시스템 프로그래밍 Lab2 Report

경영학과 2017-15108
박지상

1. 실행결과

```
2017-15108@sp2:~/shlab$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
2017-15108@sp2:~/shlab$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
2017-15108@sp2:~/shlab$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
2017-15108@sp2:~/shlab$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (2630705) ./myspin 1 &
```

```
2017-15108@sp2:~/shlab$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (2630744) ./myspin 2 &
tsh> ./myspin 3 &
[2] (2630746) ./myspin 3 &
tsh> jobs
[1] (2630744) Running ./myspin 2 &
[2] (2630746) Running ./myspin 3 &
2017-15108@sp2:~/shlab$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (2630814) terminated by signal 2
2017-15108@sp2:~/shlab$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (2630854) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2630856) terminated by signal 2
tsh> jobs
[1] (2630854) Running ./myspin 4 &
```

```
2017-15108@sp2:~/shlab$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (2630954) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2630956) stopped by signal 20
tsh> jobs
[1] (2630954) Running ./myspin 4 &
[2] (2630956) Stopped ./myspin 5
2017-15108@sp2:~/shlab$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (2631021) ./myspin 4 &
tsh> ./myspin 5
Job [2] (2631023) stopped by signal 20
tsh> jobs
[1] (2631021) Running ./myspin 4 &
[2] (2631023) Stopped ./myspin 5
tsh> bg %2
[2] (2631023) ./myspin 5
tsh> jobs
[1] (2631021) Running ./myspin 4 &
[2] (2631023) Running ./myspin 5
```

1. 실행결과

```
2017-15108@sp2:~/shlab$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (2631100) ./myspin 4 &
tsh> fg %1
Job [1] (2631100) stopped by signal 20
tsh> jobs
[1] (2631100) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
2017-15108@sp2:~/shlab$ make test11
./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (2631141) terminated by signal 2
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
    762 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 hvc0 vt220
   1151 tty6     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty6 linux
   1343 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
2564444 pts/2    Ss+    0:00 -bash
2624063 pts/3    Ss+    0:00 -bash
2627256 pts/0    Ss+    0:00 -bash
2628921 pts/6    Ss     0:00 /usr/bin/bash
2631136 pts/6    S+     0:00 make test11
2631137 pts/6    S+     0:00 /bin/sh -c ./sdriver.pl -t trace11.txt -s ./tsh -a "-p"
2631138 pts/6    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace11.txt -s ./tsh -a -p
2631139 pts/6    S+     0:00 ./tsh -p
2631159 pts/6    R      0:00 /bin/ps a
```

```
2017-15108@sp2:~/shlab$ make test12
./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (2631188) stopped by signal 20
tsh> jobs
[1] (2631188) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
    762 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 hvc0 vt220
   1151 tty6     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty6 linux
   1343 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
2564444 pts/2    Ss+    0:00 -bash
2624063 pts/3    Ss+    0:00 -bash
2627256 pts/0    Ss+    0:00 -bash
2628921 pts/6    Ss     0:00 /usr/bin/bash
2631183 pts/6    S+     0:00 make test12
2631184 pts/6    S+     0:00 /bin/sh -c ./sdriver.pl -t trace12.txt -s ./tsh -a "-p"
2631185 pts/6    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace12.txt -s ./tsh -a -p
2631186 pts/6    S+     0:00 ./tsh -p
2631188 pts/6    T      0:00 ./mysplit 4
2631189 pts/6    T      0:00 ./mysplit 4
2631207 pts/6    R      0:00 /bin/ps a
```

1. 실행결과

```
2017-15108@sp2:~/shlab$ make test13
./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (2631240) stopped by signal 20
tsh> jobs
[1] (2631240) Stopped ./mysplit 4
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
    762 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 hvc0 vt220
   1151 tty6     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty6 linux
   1343 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
2564444 pts/2    Ss+    0:00 -bash
2624063 pts/3    Ss+    0:00 -bash
2627256 pts/0    Ss+    0:00 -bash
2628921 pts/6    Ss     0:00 /usr/bin/bash
2631235 pts/6    S+     0:00 make test13
2631236 pts/6    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
2631237 pts/6    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
2631238 pts/6    S+     0:00 ./tsh -p
2631240 pts/6    T      0:00 ./mysplit 4
2631241 pts/6    T      0:00 ./mysplit 4
2631259 pts/6    R      0:00 /bin/ps a
tsh> fg %1
tsh> /bin/ps a
    PID TTY      STAT   TIME COMMAND
    762 hvc0     Ss+    0:00 /sbin/agetty -o -p -- \u --keep-baud 115200,38400,9600 hvc0 vt220
   1151 tty6     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty6 linux
   1343 tty1     Ss+    0:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
2564444 pts/2    Ss+    0:00 -bash
2624063 pts/3    Ss+    0:00 -bash
2627256 pts/0    Ss+    0:00 -bash
2628921 pts/6    Ss     0:00 /usr/bin/bash
2631235 pts/6    S+     0:00 make test13
2631236 pts/6    S+     0:00 /bin/sh -c ./sdriver.pl -t trace13.txt -s ./tsh -a "-p"
2631237 pts/6    S+     0:00 /usr/bin/perl ./sdriver.pl -t trace13.txt -s ./tsh -a -p
2631238 pts/6    S+     0:00 ./tsh -p
2631277 pts/6    R      0:00 /bin/ps a
```

```
2017-15108@sp2:~/shlab$ make test14
./sdriver.pl -t trace14.txt -s ./tsh -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (2631285) ./myspin 4 &
tsh> fg
fg command requires PID or %jobid argument
tsh> bg
bg command requires PID or %jobid argument
tsh> fg a
fg: argument must be a PID or %jobid
tsh> bg a
bg: argument must be a PID or %jobid
tsh> fg 9999999
(9999999): No such process
tsh> bg 9999999
(9999999): No such process
tsh> fg %2
%2: No such job
tsh> fg %1
Job [1] (2631285) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (2631285) ./myspin 4 &
tsh> jobs
[1] (2631285) Running ./myspin 4 &
```

1. 실행결과

```
2017-15108@sp2:~/shlab$ make test15
./sdriver.pl -t trace15.txt -s ./tsh -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (2631331) terminated by signal 2
tsh> ./myspin 3 &
[1] (2631347) ./myspin 3 &
tsh> ./myspin 4 &
[2] (2631349) ./myspin 4 &
tsh> jobs
[1] (2631347) Running ./myspin 3 &
[2] (2631349) Running ./myspin 4 &
tsh> fg %1
Job [1] (2631347) stopped by signal 20
tsh> jobs
[1] (2631347) Stopped ./myspin 3 &
[2] (2631349) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (2631347) ./myspin 3 &
tsh> jobs
[1] (2631347) Running ./myspin 3 &
[2] (2631349) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
2017-15108@sp2:~/shlab$ 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] (2631395) stopped by signal 20
tsh> jobs
[1] (2631395) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (2631442) terminated by signal 2
```

2. 구현 방법 : eval()

```c
void eval(char *cmdline)
{
    // 0. Local Variable
    char* argv[MAXARGS];     /* holds arguments of command line */
    int bg;                  /* background job? */
    sigset_t blocked;        /* blocked signals */
    pid_t pid;               /* process id */

    // 1. Parse & Check cmd
    bg = parseline(cmdline, argv);
    if(bg == 1 && argv[0] == NULL) return; // blank line인 경우 종료
    if(builtin_cmd(argv)) return; // builtin_cmd인 경우 command 실행

    // 2. Block SIGCHLD Signal
    if(sigemptyset(&blocked) == -1) app_error("sigemptyset error"); // init
    if(sigaddset(&blocked, SIGCHLD) == -1) app_error("sigaddset error"); // add
    if(sigprocmask(SIG_BLOCK, &blocked, NULL) == -1) app_error("sigprocmask error");
```

cmdline의 input을 parse해서 그에 따른 처리를 한다.

sigemptyset, sigaddset, sigprocmask를 통해
SIGCHLD만을 block한 sig_set을 만든다.

Fork()를 한 뒤, 다시 sigchld를 unblock하고
Child는 argv에 따른 프로그램을 실행한다.
Parent는 child process를 jobs에 추가하고,
fg/bg에 따라 실행결과를 기다리거나 log를 출력한다.

```c
    // Loads and runs in the context of an initial child process
    // 3. Create Child Process
    pid = fork();

    if(pid == 0){
        /* Child Process */
        // 3.1 1) Unblock signal
        if(sigprocmask(SIG_UNBLOCK, &blocked, NULL) == -1) app_error("sigprocmask Error");

        // 3.1 2) Get new process group ID
        if(setpgid(0, 0) == -1) app_error("setpgid Error");

        // 3.1 3) Load & run new program
        if(execve(argv[0], argv, environ) < 0){
            printf("%s: Command not found\n", argv[0]);
            exit(0);
        }
    }else{
        /* Parent Process */
        // 3.2. 1) Addjob
        // 3.2. 2) Unblock signal
        // 3.3. 3) (if bg) print log message
        if(!bg){  /* parent waits for fg job to terminate */
            addjob(jobs, pid, FG, cmdline);
            if(sigprocmask(SIG_UNBLOCK, &blocked, NULL) == -1) app_error("SIG_UNBLOCK Error");
            waitfg(pid);
        } else{   /* otherwise, don't wait for bg job */
            addjob(jobs, pid, BG, cmdline);
            if(sigprocmask(SIG_UNBLOCK, &blocked, NULL) == -1) app_error("SIG_UNBLOCK Error");
            printf("[%d] (%d) %s", pid2jid(pid), (int) pid, cmdline);
        }
    }
```

2. 구현 방법 : builtin_cmd()

```
int builtin_cmd(char **argv)
{
    char* name = argv[0];

    if(strcmp(name, "quit") == 0){
        /* quit terminates the shell */
        exit(0);
    }else if(strcmp(name, "jobs")==0){
        /* jobs lists all background jobs */
        listjobs(jobs);
        return 1; // return true
    }else if(strcmp(name, "bg")==0 || strcmp(name, "fg")==0){
        /* bg <job> or fg <job> */
        do_bgfg(argv);
        return 1; // return true
    } else{
        /* not a builtin command */
        return 0; // return false
    }
}
```

cmdline으로 받은 argv 중
첫 번째 argument를 4가지 built-in 커맨드에
속하는지 판별한 뒤, 각자에 맞는 action을 수행한다.

Built-in 커맨드에 해당하는 경우 1 (true)를 리턴하여,
eval()에서 뒤의 커맨드를 실행하지 않고 종료할 수 있
도록한다.

2. 구현 방법 : do_bgfg()

```c
void do_bgfg(char **argv)
{
    // 0. Local Variable
    struct job_t  *obj;

    // Error Handling
    // 1. bg or fg command 뒤에 아무 인자가 없을 경우
    if(argv[1] == NULL){
        printf("%s command requires PID or %%jobid argument\n", argv[0]);
        return;
    }

    // 2. bg or fg command 뒤에 적절하지 않은 인자가 있을 경우
    if(argv[1][0] != '%' && !isdigit(argv[1][0])){
        printf("%s: argument must be a PID or %%jobid\n", argv[0]);
        return;
    }
```

```c
    // 3. Get Job Object
    if(argv[1][0] == '%'){
        // JobID
        obj = getjobjid(jobs, atoi(&argv[1][1]));
        if(obj == NULL){
            printf("%s: No such job\n", argv[1]);
            return;
        }
    } else{
        // PID
        obj = getjobpid(jobs, (pid_t) atoi(argv[1]));
        if(obj == NULL){
            printf("(%d): No such process\n", atoi(argv[1]));
            return;
        }
    }
```

```c
    // 4. Change the status of a stopped job
    int bg = strcmp(argv[0], "bg") == 0 ? 1 : 0;
    if(bg){
        obj->state = BG;
        printf("[%d] (%d) %s", obj->jid, obj->pid, obj->cmdline);
        if(kill(-obj->pid, SIGCONT) == -1){
            app_error("Kill Error");
        }
    } else{
        obj->state = FG;
        if(kill(-obj->pid, SIGCONT) == -1){
            app_error("Kill Error");
        }
        waitfg(obj->pid);
    }


    return;
```

먼저 정상적인 커맨드가 입력되었는지를 판별한다.

이후, argv의 두 번째 인자에 따라서 jobid로 지칭을 했는지, pid로 지칭을 했는지 판단한 뒤,
해당 id에 맞는 job object를 가져온다.

이후 커맨드에서 bg인지 fg인지 여부에 따라서 job의 state를 바꾼 뒤,
정지되어있던 프로세스에 SIGCONT 시그널을 보낸다.

2. 구현 방법 : waitfg()

```
void waitfg(pid_t pid)
{
    while(1){
        if(pid == fgpid(jobs))  sleep(1);
        else break;
    }
    return;
}
```

기다림의 대상이 되는 process가

Foreground에서 계속 실행되고 있는지 여부를

Fgpid(jobs) == pid로 판별한 뒤,

실행되고 있다면 sleep(1)을 반복해서

실행이 끝날 때까지 기다린 뒤,

끝난다면 while loop을 break후 return한다.

2. 구현 방법 : sigchld_handler()

```c
void sigchld_handler(int sig)
{
    int child_status;
    pid_t pid;
    int jobid;

    // Reap Child
    while((pid = waitpid(-1, &child_status, WNOHANG | WUNTRACED)) > 0){
        jobid = pid2jid(pid);

        if(WIFEXITED(child_status)){
            // 일반적으로 종료된 경우 - 미출력 (SIGCHLD의 default behavior는 ignore)
            deletejob(jobs, pid);
        } else if(WIFSIGNALED(child_status)){
            // SIGNAL에 의해 종료된 경우 - 출력
            deletejob(jobs, pid);
            printf("Job [%d] (%d) terminated by signal %d\n", jobid, (int) pid, WTERMSIG(child_status));


        } else if(WIFSTOPPED(child_status)){
            struct job_t  *obj = getjobpid(jobs, pid);
            obj->state = ST;
            printf("Job [%d] (%d) stopped by signal %d\n", jobid, (int) pid, WSTOPSIG(child_status));
        }
    }

    return;
}
```

종료된/정지된 모든 child process를 reap해서

Child_status에 따라 적절한 업무를 수행한다

- WIFEXITED(child_status): 일반적인 종료

- WIFSIGNALED(child_status): signal에 의한 종료

- WIFSTOPPED(child_status): process가 멈춘 경우

2. 구현 방법 : sigint_handler & sigtstp_handler

```c
void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs); // Foreground job

    if(pid != 0) kill(-pid, sig);
    return;
}
```

```c
void sigtstp_handler(int sig)
{
    pid_t pid = fgpid(jobs); // Foreground job
    if(pid != 0) kill(-pid, sig);
    return;
}
```

SIGINT와 SIGTSTP signal에 대해서

signal이 들어왔을 때 foreground에서 돌아가고 있는

Process와 해당 process group에 signal을 보낸다.

3. 어려웠던 점

- Kernel, Shell과 Process에 대한 개념이 명확히 확립되어 있지 않아, 초반에 과제를 이해하기 어려웠다.

- Process Control에 사용되는 여러 method를 파악하는 것이 어려웠다.
  - Signal(SIG, *handler)를 통한 signal handler install
  - sigemptyset(), sigaddset(), sigprocmask()을 통한 signal set의 관리
  - kill(pid, sig)를 통한 signal 전달
  - kill(pid, sig)에서 –pid로 할 경우 process group 모두에 대한 signal 전달
  - Exit(0)을 통한 process의 종료

- Sigchld_handle에서 다음과 같은 요소들을 파악하고 응용하는데 어려움을 겪었다.
  - WNOHANG, WUNTRACED와 같은 wait()의 Option
  - WIFEXITED, WIFSIGNALED, WIFSTOPPED, WTERMSIG, WSTOPSIG와 같은 매크로

4. 새롭게 배운 점

- Shell은 input을 받기 전에 tsh>와 같은 Prompt를 출력하고, 입력받은 input에 대해 정해진 절차를 수행하는 단순한 프로그램에 불과할 뿐이라는 사실을 알 수 있었다.

- Background / Foreground에서 process를 시작하고, signal을 통해서 process의 상태를 변경하거나 terminate 된 process를 reap할 수 있다. 또한 이 때 signal에 대해서 어떠한 signal을 받고 무시할지, 또한 받은 signal에 대해서 어떠한 시행을 할지 customize할 수 있다는 것을 배웠다.