2023 Fall CSED 211 Lab Report

Lab: Shell lab

Student number: 20220302

Name: 김지현

1. Introduction

- In this lab, I have implemented a shell using C. Shell lab provides students with a valuable opportunity to experience the functioning principles fo shell used in real os.

2. System Design

1. Eval function
   - This function takes the command line string as an argument. So, parse the command line and save it in the argument string array.
   - By the existence of the ampersand (“&”), the background or foreground is determined. This information is saved in the integer variable called “bg.”
   - First, check whether the input command is in a built-in command. If not, continue onto the if statement.
   - Fork() function will allow the parent process to create the child process. If pid, which contains the return value of the fok() function, equals to 0, then it is in the child process. If pid is greater than 0, then it is in the parent process.
   - In the child process, set the process group id to the present process id and run a different program within the same process using execve function.
   - If error happens while running execve function, print the error message “Command Not Found” and exit.
   - In the parent process, determine whether it is in the background or foreground using the variable, bg.
   - If in the background, add job to the background and print the job id, process id, and command line.
   - If in the foreground, add job to the foreground and wait.

2. Builtin_cmd function
   - This function checks whether the user input command is in built-in command.
   - First save the first input string, which will be the string saved in the first array block of argv.
   - If the input command is “quit,” terminate the process using exit function.
   - If the input command is “jobs,” print the job lists and returns 1.
   - If the input command is “fg” or “bg,” execute the do_fgbg function and returns 1.
   - Return 1 tells that the command is taken care in the builtin_cmd function.

3. Do_bgfg function
   - There are two conditions to check: command and id.

- First, id is the second input command (= argv[1]) and if it is null, there are no pid or %jid argument. Therefore, print the error message and return void.
- If id starts with %, then the jid is received. Using jid, find the job using getjobjid and if it doesn't exist, print an error message that such job does not exist.
- Otherwise, the input will be pid. If pid equals to 0, such pid does not exist, so print an error message. Else, continue the like the jid case except using getjobpid function.
- Next, determine which instruction is received for the command: fg or bg.
- If fg (foreground), change the job state into foreground and send "continue" signal to the process pid directs to. Then, wait.
- If bg (background), change the job state into background and print the job id, process id, command line. Then, send "continue" signal to the process in which pid directs to.

4. Waitfg function
   - If pid is in the foreground, continue to the while loop with sleep function.
   - Sleep function returns 0 if the requested time has elapsed. In the code, there will be a delay of one second.

5. Sigcld_handler
   - SIGCHLD is passed when the child process terminates but in the zombie state.
   - Waitpid function waits for the child process and returns the pid of child process when the child terminates and returns 0 otherwise.
   - If the child process terminates (or, waitpid function return value > 0), continue onto the while loop.
   - If WIFSTOPPED returns true, the process is stopped. Therefore, change the job state to stopped and print that the job is stopped by signal 20.
   - If WIFSIGNALED returns true, the process is terminated. Therefore, print a message that the process is terminated and deleted the job.
   - If WIFEXITED returns true, the process is terminated normally (by exit or return). So, delete the job.

6. Sigint_handler
   - SIGINT is sent when the user types ctrl-C. The default action is to terminate each process. Therefore, while pid is not equal to 0, send SIGINT signal to the processes.

7. Sigtstp_handler
   - SIGTSTP is sent when the user types ctrl-Z. The default action is to suspend each process. Similar to sigint_handler function, send SIGTSTP signal to the process.

3. Code Snippet:

```c
void eval(char *cmdline)
{
    char* argument [MAXARGS]; //saves the different command line arguments

    pid_t pid; //process id

    int bg = parseline(cmdline, argument); //parse the command line
                                           //and saves in the argument array
                                           //then returns whether it is in background for foreground (existence of &)

    struct job_t* job;

    if(!builtin_cmd(argument)){ //if not take cared by the built in command, move to User-defined function

        if((pid = fork())== 0) {  //parent process creates the child process
                                  //if pid = 0, in the child process.

            setpgid(0, 0); //setpgid function changes the process group ID to the pressent process ID

            if(execve(argument[0], argument, environ) < 0){ // execve function is to run same process but different program
                                                            // execve function does not have the return value,
                                                            // so only care about the error when execve < 0

                printf("%s : Command Not Found\n", argument[0]);
                exit(0);

            }

        }else{ //if pid not equal to 0, in the parent process


            if(bg){ //in the background

                addjob(jobs, pid, BG, cmdline);  //add job
                printf("[%d] (%d) %s", pid2jid(pid), pid, cmdline);  //print "[job id] (process id) commandline"

            }else{ // in the foreground

                addjob(jobs, pid, FG , cmdline);  //add job
                waitfg(pid);  //wait foreground job

            }
        }
    }

    return;
}
```

```c
int builtin_cmd(char **argv)
{
    //checks whether the input is a built-in command

    char* com = argv[0]; // argv[0] saves the built in instruction
                         // it's the first word typed in the command line

    if(!strcmp(com, "quit")){   //when the command was "quit"

        exit(0); //terminate

    }
    else if (!strcmp(com, "jobs")){  //when the command was "jobs"

        listjobs(jobs); //print job lists

        return 1;   //return 1 means that the user input exists in built-in command

    }
    else if (!strcmp(com, "fg") || !strcmp (com, "bg")){ //when the command was either "fg" or "bg"

        do_bgfg(argv); // execute the builtin bg and fg command

        return 1;

    }
    return 0;    //not included in the built-in command
}
```

```c
void do_bgfg(char **argv)
{
    char* com = argv[0]; //com saves the user input command instruction

    char* id = argv [1]; //id saves the the second input

    struct job_t* job;

    if(id == NULL){ //no second input for id (pid or jid)

        printf("%s command reguires PID or %%jobid argument\n", com); //prints the error message and returns

        return;

    }

    else if (id[0] == '%'){ //when the id starts with "%" -> background job

        int jid; // variable to save the job id

        jid = atoi(&id[1]); // saves the job id address in integer type

        job = getjobjid(jobs, jid);

        if(job == NULL){
            printf("%s: No such job\n", id); // jid not existing
            return;
        }
    }
    else{ // when the input id is pid (process id)

        pid_t pid = atoi(id); //saves the pid

        if(pid == 0){ //when pid == 0, such pid does not exists
            printf("%s: argument must be a PID or %%jobid\n", com); //prints an error message that either pid or jobid must be inputted
            return;
        }
        else{ //when valid pid input received

            job = getjobpid(jobs, pid);

            if (job == NULL)
            {
                printf("(%d): No such process\n", pid);
                return;
            }
        }
    }
```

```c
    if (!strcmp(com, "fg")) { // when the command was foreground (fg)

        job->state = FG; // change the job state into foreground

        kill(-(job->pid), SIGCONT); // send "continue" signal to process which has pid
                                    // kill sys call simply sends a signal

        waitfg(job->pid);

    }else{ // when the command was background (bg)

        job->state = BG; // change the job state into background

        printf("[%d] (%d) %s", job->jid, job->pid, job->cmdline);

        kill(-(job->pid), SIGCONT);  // send "continue" signal to process which has pid

    }

    return;
}
```

```c
/*
 * waitfg - Block until process pid is no longer the foreground process
 */
void waitfg(pid_t pid)
{
    while (fgpid(jobs) == pid) {
        sleep(1); // a delay of one second when the pid is a foreground process
    }
    return;
}
```

```c
void sigchld_handler(int sig)
{
    pid_t pid;
    struct job_t* job;
    int status;

    while ((pid = waitpid(-1, &status, WNOHANG | WUNTRACED)) > 0) { // wait for the child process
                                                                    // returns the pid of child process when the child process is done
                                                                    // otherwise, returns 0
                                                                    // so this while loop is when the child process is a zombie

        job = getjobpid(jobs, pid);

        if (job != NULL) {
            if (WIFSTOPPED(status)) //if process is stopped
            {
                job->state = ST; //change the job state into stopped

                printf("Job [%d] (%d) stopped by signal 20\n", job->jid, job->pid);

            }
            else if (WIFSIGNALED(status)) { //if process is terminated

                printf("Job [%d] (%d) terminated by signal 2\n", job->jid, job->pid);

                deletejob(jobs, pid); // if the process is terminated ot exited, delete the job

            }
            else if (WIFEXITED(status)){

                deletejob(jobs, pid);

            }
        }
    }
    return;
}
```

```c
/*
 * sigint_handler - The kernel sends a SIGINT to the shell whenver the
 *    user types ctrl-c at the keyboard.  Catch it and send it along
 *    to the foreground job.
 */

void sigint_handler(int sig)
{
    pid_t pid = fgpid(jobs); // saves the process id of fg job

    if (pid != 0)
    {
        kill(-pid, sig); //send signal (SIGINT)
    }
    return;
}

/*
 * sigtstp_handler - The kernel sends a SIGTSTP to the shell whenever
 *     the user types ctrl-z at the keyboard. Catch it and suspend the
 *     foreground job by sending it a SIGTSTP.
 */

void sigtstp_handler(int sig)  //suspend each process
{
    pid_t pid = fgpid(jobs); // saves the process id of fg job

    if (pid != 0) //if fg job exists
    {
        kill(-pid, sig); // send signal (SIGTSTP)
    }
    return;
}
```

4. Trace Results

```
[jihyunk@programming2 ShellLab]$ make test01
./sdriver.pl -t trace01.txt -s ./tsh -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
[jihyunk@programming2 ShellLab]$ make rtest01
./sdriver.pl -t trace01.txt -s ./tshref -a "-p"
#
# trace01.txt - Properly terminate on EOF.
#
```

```
[jihyunk@programming2 ShellLab]$ make test02
./sdriver.pl -t trace02.txt -s ./tsh -a "-p"
#
# trace02.txt - Process builtin quit command.
#
[jihyunk@programming2 ShellLab]$ make rtest02
./sdriver.pl -t trace02.txt -s ./tshref -a "-p"
#
# trace02.txt - Process builtin quit command.
#
[jihyunk@programming2 ShellLab]$
```

```
[jihyunk@programming2 ShellLab]$ make test03
./sdriver.pl -t trace03.txt -s ./tsh -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
[jihyunk@programming2 ShellLab]$ make rtest03
./sdriver.pl -t trace03.txt -s ./tshref -a "-p"
#
# trace03.txt - Run a foreground job.
#
tsh> quit
```

```
[jihyunk@programming2 ShellLab]$ make test04
./sdriver.pl -t trace04.txt -s ./tsh -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
(1) (3371) ./myspin 1 &
[jihyunk@programming2 ShellLab]$ make rtest04
./sdriver.pl -t trace04.txt -s ./tshref -a "-p"
#
# trace04.txt - Run a background job.
#
tsh> ./myspin 1 &
[1] (3397) ./myspin 1 &
[jihyunk@programming2 ShellLab]$
```

```
[jihyunk@programming2 ShellLab]$ make test05
./sdriver.pl -t trace05.txt -s ./tsh -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
(1) (3844) ./myspin 2 &
tsh> ./myspin 3 &
(2) (3846) ./myspin 3 &
tsh> jobs
[1] (3844) Running ./myspin 2 &
[2] (3846) Running ./myspin 3 &
[jihyunk@programming2 ShellLab]$ make rtest05
./sdriver.pl -t trace05.txt -s ./tshref -a "-p"
#
# trace05.txt - Process jobs builtin command.
#
tsh> ./myspin 2 &
[1] (3898) ./myspin 2 &
tsh> ./myspin 3 &
[2] (3900) ./myspin 3 &
tsh> jobs
[1] (3898) Running ./myspin 2 &
[2] (3900) Running ./myspin 3 &
```

```
[jihyunk@programming2 ShellLab]$ make test06
./sdriver.pl -t trace06.txt -s ./tsh -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (8733) terminated by signal 2
[jihyunk@programming2 ShellLab]$ make rtest06
./sdriver.pl -t trace06.txt -s ./tshref -a "-p"
#
# trace06.txt - Forward SIGINT to foreground job.
#
tsh> ./myspin 4
Job [1] (8772) terminated by signal 2
```

```
[jihyunk@programming2 ShellLab]$ make test07
./sdriver.pl -t trace07.txt -s ./tsh -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
(1) (8861) ./myspin 4 &
tsh> ./myspin 5
Job [2] (8863) terminated by signal 2
tsh> jobs
[1] (8861) Running ./myspin 4 &
[jihyunk@programming2 ShellLab]$ make rtest07
./sdriver.pl -t trace07.txt -s ./tshref -a "-p"
#
# trace07.txt - Forward SIGINT only to foreground job.
#
tsh> ./myspin 4 &
[1] (8902) ./myspin 4 &
tsh> ./myspin 5
Job [2] (8904) terminated by signal 2
tsh> jobs
[1] (8902) Running ./myspin 4 &
```

```
[jihyunk@programming2 ShellLab]$ make test08
./sdriver.pl -t trace08.txt -s ./tsh -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
(1) (8991) ./myspin 4 &
tsh> ./myspin 5
Job [2] (8993) stopped by signal 20
tsh> jobs
[1] (8991) Running ./myspin 4 &
[2] (8993) Stopped ./myspin 5
[jihyunk@programming2 ShellLab]$ make rtest08
./sdriver.pl -t trace08.txt -s ./tshref -a "-p"
#
# trace08.txt - Forward SIGTSTP only to foreground job.
#
tsh> ./myspin 4 &
[1] (9041) ./myspin 4 &
tsh> ./myspin 5
Job [2] (9044) stopped by signal 20
tsh> jobs
[1] (9041) Running ./myspin 4 &
[2] (9044) Stopped ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ make test09
./sdriver.pl -t trace09.txt -s ./tsh -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
(1) (9137) ./myspin 4 &
tsh> ./myspin 5
Job [2] (9139) stopped by signal 20
tsh> jobs
[1] (9137) Running ./myspin 4 &
[2] (9139) Stopped ./myspin 5
tsh> bg %2
[2] (9139) ./myspin 5
tsh> jobs
[1] (9137) Running ./myspin 4 &
[2] (9139) Running ./myspin 5
[jihyunk@programming2 ShellLab]$ make rtest09
./sdriver.pl -t trace09.txt -s ./tshref -a "-p"
#
# trace09.txt - Process bg builtin command
#
tsh> ./myspin 4 &
[1] (9223) ./myspin 4 &
tsh> ./myspin 5
Job [2] (9225) stopped by signal 20
tsh> jobs
[1] (9223) Running ./myspin 4 &
[2] (9225) Stopped ./myspin 5
tsh> bg %2
[2] (9225) ./myspin 5
tsh> jobs
[1] (9223) Running ./myspin 4 &
[2] (9225) Running ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ make test10
./sdriver.pl -t trace10.txt -s ./tsh -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
(1) (9346) ./myspin 4 &
tsh> fg %1
Job [1] (9346) stopped by signal 20
tsh> jobs
[1] (9346) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
[jihyunk@programming2 ShellLab]$ make rtest10
./sdriver.pl -t trace10.txt -s ./tshref -a "-p"
#
# trace10.txt - Process fg builtin command.
#
tsh> ./myspin 4 &
[1] (9388) ./myspin 4 &
tsh> fg %1
Job [1] (9388) stopped by signal 20
tsh> jobs
[1] (9388) Stopped ./myspin 4 &
tsh> fg %1
tsh> jobs
```

```
[jihyunk@programming2 ShellLab]$ 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] (9587) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  499 pts/29   S      0:00 ./tsh -p
  503 pts/29   T      0:00 ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ make rtest11
./sdriver.pl -t trace11.txt -s ./tshref -a "-p"
#
# trace11.txt - Forward SIGINT to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (9665) terminated by signal 2
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  499 pts/29   S      0:00 ./tsh -p
  503 pts/29   T      0:00 ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ 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] (9801) stopped by signal 20
tsh> jobs
[1] (9801) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  499 pts/29   S      0:00 ./tsh -p
  503 pts/29   T      0:00 ./myspin 5
  840 pts/29   S      0:00 ./tsh -p
  844 pts/29   T      0:00 ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ make rtest12
./sdriver.pl -t trace12.txt -s ./tshref -a "-p"
#
# trace12.txt - Forward SIGTSTP to every process in foreground process group
#
tsh> ./mysplit 4
Job [1] (10002) stopped by signal 20
tsh> jobs
[1] (10002) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  499 pts/29   S      0:00 ./tsh -p
  503 pts/29   T      0:00 ./myspin 5
  840 pts/29   S      0:00 ./tsh -p
  844 pts/29   T      0:00 ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ 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] (10085) stopped by signal 20
tsh> jobs
[1] (10085) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  499 pts/29   S      0:00 ./tsh -p
  503 pts/29   T      0:00 ./myspin 5
  840 pts/29   S      0:00 ./tsh -p
  844 pts/29   T      0:00 ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ make rtest13
./sdriver.pl -t trace13.txt -s ./tshref -a "-p"
#
# trace13.txt - Restart every stopped process in process group
#
tsh> ./mysplit 4
Job [1] (10203) stopped by signal 20
tsh> jobs
[1] (10203) Stopped ./mysplit 4
tsh> /bin/ps a
  PID TTY      STAT   TIME COMMAND
  499 pts/29   S      0:00 ./tsh -p
  503 pts/29   T      0:00 ./myspin 5
  840 pts/29   S      0:00 ./tsh -p
  844 pts/29   T      0:00 ./myspin 5
```

```
[jihyunk@programming2 ShellLab]$ 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) (10596) ./myspin 4 &
tsh> fg
fg command reguires PID or %jobid argument
tsh> bg
bg command reguires 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] (10596) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (10596) ./myspin 4 &
tsh> jobs
[1] (10596) Running ./myspin 4 &
```

```
make[jihyunk@programming2 ShellLab]$ make rtest14
./sdriver.pl -t trace14.txt -s ./tshref -a "-p"
#
# trace14.txt - Simple error handling
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 4 &
[1] (10650) ./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] (10650) stopped by signal 20
tsh> bg %2
%2: No such job
tsh> bg %1
[1] (10650) ./myspin 4 &
tsh> jobs
[1] (10650) Running ./myspin 4 &
```

```
[jihyunk@programming2 ShellLab]$ 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] (10809) terminated by signal 2
tsh> ./myspin 3 &
(1) (10823) ./myspin 3 &
tsh> ./myspin 4 &
(2) (10825) ./myspin 4 &
tsh> jobs
[1] (10823) Running ./myspin 3 &
[2] (10825) Running ./myspin 4 &
tsh> fg %1
Job [1] (10823) stopped by signal 20
tsh> jobs
[1] (10823) Stopped ./myspin 3 &
[2] (10825) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (10823) ./myspin 3 &
tsh> jobs
[1] (10823) Running ./myspin 3 &
[2] (10825) Running ./myspin 4 &
tsh> fg %1
tsh> quit
```

```
[jihyunk@programming2 ShellLab]$ make rtest15
./sdriver.pl -t trace15.txt -s ./tshref -a "-p"
#
# trace15.txt - Putting it all together
#
tsh> ./bogus
./bogus: Command not found
tsh> ./myspin 10
Job [1] (10902) terminated by signal 2
tsh> ./myspin 3 &
[1] (10916) ./myspin 3 &
tsh> ./myspin 4 &
[2] (10918) ./myspin 4 &
tsh> jobs
[1] (10916) Running ./myspin 3 &
[2] (10918) Running ./myspin 4 &
tsh> fg %1
Job [1] (10916) stopped by signal 20
tsh> jobs
[1] (10916) Stopped ./myspin 3 &
[2] (10918) Running ./myspin 4 &
tsh> bg %3
%3: No such job
tsh> bg %1
[1] (10916) ./myspin 3 &
tsh> jobs
[1] (10916) Running ./myspin 3 &
[2] (10918) Running ./myspin 4 &
tsh> fg %1
tsh> quit
[jihyunk@programming2 ShellLab]$
```

```
[jihyunk@programming2 ShellLab]$ 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] (11059) stopped by signal 20
tsh> jobs
[1] (11059) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (11087) terminated by signal 2
[jihyunk@programming2 ShellLab]$ 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] (11142) stopped by signal 20
tsh> jobs
[1] (11142) Stopped ./mystop 2
tsh> ./myint 2
Job [2] (11179) terminated by signal 2
```