 0x0, 0x0, 0x0, 0x0, 0x0}
(gdb)
```

p tokens를 통해 명령어를 입력후 tokens내부 변수의 값을 확인하여 tokenizer가 정상적으로 작동함을 확인할 수 있었습니다.

5. Discussion

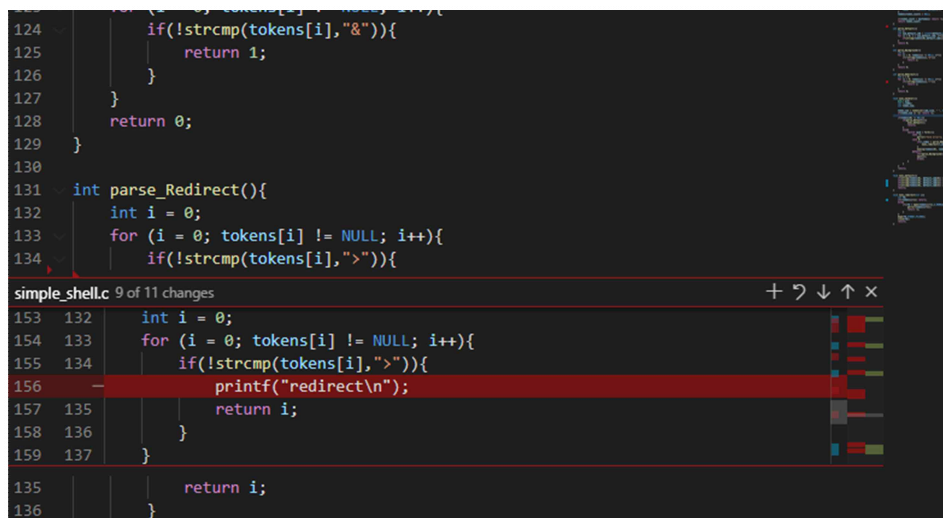
이번 과제는 예상했던 것과는 너무 다르게 여태까지 해왔던 모든 과제 중에 가장 시간이 오래 걸린 듯합니다. 그럼에도 불구하고 코드도 지저분하고 잔 버그들을 해결하지 못했습니다. 다만 깨달은 것들이 몇 가지 있습니다.

입출력이 있을 때는 gdb를 무조건 사용해야겠다.

처음 과제를 시작하면서 tokenizer 함수를 작성하는 부분에서 가장 애를 먹었습니다. tokenizer 함수를 작성하고 이후 다른 함수를 작성하면서 오류가 발생 했습니다. gdb를 사용하지 않았으니 token배열에 들어가는 부분들을 예상만 하고 있었기에 tokenizer가 출력을 잘못했구나! 라고 생각하고 tokenizer를 수정하고 있었습니다. 그렇게 몇시간을 고민하다가 gdb를 사용하였고 결국 tokenizer가 정상적으로 작동하고 있었고 이후 작성한 코드에서 잔 실수가 있었음을 확인할 수 있었습니다.

git을 사용하자

저는 vscode에 github를 연결하여 사용하고 있습니다. 여태까지는 여러대의 컴퓨터에 파일을 동기화하는 용도로만 사용하고 있었습니다. 과제를 하면서 워낙 다양한 함수 간의 교류가 있기에 살짝 다른 함수를 수정하고 돌아와 보니 오류가 발생하는 상황이 자주 연출되었습니다. 수정하기 이전의 내용을 확인하기 위해 git이 이전에 커밋 했던 코드와 비교해주는 것을 기억하여 활용하기 시작했습니다.



```
124         if(!strcmp(tokens[i], "&")){
125             return 1;
126         }
127     }
128     return 0;
129 }
130
131 int parse_Redirect(){
132     int i = 0;
133     for (i = 0; tokens[i] != NULL; i++){
134         if(!strcmp(tokens[i], ">")){
135
136         }
137     }
138
139     return i;
140 }
141
142 int main(){
143     char *tokens[100];
144     int i = 0;
145     while(1){
146         char *line = readline("simple_shell> ");
147         if(!line) continue;
148         if(!strcmp(line, "exit")){
149             exit(0);
150         }
151         tokens[i] = strtok(line, " ");
152         i++;
153     }
154     if(parse_Command() == 1){
155         if(parse_Redirect() != 0){
156             printf("redirect\\n");
157         }
158         execute_Command();
159     }
160 }
```

simple_shell.c 9 of 11 changes

```
153 132     int i = 0;
154 133     for (i = 0; tokens[i] != NULL; i++){
155 134         if(!strcmp(tokens[i], ">")){
156 -             printf("redirect\\n");
157 135         }
158 136     }
159 137 }
135
136     return i;
136 }
```

덕분에 이전엔 되었지만 수정하여 안 되는 부분이 있으면 다시 쉽게 돌아가 고민할 수 있었고 항상 git을 활용해야겠다는 생각을 가지게 되었습니다.