운영체제론 Project #1

2017012251 윤영훈

1. 소스코드

[헤더 및 초기선언]

```
1   #include <stdio.h>
2   #include <unistd.h>
3   #include <string.h>
4   #include <sys/wait.h>
5   #include <fcntl.h>
6
7   #define MAX_LINE 80 /* The maximum length command */
8   #define READ_END 0
9   #define WRITE_END 1
10
11  int main(void)
12  {
13      char *args[MAX_LINE/2 + 1]; // command line arguments
14      char *args_pipe[MAX_LINE/2 + 1]; // arguments for pipe
15      int should_run = 1; // flag to determine when to exit program
16      char buffer[MAX_LINE]; // sub string to make args
17      int num = 0, background = 0, status;
18      int pipe_index = 0; // index of |
19      pid_t pid;
```

strcpm, strtok 함수 사용을 위해 line 3. #include <string.h>를 추가.
터미널 환경에서 wait 함수 사용 시 오류 발생을 막기 위해 line 4. #include <sys/wait.h>를 추가.
mode_t 타입 선언을 위해 line 5. #include <fcntl.h>를 추가.

[args 짜깁기]

```
21      while (should_run) {
22          printf("osh>");
23          fflush(stdout);
24          fflush(stdin);
25
26          fgets(buffer, sizeof(buffer), stdin);
27
28          // 28 - 40 : Make args using buffer and strtok
29          if (buffer[0] == '\n')
30              continue;
31          else
32              buffer[strlen(buffer) - 1] = '\0';
33
34          args[0] = strtok(buffer, " ");
35          if (strcmp(args[0], "quit") == 0)
36              break;
37
38          while (args[num] != NULL){
39              args[++num] = strtok(NULL, " ");
40          }
41
42          if (strcmp(args[num-1], "&") == 0){ // If there are background sign &,
43              background = 1;                 // Delete & sign and background = 1
44              num--;
45              args[num] = NULL;
46          }
```

fgets 함수를 통해 buffer에 한 줄 형식의 명령어 및 옵션을 받음.

만약 아무 입력 없이 ENTER만 눌러서 개행 문자만 입력된 경우 명령어를 다시 입력받고, 그 외의 경우에는 개행 문자 제거를 위해 line 32. buffer[strlen(buffer) – 1] = '\0'; 처리.

기본적으로 일반 명령어, Redirect 명령어, Pipe 명령어 세가지 명령어만 들어온다고 가정하고 처리하였고, Redirect, Pipe 명령어는 변경사항 체크를 위해 한번 실행 후 프로그램이 종료되도록 하였고 일반 명령어는 명령어 하나가 끝나더라도 다시 입력받을 수 있게 작성. 이때 quit 입력 시 프로그램을 종료할 수 있도록 line 35-36을 작성.

buffer는 띄어쓰기를 포함한 String 한 줄이므로 띄어쓰기를 기준으로 명령어 나눠 args 배열에 입력.

Background 실행 명령어인 &가 명령어에 존재한다면 기존 0으로 선언된 background 변수를 1로 변경하여 뒷부분의 PARENT를 실행할 때 적용. 그 후 명령어의 &를 제거함.

[CASE : Redirect 〉 or 〈]

```
48          for (int i=0; i < num; i++){
49              if (strcmp(args[i], ">") == 0 | strcmp(args[i], "<") == 0){
50                  int fd;
51                  mode_t mode = S_IRUSR | S_IWUSR | S_IXUSR | S_IRGRP | S_IROTH;
52                  if (strcmp(args[i], ">") == 0){                                  // Case 1 : >
53                      if ((fd = open(args[i+1], O_CREAT | O_WRONLY | O_TRUNC, mode)) == -1){
54                          perror("OPEN ERROR AT >");
55                          break;
56                      }
57                      dup2(fd, STDOUT_FILENO);
58                      close(fd);
59                  }
60                  else{                                                            // Case 2 : <
61                      if ((fd = open(args[i+1], O_RDONLY, mode)) == -1){
62                          perror("OPEN ERROR AT <");
63                          break;
64                      }
65                      dup2(fd, STDIN_FILENO);
66                      close(fd);
67                  }
68                  args[i] = NULL;
69                  num -= 2;
70                  should_run = 0;                 // To check the change, stop program. ( > or < )
71              }
```

명령어에 Redirection sign 〉 또는 〈 가 입력되어 있다면 file descriptor fd를 선언하여

Case 〉 :

O_CREAT – 파일이 존재하지 않으면 생성

O_WRONLY – 쓰기 전용

O_TRUNC – 파일이 존재 시 자름

Case 〈 :

O_RDONLY : 읽기 전용

Case에 따른 각 Flag 및 Mode를 적용하여 파일을 Open.

이때, Mode = S_IRUSR | S_IWUSR | S_IXUSR | S_IRGRP | S_IROTH

USER	GROUP + OTHER
Read, Write, Execute	Read

다음과 같은 권한을 적용한다.

그 후 각각 dup2를 통해 표준입출력 STDOUT_FILENO / STDIN_FILENO으로 fd를 복사한 뒤 fd를 닫아준다. 그 후 args를 다시 정리하고 Redirection 명령어는 변경사항을 확인하기 위해 한번 실행 후 프로그램을 닫아주어야 하므로 line 70. should_run = 0;을 적용한다.

[CASE : PIPE |]

```
72              else if (strcmp(args[i], "|") == 0){                                    // Case 3 : | (Pipe)
73                  pipe_index = i;
74                  args[i] = NULL;
75                  int k=0;
76                  while (k < num - i - 1){        // If command is A | B, Make A -> args / B -> args_pipe
77                      args_pipe[k] = args[i + k + 1];
78                      args[i + k + 1] = NULL;
79                      k++;
80                  }
81                  if (pipe_index != i)
82                      num++;
83                  num = num - k - 1;
84                  args_pipe[k] = NULL;
85                  should_run = 0;             // To check the change, stop program. ( PIPE)
86              }
87          }
```

명령어에 PIPE sign | 가 입력되어 있다면 제거 후 line 76-80을 통해 args_pipe에 PIPE sign | 오른쪽에 위치한 명령어를 저장한다.

ex) cat sort.txt | grep 3 -〉

args	args_pipe
cat sort.txt	grep 3

그 후 num, args_pipe를 채워진 내용에 맞게 정리한 뒤 PIPE 명령어 역시 변경사항 확인을 위해 실행 후 종료를 line 85. should_run = 0;로 적용한다.

[fork (1)]

```
89          pid = fork();
90
91          if (pid < 0){                                   // Fork error
92              perror("FORK FAILED");
93              return -1;
94          }
95          else if (pid == 0){                             // CHILD
96              if (pipe_index != 0){                       // IF PIPE
97                  int pipe_fd[2];
98                  pid_t pid_pipe;
99
100                 if (pipe(pipe_fd) == -1){
101                     perror("PIPE FAILED");
102                     return -1;
103                 }
104
105                 pid_pipe = fork();
```

line 89. pid = fork();를 통해 fork를 진행해주고 이때의 pid 값을 이용해 Child, Parent를 구분한다. 만약 pid == 0 이라면 Child 섹션에 진입하고, line 96. if (pipe_index != 0){를 통해 PIPE인지 구분한다. (pipe_index의 기본값은 0으로 선언했으므로 0이 아니라면 PIPE이다.)

만약 PIPE라면 Parent, Child 간 전달을 위해 pipe_fd를 선언 후 pid_pipe를 통해 다시한번 fork를 진행한다.

```
107                 if (pid_pipe < 0){
108                     perror("PIPE FORK FAILED");
109                     return -1;
110                 }
111                 else if (pid_pipe == 0){        // PIPE_CHILD
112                     close(pipe_fd[WRITE_END]);
113                     dup2(pipe_fd[READ_END], STDIN_FILENO);
114                     close(pipe_fd[READ_END]);
115                     status = execvp(args_pipe[0], args_pipe);
116                     if (status == -1){
117                         perror("FAIL TO EXECUTE THE PIPE COMMAND
118                         return -1;
119                     }
120                 }
121                 else {                          // PIPE_PARENT
122                     close(pipe_fd[READ_END]);
123                     dup2(pipe_fd[WRITE_END], STDOUT_FILENO);
124                     close(pipe_fd[WRITE_END]);
125                 }
126             }
127             status = execvp(args[0], args);
128             if (status == -1){
129                 perror("FAIL TO EXECUTE THE COMMAND");
130                 return -1;
131             }
```

PIPE_CHILD의 경우 사용하지 않는 WRITE_END를 닫아주고, dup2를 통해 표준입력으로 pipe_fd[READ_END]를 복사하고 line 115. execvp(args_pipe[0], args_pipe);를 통해 args_pipe의 명령문을 실행한다.

PIPE_PARENT의 경우 READ_END를 닫아주고, dup2를 통해 표준출력으로 pipe_fd[WRITE_END]를 복사한다.

line 127. status = execvp(args[0], args);는 args의 명령어를 실행한다.

이때 execvp(args_pipe[0], args_pipe) 와 execvp(args[0], args);는 PIPE를 통해 정보를 교환하는데, execvp(args[0], args)의 Output이 execvp(args_pipe[0], args_pipe)의 Input으로 입력된다.

[PARENT & 초기화]

```
132         }
133         else{                                   // PARENT
134             if (background){                    // BACKGROUND, NO WAIT
135                 waitpid(pid, NULL, WNOHANG);
136                 printf("PID #%d IS WORKING IN BACKGROUND : %s\n", pid, buffer);
137             } else{                             // FOREGROUND, WAIT CHILD
138                 wait(&status);
139             }
140         }
141
142         num = 0;
143         background = 0;
144
145         /**
146          * After reading user input, the steps are:
147          * (1) fork a child process using fork()
148          * (2) the child process will invoke execvp()
149          * (3) parent will invoke wait() unless command included &
150          **/
151     }
152
153     return 0;
154 }
```

PARENT는 앞서 & 여부에 따라 입력했던 background의 값을 통해

CASE background = 1 (& O) :

CHILD를 기다리지 않음

CASE background = 0 (& X) :

CHILD를 기다림

의 과정을 수행한다.

그 후 일반 명령어의 경우를 위해 line 142-143의 초기화를 진행해주고 코드를 마무리한다.

2. 실행

[명령어+옵션 , 명령어+옵션 &]

ls -al , ls -al &

```
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ gcc -o osproj1_yh osproj1_yh.c
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>ls -al
합계 52
drwxrwxr-x 2 xion xion  4096  3월 30 23:43 .
drwxrwxr-x 5 xion xion  4096  3월 29 15:35 ..
-rwxr-xr-x 1 xion xion 13144  3월 30 23:36 main
-rwxr-xr-x 1 xion xion 13144  3월 30 23:43 osproj1_yh
-rw-rw-r-- 1 xion xion  3749  3월 30 23:40 osproj1_yh.c
-rwxr--r-- 1 xion xion   308  3월 30 23:01 out.txt
-rw-r--r-- 1 xion xion    14  3월 30 19:37 sort.txt
osh>ls -al &
PID #31780 IS WORKING IN BACKGROUND : ls
osh>합계 52
drwxrwxr-x 2 xion xion  4096  3월 30 23:43 .
drwxrwxr-x 5 xion xion  4096  3월 29 15:35 ..
-rwxr-xr-x 1 xion xion 13144  3월 30 23:36 main
-rwxr-xr-x 1 xion xion 13144  3월 30 23:43 osproj1_yh
-rw-rw-r-- 1 xion xion  3749  3월 30 23:40 osproj1_yh.c
-rwxr--r-- 1 xion xion   308  3월 30 23:01 out.txt
-rw-r--r-- 1 xion xion    14  3월 30 19:37 sort.txt

osh>quit
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$
```

정상적으로 실행 완료

ls -al &의 경우 PID #31780 IS WORKING IN BACKGROUND : ls 메시지가 출력됨.

(실제 출력 여부를 위해 Background 실행이지만 내용을 출력했기 때문에 ➡ 부분에 osh〉 와 출력 내용이 겹치는 현상 발생)

[명령어+옵션 〉 파일명 , 명령어+옵션 〉 파일명 &]

ls -al 〉 out.txt , ls -al 〉 out.txt &

```
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ gcc -o osproj1_yh osproj1_yh.c
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>ls -al > out.txt
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>ls -al > out2.txt &
```

정상적으로 실행 완료

out.txt

```
합계 48
drwxrwxr-x 2 xion xion  4096  3월 30 23:47 .
drwxrwxr-x 5 xion xion  4096  3월 29 15:35 ..
-rwxr-xr-x 1 xion xion 13144  3월 30 23:36 main
-rwxr-xr-x 1 xion xion 13144  3월 30 23:46 osproj1_yh
-rw-rw-r-- 1 xion xion  3749  3월 30 23:40 osproj1_yh.c
-rwxr--r-- 1 xion xion     0  3월 30 23:47 out.txt
-rw-r--r-- 1 xion xion    14  3월 30 19:37 sort.txt
```

out2.txt

```
PID #31816 IS WORKING IN BACKGROUND : ls
합계 56
drwxrwxr-x 2 xion xion  4096  3월 30 23:47 .
drwxrwxr-x 5 xion xion  4096  3월 29 15:35 ..
-rwxr-xr-x 1 xion xion 13144  3월 30 23:36 main
-rwxr-xr-x 1 xion xion 13144  3월 30 23:46 osproj1_yh
-rw-rw-r-- 1 xion xion  3749  3월 30 23:40 osproj1_yh.c
-rwxr--r-- 1 xion xion   369  3월 30 23:47 out.txt
-rwxr--r-- 1 xion xion    41  3월 30 23:47 out2.txt
-rw-r--r-- 1 xion xion    14  3월 30 19:37 sort.txt
```

[명령어+옵션 〈 파일명 , 명령어+옵션 〈 파일명 &]

sort -n 〈 sort.txt , sort -n 〈 sort.txt &

```
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ gcc -o osproj1_yh osproj1_yh.c
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>sort -n < sort.txt
1
22
30
53
79
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>sort -n < sort.txt &
PID #31864 IS WORKING IN BACKGROUND : sort
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ 1
22
30
53
79
```

정상적으로 실행 완료

sort.txt 내부에는

53
1
79
22
30

다음과 같은 숫자의 나열이 개행을 구분으로 존재함.

〈 sort.txt를 통해 가져온 53, 1, 79, 22, 30을 sort -n 명령어를 통해 오름차순으로 정렬.

[명령어+옵션 | 명령어+옵션 , 명령어+옵션 | 명령어+옵션 &]

cat sort.txt | grep 3 , cat sort.txt | grep 3 &

```
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ gcc -o osproj1_yh osproj1_yh.c
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>cat sort.txt | grep 3
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ 53
30

xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ ./osproj1_yh
osh>cat sort.txt | grep 3 &
PID #31886 IS WORKING IN BACKGROUND : cat
xion@xion-VirtualBox:~/eclipse-workspace/osproj/src$ 53
30
```

정상적으로 실행 완료.

cat sort.txt를 통해 내부의 값 53, 1, 79, 22, 30을 불러옴.

이 값을 PIPE를 통해 뒤의 grep 3에게 전달, grep 3 명령어를 통해 53, 1, 79, 22, 30 중 3이 포함된 53, 30을 반환해줌.