[Systemp Programming] SHLAB

2017-19428

컴퓨터공학과 서준원

0. Overview

이 과제는 간단한 Unix Shell Program을 직접 짜보는 과제였다. Shell은 간단하게 얘기하자면 프로그램을 실행해주는 프로그램이다. 나는 윈도우 운영체제에서 Bash Shell을 사용하곤 하는데, 이번 과제를 통해서 Shell이 무엇이고, 어떻게 작동하는 지를 구체적으로 알 수 있었다.

또한, 이 과제에서는 간단한 버전의 시그널 핸들링(Signal Handling)을 구현해야 했다. 강의자료에 따르면 시그널과 그것을 처리하는 메커니즘은 아래와 같다.

- A signal is a small message that notifies a process that an event of some type has occurred in the system
- Kernel sends (delivers) a signal to a destination process by updating some state in the context of the destination process
- A destination process receives a signal when it is forced by the kernel to react in some way to the delivery of the signal

즉, 시그널은 프로세스에게 전달되는 시스템 메시지로, 시그널 함수를 작성함을 통해 시그널을 Catch할 수 있었다. Shell은 Child Process에서 execve 함수를 통해 프로그램을 실행하므로, SIGCHLD를 핸들링 하는 것이 주 된 내용이었다.

과제에서 구현해야 하는 부분들은 아래와 같았다. 코드와 간단한 설명, 그리고 배운 점들을 기록해놓았다.

1. Shell Lab

A. eval

터미널로 명령어가 들어왔을 때, 명령어를 분석하고 실행하기 위한 주된 분석이 이루어지는 코드이다. parseline 함수를 통해 인자와 명령어를 구분하고, 주어진 명령에 따라 Child Process를 만들어서 프로그램을 실행시킨다. Fork 함수를 통해 Child Process를 만들 수 있었다.

시스템 콜을 실행하는 다양한 함수에서 알 수 없는 이유로 시스템 콜이 실패할 수 있다. 이 때 리턴 값은 음수이기 때문에 이를 핸들링 하기 위해 리턴 값을 모두 체크하였다. sigemptyset, sigprocmask, kill 등의 함수들을 래핑하였다.

프로세스가 백그라운드로 실행될 때는 기다리지 않고, 포어그라운드로 실행되었을 때만 그 프로세스가 끝날 때 까지 기다리는 작업을 실행했다.

Race 관련한 문제를 해결하기 위해 시그널 블록킹 작업을 적절하게 진행했다. 이는 addset 전에 deleteset이 일어나는 경우를 방지하는 것이다.

```
2. void eval(char *cmdline)
3. {
4.
5.     char *argv[MAXARGS];
6.
7.     int bg = parseline(cmdline,argv); // is background?
8.     sigset_t mask;
9.     if(!builtin_cmd(argv)) //builtin 아닐 경우 실행 -> fork child
10.    {
11.        if(sigemptyset(&mask) <0)
12.        {
13.            unix_error("System Call Error");
14.            exit(0);
15.        }
16.        if (sigaddset(&mask, SIGCHLD) < 0)
17.        {
18.            unix_error("System Call Error");
19.            exit(0);
```

```
20.         }
21.         if(sigprocmask(SIG_BLOCK, &mask, NULL) <0)
22.         {
23.             unix_error("System Call Error");
24.             exit(0);
25.         }
26.
27.         pid_t pid;
28.         if( (pid = fork()) == 0) // child
29.         {
30.
31.             if(sigprocmask(SIG_UNBLOCK, &mask, NULL) < 0)
32.             {
33.                 unix_error("System Call Error");
34.                 exit(0);
35.             }
36.             setpgid(0,0); // 현재 child의 PID로 PGID 만든다. ->
    Assignment Guide
37.             if(execve(argv[0],argv,environ) < 0)
38.             {
39.                 printf("%s: Command not found.\n", argv[0]);
40.                 exit(0);
41.             }
42.         }
43.
44.         if(!bg)
45.         {//foreground -> wait pid
46.             addjob(jobs, pid, FG, cmdline);
47.             if(sigprocmask(SIG_UNBLOCK, &mask, NULL) < 0)
48.             {
49.                 unix_error("System Call Error");
50.                 exit(0);
51.             }
52.             waitfg(pid);
53.         }
54.         else
55.         {//background -> No wait, but PRINT
56.             addjob(jobs, pid, BG, cmdline);
```

```
57.
58.             if(sigprocmask(SIG_UNBLOCK, &mask, NULL) < 0)
59.             {
60.                 unix_error("System Call Error");
61.                 exit(0);
62.             }
63.             printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);
64.         }
65.     }
66.     return;
67. }
```

B. Builtin_cmd

이 과제에서 제공되는 명령어들을 한 번에 처리할 수 있도록 구현한 부분이다. quit, bg, fg, jobs가 주어진 경우 따로 처리하도록 하였다.

```
0. int builtin_cmd(char **argv)
1. {
2.     if(strcmp(argv[0], "quit") == 0)
3.     {
4.         exit(0);
5.     }
6.
7.     else if(strcmp(argv[0], "fg") == 0 || strcmp(argv[0], "bg") ==
   0)
8.     {
9.         do_bgfg(argv);
10.         return 1;
11.     }
12.
13.     else if(strcmp(argv[0], "jobs") == 0)
14.     {
15.         listjobs(jobs);
16.         return 1;
17.     }
18.
```

```
19.     return 0;     /* not a builtin command */
20. }
```

C. do_bgfg

위에서 bg나 fg 명령어를 주었을 경우 처리하는 부분이다. 과제 스펙에 따르면 위 명령어는 다음과 같은 작업을 처리해야 한다.

- The bg <job> command restarts <job> by sending it a SIGCONT signal, and then runs it in the background. The <job> argument can be either a PID or a JID.

– The fg <job> command restarts <job> by sending it a SIGCONT signal, and then runs it in the foreground. The <job> argument can be either a PID or a JID.

따라서 bg나 fg 명령어가 들어왔을 때, job이나 process의 ID를 찾아서 필요한 작업을 진행했다. atoi와 isdigit 함수를 활용했다.

```
0. void do_bgfg(char **argv)
1. {
2.     struct job_t *temp;
3.
4.     if(argv[1] == NULL)
5.     { // No Argument
6.         printf("%s command requires PID or %%jobid argument\n",
   argv[0]);
7.         return;
8.     }
9.
10.    if(argv[1][0] == '%')
11.    { // by job id
12.        int jobID = atoi(&argv[1][1]);
13.        temp = getjobjid(jobs, jobID);
14.        if(temp == NULL)
15.        {
16.            printf("%s: No such job\n", argv[1]);
17.            return;
18.        }
```

```
19.     }
20.     else if(isdigit(argv[1][0]))
21.     { // by PID
22.         pid_t pid = atoi(argv[1]);
23.         temp = getjobpid(jobs, pid);
24.         if(temp == NULL)
25.         {
26.             printf("(%d): No such process\n", pid);
27.             return;
28.         }
29.     }
30.     else
31.     {
32.         printf("%s: argument must be a PID or %%jobid\n", argv[0]);
33.         return;
34.     }
35.
36.     if(kill(-temp->pid, SIGCONT) < 0)
37.     {
38.         unix_error("System Call Error");
39.         exit(0);
40.     } // RESTART , no - is ok!
41.
42.     if(strcmp(argv[0], "fg") == 0)
43.     {
44.         temp->state = FG;
45.         waitfg(temp->pid);
46.     }
47.     else
48.     {
49.         printf("[%d] (%d) %s", temp->jid, temp->pid, temp->cmdline);
50.         temp->state = BG;
51.     }
52.
53. }
```

D. waitfg

포어그라운드의 프로세스가 종료될 때 까지 기다리는 부분이다. while문을 통해 해당 pid의 process가 끝날 때 까지 기다린다. 스켈레톤 코드에서 제공 된 fgpid 함수를 이용했다. 이 함수는 포어그라운드의 프로세스 아이디를 리턴한다. 또한 아무런 프로세스가 없을 땐 0을 리턴한다. 따라서 포어그라운드 프로세스가 끝나면 루프를 빠져나가게 된다.

```
0. void waitfg(pid_t pid)
1. {//eval 에서 실행됨
2. // waitfg - Block until process pid is no longer the foreground
   process
3.
4.     struct job_t *temp;
5.
6.     temp = getjobpid(jobs, pid);
7.
8.     if(pid == 0)
9.     { // child
10.        return;
11.    }
12.    if(temp != NULL)
13.    {
14.        while(pid == fgpid(jobs)){/*sleep*/}
15.    }
16.    return;
17. }
```

E. sigchld handler: Catches SIGCHILD signals.

F. sigint handler: Catches SIGINT (ctrl-c) signals.

G. sigtstp handler: Catches SIGTSTP (ctrl-z) signals.

시그널 핸들링을 하는 부분이다. 주어진 시그널이 들어왔을 때, Child Process를 Reap 해주는 과정을 거친다. 이는 waitpid 함수를 통해 이루어진다.

SIGCHLD, SIGSTP(ctrl + z), SIGINT(ctrl + c) 총 세 가지 경우의 시그널을 핸들링하도록 구현하였다.

waitpid 함수에서 status를 저장하여 그 status에 따라 처리를 다르게 했다. 정지된 경우, 끝난 경우 등 다양한 경우를 처리했다.

또한 Async 관련하여 원칙상 printf 함수를 사용하면 안된다. 하지만 큰 문제가 없는 것 같아 사용했다.

```
0. void sigchld_handler(int sig)
1. {
2.     // use exactly one call to waitpid
3.     pid_t pid;
4.     int status;
5.     int jobID;
6.     while((pid = waitpid(fgpid(jobs), &status, WNOHANG|WUNTRACED)) >
   0)
7.     {
8.         if(WIFSTOPPED(status)/*child 가 정지된 상태*/)
9.         {
10.            getjobpid(jobs, pid)->state = ST;
11.            jobID = pid2jid(pid);
12.            printf("Job [%d] (%d) stopped by signal %d\n", jobID,
   pid, WSTOPSIG(status));
13.            //print
14.        }
15.        else if(WIFSIGNALED(status)/*child is signaled*/)
16.        {
17.            jobID = pid2jid(pid);
18.            printf("Job [%d] (%d) terminated by signal %d\n", jobID,
   pid, WTERMSIG(status));
19.            //print
20.            deletejob(jobs, pid);
21.        }
22.        else if(WIFEXITED(status)/*child is Exited*/)
23.        {
24.            deletejob(jobs, pid);
25.        }
26.    }
27.    return;
28. }
```

```
29.
30. /*
31.  * sigint_handler - The kernel sends a SIGINT to the shell whenver
    the
32.  *    user types ctrl-c at the keyboard.  Catch it and send it along
33.  *    to the foreground job.
34.  */
35. void sigint_handler(int sig)
36. {// foreground 의 process 를 종료
37.     pid_t pid = fgpid(jobs);
38.
39.     if(pid != 0)
40.     {
41.         if(kill(-pid, sig)<0)
42.         {
43.             unix_error("System Call Error");
44.             exit(0);
45.         }
46.     }
47.     return;
48. }
49.
50. /*
51.  * sigtstp_handler - The kernel sends a SIGTSTP to the shell
    whenever
52.  *     the user types ctrl-z at the keyboard. Catch it and suspend
    the
53.  *     foreground job by sending it a SIGTSTP.
54.  */
55. void sigtstp_handler(int sig)
56. {
57.     pid_t pid = fgpid(jobs);
58.     if(pid != 0)
59.     {
60.         if(kill(-pid, sig)<0)
61.         {
62.             unix_error("System Call Error");
63.             exit(0);
```

```
64.         }
65.     }
66.     return;
67. }
```

2. Verification

주어진 총 16개의 테스트를 통해 레퍼런스 코드의 결과와 일치하는지 체크하였다. 그 중 보기 좋은 두 가지 경우 결과를 첨부하였다. 과제 스펙 상 pid를 제외하고 모든 글자가 일치해야 하기 때문에 두 눈을 크게 뜨고 검사하였다. 대문자와 소문자를 잘 못 쓴 것을 확인하는 등 큰 수확이 있었다.

또한 직접 tsh 프로그램을 실행하여 볼 수도 있다. 이 경우 tsref 프로그램을 실행하여 결과를 대조해 보았다.

A. Test 16

```
givenone@DESKTOP-R88FHM3:/mnt/c/Users/user/Desktop/sp/shlab/handout$ make test16
./sdriver.pl -t trace16.txt -s ./tsh -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (690) stopped by signal 20
tsh> jobs
[1] (690) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (693) terminated by signal 2
^Z
[1]+  Stopped                 make test16
givenone@DESKTOP-R88FHM3:/mnt/c/Users/user/Desktop/sp/shlab/handout$ make rtest16
./sdriver.pl -t trace16.txt -s ./tshref -a "-p"
#
# trace16.txt - Tests whether the shell can handle SIGTSTP and SIGINT
#     signals that come from other processes instead of the terminal.
#
tsh> ./mystop 2
Job [1] (699) stopped by signal 20
tsh> jobs
[1] (699) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (702) terminated by signal 2
```

이는 SIGINT와 SIGSTP 시그널을 잘 처리하는 지 보는 테스트 코드이다. 잘 처리한 것을 확인했다. 20(SIGSTP), 2(SIGINT)를 잘 처리했다.

B. Test 10

```
givenone@DESKTOP-R88FHM3:/mnt/c/Users/user/Desktop/sp/shlab/handout$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (708) ./myspin 4 &
tsh> fg %1
Job [1] (708) stopped by signal 20
tsh> jobs
[1] (708) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
givenone@DESKTOP-R88FHM3:/mnt/c/Users/user/Desktop/sp/shlab/handout$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (718) ./myspin 4 &
tsh> fg %1
Job [1] (718) stopped by signal 20
tsh> jobs
[1] (718) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

이 코드는 fg 커맨드를 잘 처리하는 지 보는 코드이다. fg를 실행하므로서 백그라운드 프로세스가 포어그라운드 프로세스로 전환이 되었다.

3. Review

Shell 프로그램이 무엇인지 정확히 이해하고, 컴퓨터 시스템이 시그널 핸들링을 어떻게 하는지 더 잘 이해할 수 있는 계기가 되었다.

다만, 처음에는 코드가 너무 방대하여 어떻게 시작할 지 몰라 어려움을 겪기도 했다. 작은 부분을 놓치면 중대한 에러가 나기 때문이다. 숨을 가다듬고 천천히 그리고 열심히 한 결과 좋은 코딩 경험이 생긴 것 같다.

또한 프린트 포맷을 직접 알아내야 하는 과정에서 큰 어려움이 있었다. 그냥 과제에 처음부터 주어져도 되지 않을 까 싶기도 하다.

그래도 많은 것을 배울 수 있었다.