Operating Systems

Project #1

(Draft report)

미완성 보고서

2018045214 | 최 준 희

2021.03.28

Simple Shell algorithm

함수

```
int shell_inputScan(char prompt[], char *cmd_list[]);
void shell_execute(int num, char *cmd_list[]);
int *shell_flag(int num, char *cmd_list[]);
void shell_input_redirect(char *cmd1[], char *cmd2[]);
void shell_output_redirect(char *cmd1[], char *cmd2[]);
void shell_pipe(char *cmd1[], char *cmd2[]);
```

프로그램 흐름

1. 프로그램이 실행되면, 바로 자식프로세스를 생성하여 자식프로세스에서 입력을 받는다.

2. 입력 받은 내용을 inputScan 을 통해 토큰화

3. 토큰화된 단어 포인터 배열을 가지고 execute 를 실행한다.

4. execute 에서는 shell_flag 함수를 사용하여 리다이렉션이나 파이프가 있는지 확인한다.

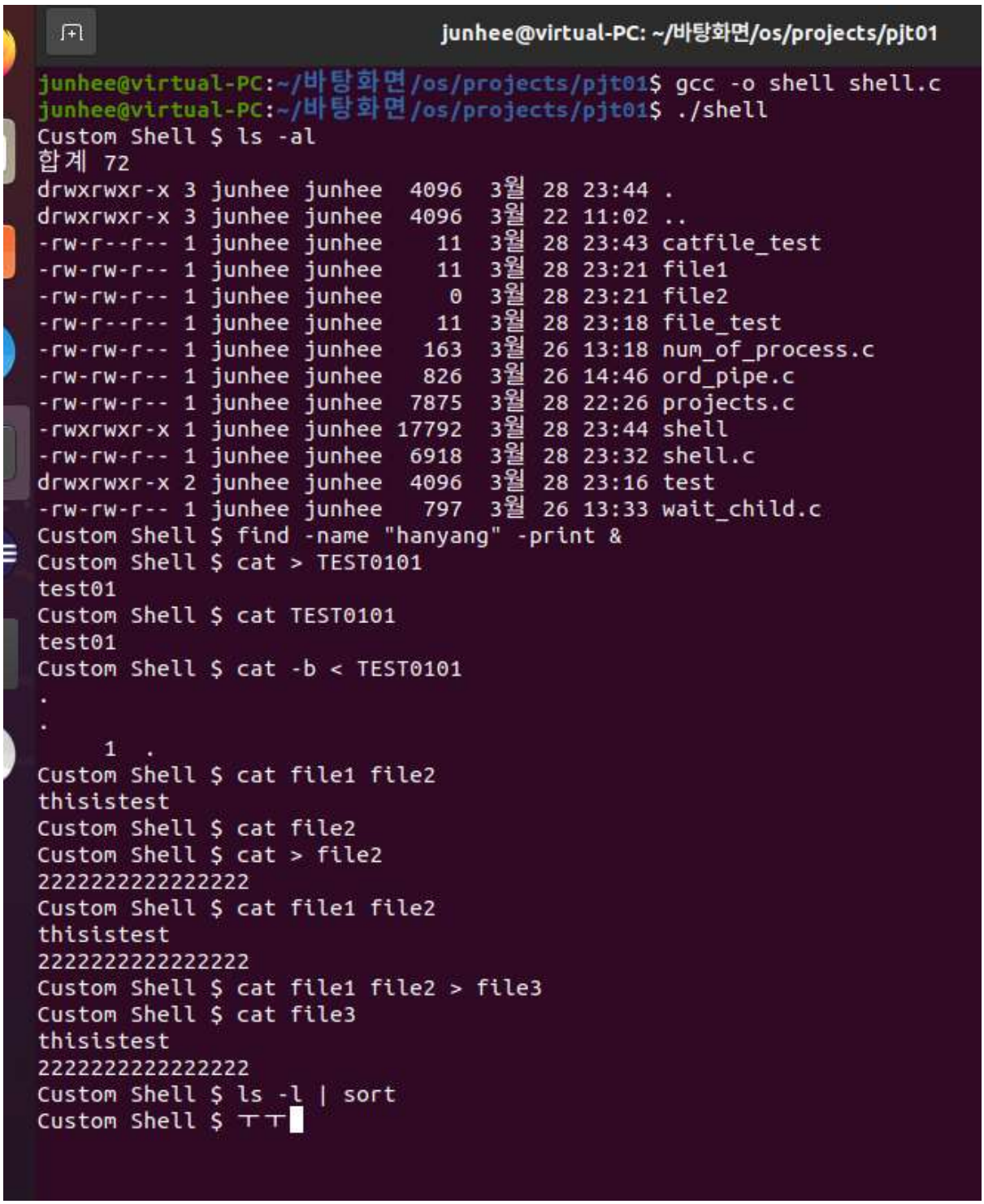
5-1. 리다이렉션이 있는 경우에는 input_redirect, output_redirect 함수를 호출한다.

5-2. input_redirect 에서는 파일 디스크립터를 이용해 새로운 파일을 open 하고, dup2 를 사용하여 STDOUT_FILENO (표준 출력)을 파일 디스크립터가 가르키도록 하고, execvp 를 통해 명령어를 실행한다.

5-3. output_redirect 에서도 파일 디스크립터를 통해 open 을 사용하며, 내용을 읽어올 파일을 read 하여 저장, 2 번에서 사용했던 inputScan 을 통해 내용을 토큰화 하고, 그 내용을 가지고 execvp 를 통해 실행할 수 있도록 한다.

6. 파이프가 있는 경우엔 두번의 fork 를 통해 표준 출력을 받아서 담아두었다가, 표준 출력을 표준 입력에 집어넣어서 연산이 이루어 지도록 기획했다.

실행 화면



- Cat -b < test0101 과 같이 리다이렉션이 아직 버그가 있습니다...
- 파이프가 작동하지 않습니다.
- 4 월 1 일까지 디버깅 하겠습니다.

소스 코드

shell.c untitled

```c
/*
   이 프로젝트는 유저의 입력을 받아 별도의 자식 프로세스에서
   각 명령을 실행하는 쉘 인터페이스 역할을 하는 프로그램을 설계한다.

   이 프로젝트를 위해 UNIX System call (pork, exec, wait, dup2, pipe)를 사용한다.
   입력 받는 명령어의 형식은 다음과 같다.

   $ 명령어 옵션
   $ 명령어 옵션 &
   $ 명령어 옵션 > 파일명
   $ 명령어 옵션 > 파일명 &
   $ 명령어 옵션 < 파일명
   $ 명령어 옵션 < 파일명
   $ 명령어 옵션 < 파일명 &
   $ 명령어 옵션 | 명령어 옵션
   $ 명령어 옵션 | 명령어 옵션 &
   $ exit
*/

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>

#define MAX_ARG 8
#define MAX_LINE 80

int shell_inputScan(char prompt[], char *cmd_list[]);
void shell_execute(int num, char *cmd_list[]);
int *shell_flag(int num, char *cmd_list[]);
void shell_input_redirect(char *cmd1[], char *cmd2[]);
void shell_output_redirect(char *cmd1[], char *cmd2[]);
void shell_pipe(char *cmd1[], char *cmd2[]);

int main(){ ...
}
```

shell.c untitled

```c

/*
   shell_flag()
   입력 받은 문자 스트림에 <, >, | 가 포함되었는지 알려주는 함수
   정수 배열 (플래그, 위치) 리턴.
   플래그>> 없음 : 0, > : 1, < : 2, | : 3.
*/
int *shell_flag(int num, char*cmd_list[]){
    int i;
    static int rtn[2] = {0, 0};
    for (i=0; i<num; i++) {
        if (!strcmp(cmd_list[i], ">")) {
            rtn[0] = 1;
            rtn[1] = i;
        }
        else if (!strcmp(cmd_list[i], "<")) {
            rtn[0] = 2;
            rtn[1] = i;
        }
        else if (!strcmp(cmd_list[i], "|")) {
            rtn[0] = 3;
            rtn[1] = i;
        }
    }
    return rtn;
}


/*
   shell_execute()
   입력 : 리스트 사이즈, 리스트
   커맨드 리스트에서 명령어들을 구문하고, 명령어를 실행
*/
void shell_execute(int num, char *cmd_list1[]){
    char *cmd_list2[MAX_ARG];

    /*
       입력된 명령어에 따라 flag를 설정하고, flag에 맞는 명령어 실행
    */
    int i, status, background = 0;
    pid_t cpid;

```

shell.c untitled

```
137     pid_t cpid;
138
139     /* Background & 확인 */
140     if (!strcmp(cmd_list1[num-1], "&")) {
141         background = 1;
142         cmd_list1[num-1] = '\0';
143     }
144
145     if ((cpid = fork()) < 0) {
146         printf("$ 프로세스 생성 오류\n");
147         exit(1);
148     }
149     else if (cpid == 0) {
150         /* 손자 프로세스 */
151         /* 리다이렉션이나 파이프 명령어 체크 */
152         /* flags = {flag, 위치} // 노멀 0, > 1, < 2, | 3 */
153         int *flags = shell_flag(num, cmd_list1);
154
155         /* 리다이렉션이나 파이프가 있는 경우,
156            cmd_list[]를 잘라서 cmd_argv를 생성
157            */
158         if (flags[0] != 0) {
159             for (i=flags[1]+1; i<num; i++) {
160                 cmd_list2[i-flags[1]-1] = cmd_list1[i];
161             }
162             cmd_list2[num-flags[1]+1] = NULL; // 파일명
163             cmd_list1[flags[1]] = NULL; //(명령어 옵션)
164         }
165         switch (flags[0]) {
166             case 0: /* 단일 명령어 실행인 경우 */
167                 execvp(cmd_list1[0], cmd_list1);
168                 exit(0);
169                 break;
170             case 1: /* input_redirection > */
171                 shell_input_redirect(cmd_list1, cmd_list2);
172                 exit(0);
173                 break;
174             case 2: /* output_redirection < */
175                 shell_output_redirect(cmd_list1, cmd_list2);
176                 exit(0);
177                 break;
178             case 3: /* pipe */
179                 shell_pipe(cmd_list1, cmd_list2);
```

shell.c untitled

```
176                 exit(0);
177                 break;
178             case 3: /* pipe */
179                 shell_pipe(cmd_list1, cmd_list2);
180                 exit(0);
181                 break;
182         }
183     }
184     else {
185         /* & 연산자를 위해 WNOHANG 옵션 */
186         if (background) {
187             waitpid(cpid, &status, WNOHANG);
188         }
189         else {
190             wait(NULL);
191         }
192     }
193 }
194 /* 명령어 + 옵션 > 파일명 */
195 void shell_input_redirect(char *cmd1[], char *cmd2[]){
196     int fd;
197 
198     if ((fd = open(cmd2[0], O_RDWR | O_CREAT | S_IROTH, 0644)) < 0) {
199         printf("$ 파일 생성 오류\n");
200         exit(1);
201     }
202     dup2(fd, STDOUT_FILENO);
203     execvp(cmd1[0], cmd1);
204     close(fd);
205     exit(0);
206 }
207 /* 명령어 + 옵션 < 파일명 */
208 void shell_output_redirect(char *cmd1[], char *cmd2[]){
209     int fd, num, i;
210     char buf[MAX_LINE];
211     char *list[MAX_ARG];
212     printf("%s", cmd1[1]);
213 
214     if (fd = open(cmd2[0], O_RDONLY) < 0) {
215         printf("$ 존재하지 않는 파일명 입니다. \n");
216         exit(1);
217     }
218     if (read(fd, buf, MAX_LINE) < 0) {
```

shell.c untitled

```
217     }
218     if (read(fd, buf, MAX_LINE) < 0) {
219         printf("$ 파일을 읽지 못하였습니다.\n");
220     }
221     num = shell_inputScan(buf, list);
222     for (i=num+1; i>0; i--) {
223         list[i] = list[i-2];
224     }
225 
226     list[0] = cmd1[0];
227     list[1] = cmd1[1];
228     printf("%s", cmd1[1]);
229     list[num+1] = NULL;
230     execvp(list[0], list);
231     close(fd);
232     exit(0);
233 }
234 /* 명령어 + 옵션 | 명령어 + 옵션 */
235 void shell_pipe(char *cmd1[], char *cmd2[]){
236     int fd[2];
237     if (pipe(fd) == -1) {
238         printf("$ 파이프 생성 오류 \n");
239     }
240     if (fork() == 0) {
241         close(STDOUT_FILENO);
242         dup2(fd[1], STDOUT_FILENO);
243         close(fd[0]);
244         close(fd[1]);
245         execvp(cmd1[0], cmd1);
246         exit(1);
247     }
248     if (fork() == 0) {
249         close(STDIN_FILENO);
250         dup2(fd[0], STDIN_FILENO);
251         close(fd[1]);
252         close(fd[0]);
253     }
254     close(fd[0]);
255     close(fd[1]);
256     wait(NULL);
257     wait(NULL);
258 }
259 
```

Line 205, Column 13 Tab

```c
/*
   shell_inputScan()
   사용자의 입력이 기록되어 있는 prompt에서 명령어(단어)들을 토큰화 하는 함수
   입력 규격에 대해서는 최대 인자 개수만 체크
   토큰 개수를 리턴, exit의 경우 -1를 리턴하며 규격 오류는 0 리턴
*/
int shell_inputScan(char prompt[], char *cmd_list[]){
    /* 입력 스트림에 있는 \n을 잘라냄 */
    if(prompt[strlen(prompt)-1] == '\n') prompt[strlen(prompt)-1] = '\0';

    /* EXIT 입력을 받은 경우 */
    if (!strcmp(prompt, "exit")) {
        cmd_list[0] = prompt;
        return -1;
    }

    int i=0;
    char *ptr = strtok(prompt, " ");

    while(ptr != NULL){
        if (i+1 > MAX_ARG) return 0; /* 최대 인자 초과 */
        cmd_list[i] = ptr;
        i++;
        ptr = strtok(NULL, " ");
    }
    return i;
}
```