SUSTech CS302 OS Lab2 Report

Title: Process

Student: Name: Lu Ning, ID: 11610310

Time: 2019.3.12

Experimental Environment: linux

Deadline: **23:59, 2019-03-13**

Summit by: Blackboard

Task：

Task 1. Use the man command to get the manual of fork, exec, wait, exit, pipe

Task 2. Compile and execute fork.c, observe the results and the process

Task 3．Compile and execute pipe.c, observe the results

Task 4．Use the man command to get the manual of sigaction, tcsetpgrp, setpgid

Task 5．Compile and execute signal.c, observe the results and the process

Task 6．Compile and execute process.c, observe the results and the process

Experiments:

1. fundamental：

- What is a system call: The system call is the fundamental interface between an application and the Linux kernel. System calls are generally not invoked directly, but rather via wrapper functions in glibc (or perhaps some other library).
- What is fork:

  fork() creates a new process by duplicating the calling process. The new process is referred to as the child process. The calling process is referred to as the parent process.
- How to realize inter-process communication:

  Use pipe() to create a pipe, a unidirectional data channel that can be used for interprocess communication.

- How to realize inter-process connection: Use the pipe()

2、 Write the prototype of the following functions:

fork: pid_t fork(void);

signal: typedef void (*sighandler_t)(int);

sighandler_t signal(int signum, sighandler_t handler);

pipe: int pipe(int pipefd[2]);

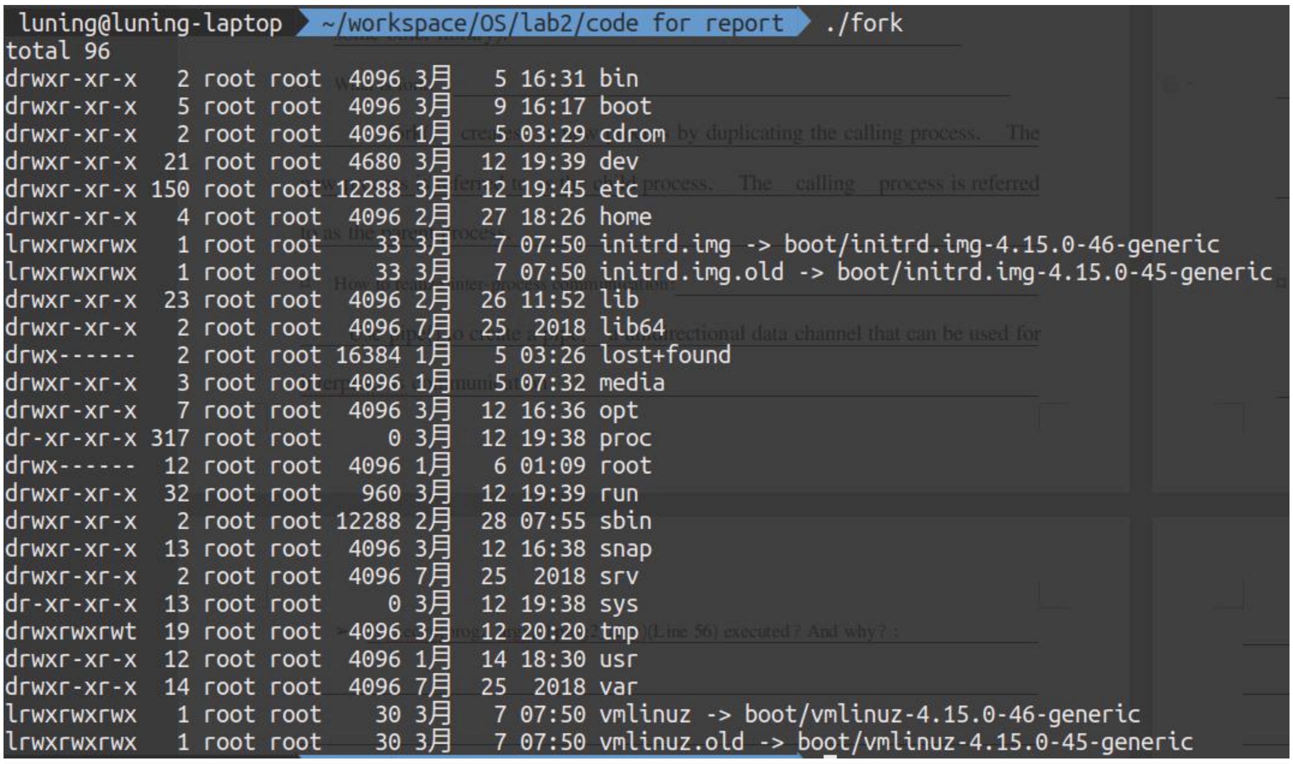
The array pipefd is used to return two file descriptors referring to the ends of the pipe. pipefd[0] refers to the read end of the pipe. pipefd[1] refers to the write end of the pipe. Data written to the write end of the pipe is buffered by the kernel until it is read from the read end of the pipe.

tcsetpgrp: int tcsetpgrp(int fd, pid_t pgrp);

3、 Execute and observe

- **fork.c**

  ➢ Result:

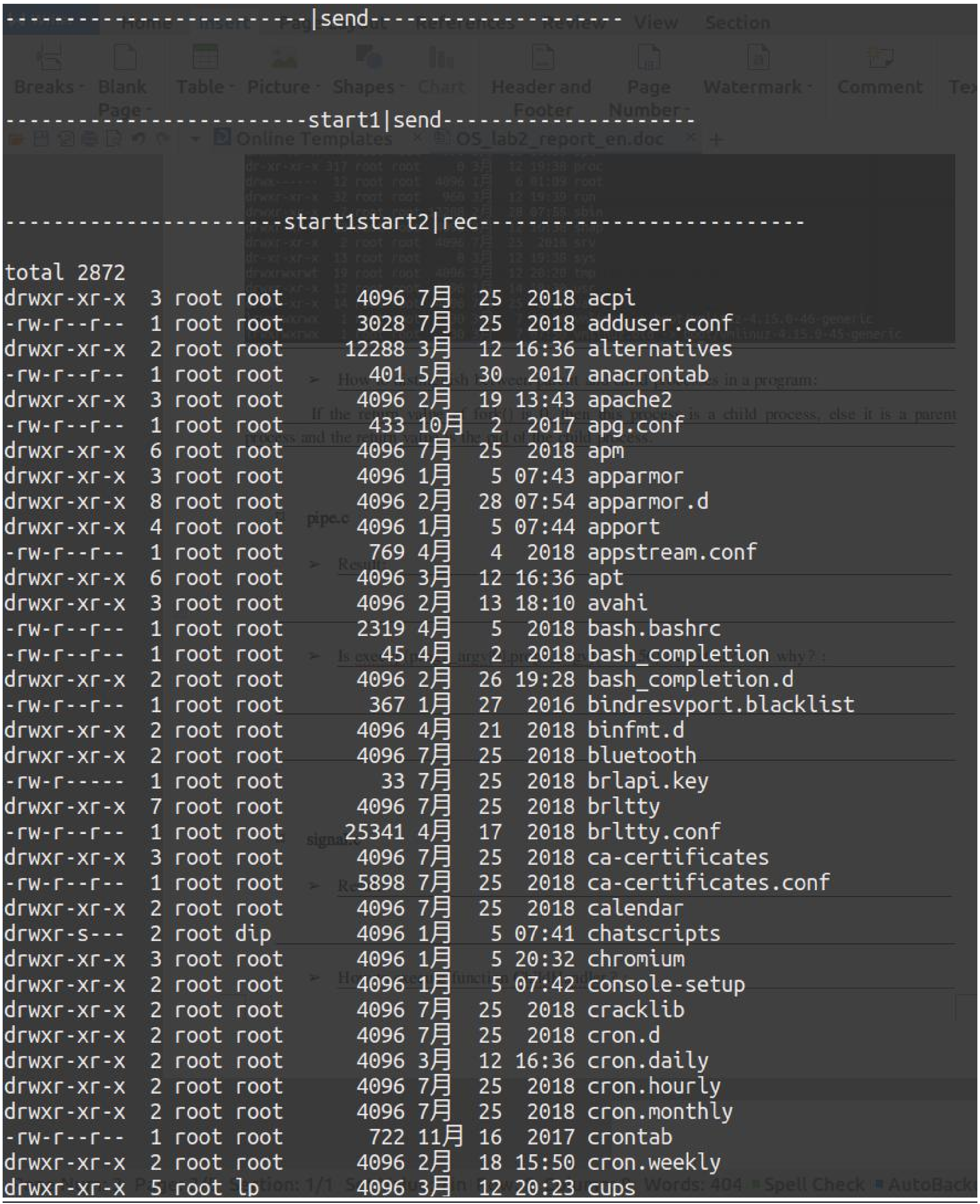

```
luning@luning-laptop  ~/workspace/OS/lab2/code for report  ./fork
total 96
drwxr-xr-x   2 root root  4096 3月   5 16:31 bin
drwxr-xr-x   5 root root  4096 3月   9 16:17 boot
drwxr-xr-x   2 root root  4096 1月   5 03:29 cdrom
drwxr-xr-x  21 root root  4680 3月  12 19:39 dev
drwxr-xr-x 150 root root 12288 3月  12 19:45 etc
drwxr-xr-x   4 root root  4096 2月  27 18:26 home
lrwxrwxrwx   1 root root    33 3月   7 07:50 initrd.img -> boot/initrd.img-4.15.0-46-generic
lrwxrwxrwx   1 root root    33 3月   7 07:50 initrd.img.old -> boot/initrd.img-4.15.0-45-generic
drwxr-xr-x  23 root root  4096 2月  26 11:52 lib
drwxr-xr-x   2 root root  4096 7月  25  2018 lib64
drwx------   2 root root 16384 1月   5 03:26 lost+found
drwxr-xr-x   3 root root  4096 1月   5 07:32 media
drwxr-xr-x   7 root root  4096 3月  12 16:36 opt
dr-xr-xr-x 317 root root     0 3月  12 19:38 proc
drwx------  12 root root  4096 1月   6 01:09 root
drwxr-xr-x  32 root root   960 3月  12 19:39 run
drwxr-xr-x   2 root root 12288 2月  28 07:55 sbin
drwxr-xr-x  13 root root  4096 3月  12 16:38 snap
drwxr-xr-x   2 root root  4096 7月  25  2018 srv
dr-xr-xr-x  13 root root     0 3月  12 19:38 sys
drwxrwxrwt  19 root root  4096 3月  12 20:20 tmp
drwxr-xr-x  12 root root  4096 1月  14 18:30 usr
drwxr-xr-x  14 root root  4096 7月  25  2018 var
lrwxrwxrwx   1 root root    30 3月   7 07:50 vmlinuz -> boot/vmlinuz-4.15.0-46-generic
lrwxrwxrwx   1 root root    30 3月   7 07:50 vmlinuz.old -> boot/vmlinuz-4.15.0-45-generic
```

➢ How to distinguish between parent and child processes in a program:

If the return value of fork() is 0, then this process is a child process, else it is a parent process and the return value is the pid of the child process.

⌑ **pipe.c**

➢ Result:



```
-------------------------|send---------------------

-------------------------start1|send--------------------



----------------------start1start2|rec---------------------------

total 2872
drwxr-xr-x  3 root root      4096 7月  25  2018 acpi
-rw-r--r--  1 root root      3028 7月  25  2018 adduser.conf
drwxr-xr-x  2 root root     12288 3月  12 16:36 alternatives
-rw-r--r--  1 root root       401 5月  30  2017 anacrontab
drwxr-xr-x  3 root root      4096 2月  19 13:43 apache2
-rw-r--r--  1 root root       433 10月  2  2017 apg.conf
drwxr-xr-x  6 root root      4096 7月  25  2018 apm
drwxr-xr-x  3 root root      4096 1月   5 07:43 apparmor
drwxr-xr-x  8 root root      4096 2月  28 07:54 apparmor.d
drwxr-xr-x  4 root root      4096 1月   5 07:44 apport
-rw-r--r--  1 root root       769 4月   4  2018 appstream.conf
drwxr-xr-x  6 root root      4096 3月  12 16:36 apt
drwxr-xr-x  3 root root      4096 2月  13 18:10 avahi
-rw-r--r--  1 root root      2319 4月   5  2018 bash.bashrc
-rw-r--r--  1 root root        45 4月   2  2018 bash_completion
drwxr-xr-x  2 root root      4096 2月  26 19:28 bash_completion.d
-rw-r--r--  1 root root       367 1月  27  2016 bindresvport.blacklist
drwxr-xr-x  2 root root      4096 4月  21  2018 binfmt.d
drwxr-xr-x  2 root root      4096 7月  25  2018 bluetooth
-rw-r-----  1 root root        33 7月  25  2018 brlapi.key
drwxr-xr-x  7 root root      4096 7月  25  2018 brltty
-rw-r--r--  1 root root     25341 4月  17  2018 brltty.conf
drwxr-xr-x  3 root root      4096 7月  25  2018 ca-certificates
-rw-r--r--  1 root root      5898 7月  25  2018 ca-certificates.conf
drwxr-xr-x  2 root root      4096 7月  25  2018 calendar
drwxr-s---  2 root dip       4096 1月   5 07:41 chatscripts
drwxr-xr-x  3 root root      4096 1月   5 20:32 chromium
drwxr-xr-x  2 root root      4096 1月   5 07:42 console-setup
drwxr-xr-x  2 root root      4096 7月  25  2018 cracklib
drwxr-xr-x  2 root root      4096 7月  25  2018 cron.d
drwxr-xr-x  2 root root      4096 3月  12 16:36 cron.daily
drwxr-xr-x  2 root root      4096 7月  25  2018 cron.hourly
drwxr-xr-x  2 root root      4096 7月  25  2018 cron.monthly
-rw-r--r--  1 root root       722 11月 16  2017 crontab
drwxr-xr-x  2 root root      4096 2月  18 15:50 cron.weekly
drwxr-xr-x  5 root lp        4096 3月  12 20:23 cups
```

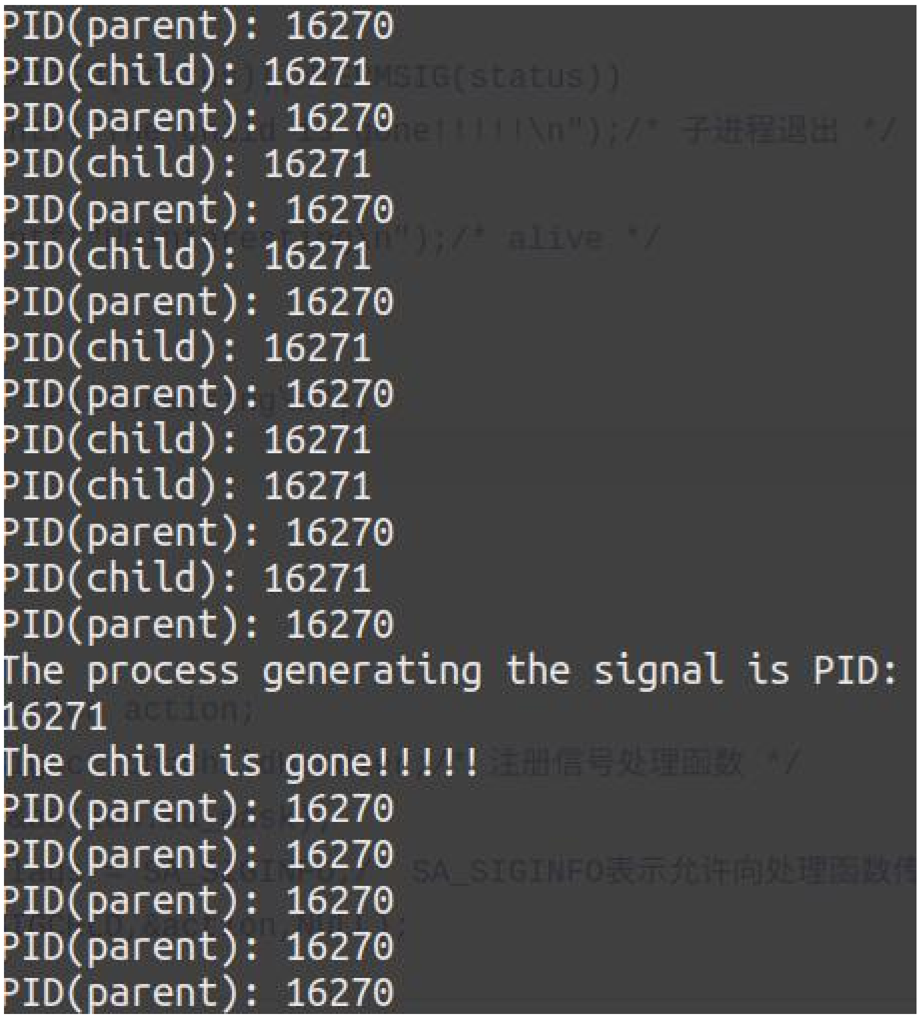
➢ Is execvp(prog2_argv[0],prog2_argv)(Line 56) executed？And why？:

Yes. First, the first child process wait until the parent process write to the pipe. Then dup2(pipe_fd[1],1) let the stdout file descriptor change to write end of pipe. So the output of the ‘ls -l /etc/’ is written into pipe file. dup2(pipe_fd[0],0) let the pipe read end to be stdin stream file descriptor 0. So the input of command ‘more’ is from the pipe file, which is the output of command ‘ls’. So we can see the result in terminal.

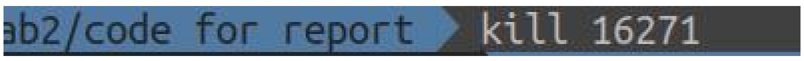
It acts like ‘ls -l /etc/ | more’

- **signal.c**

  ➢ Result:



```
PID(parent): 16270
PID(child): 16271
PID(parent): 16270
PID(child): 16271
PID(parent): 16270
PID(child): 16271
PID(parent): 16270
PID(child): 16271
PID(parent): 16270
PID(child): 16271
PID(child): 16271
PID(parent): 16270
PID(child): 16271
PID(parent): 16270
The process generating the signal is PID:
16271
The child is gone!!!!!
PID(parent): 16270
PID(parent): 16270
PID(parent): 16270
PID(parent): 16270
PID(parent): 16270
```



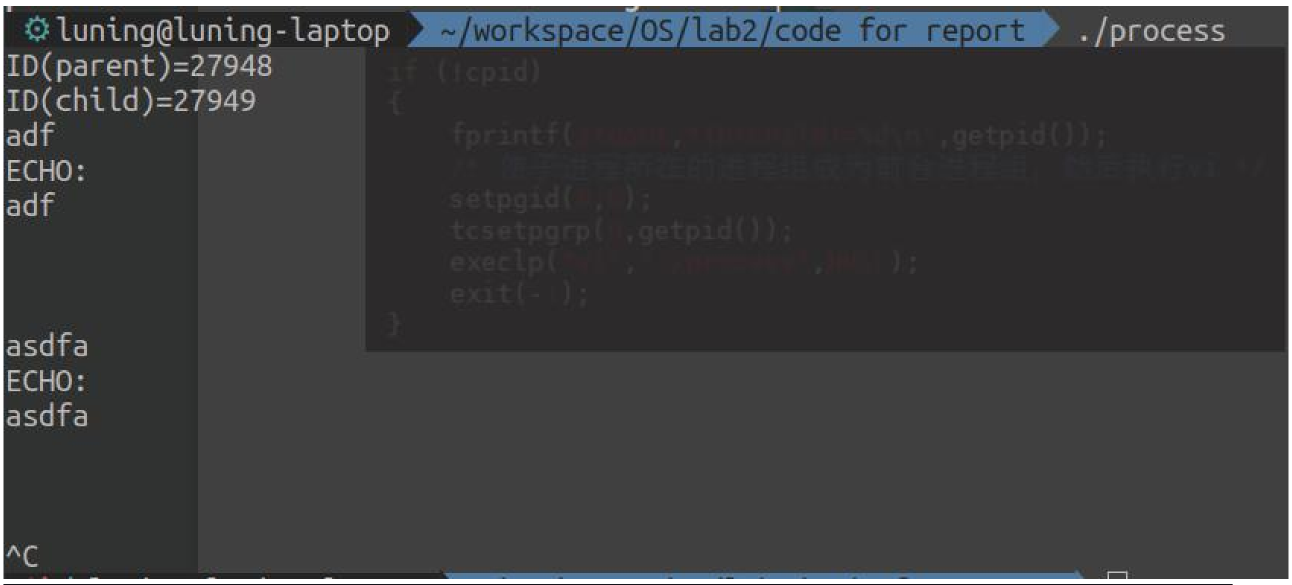
```
ab2/code for report  kill 16271
```

  ➢ How to execute function ChildHandler ? :

Because the sigaction process signal SIGCHLD, which is sent by a child process when it exits. So I kill the child process and the parent process receive SIGCHLD and executes the signal handler.

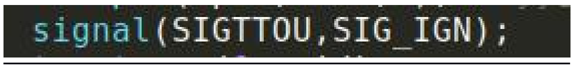
- **process.c**

  ➢ Result:

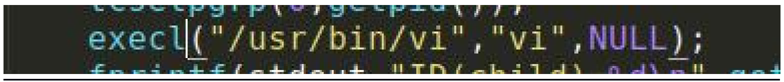


```
luning@luning-laptop  ~/workspace/OS/lab2/code for report  ./process
ID(parent)=27948
ID(child)=27949
adf
ECHO:
adf


asdfa
ECHO:
asdfa



^C
```

- DEBUG



```
signal(SIGTTOU,SIG_IGN);
```



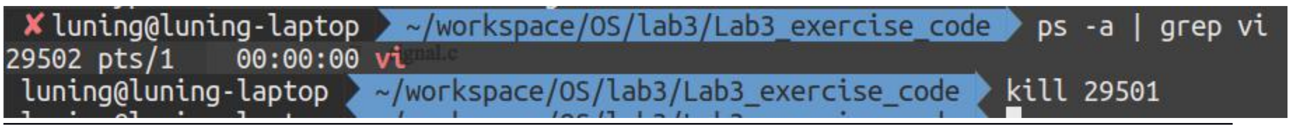
```
execl("/usr/bin/vi","vi",NULL);
```

- How many *./process* in the process list ? And what's the difference between them ?:

There are two processes, the parent and the child process. One is a foreground process and

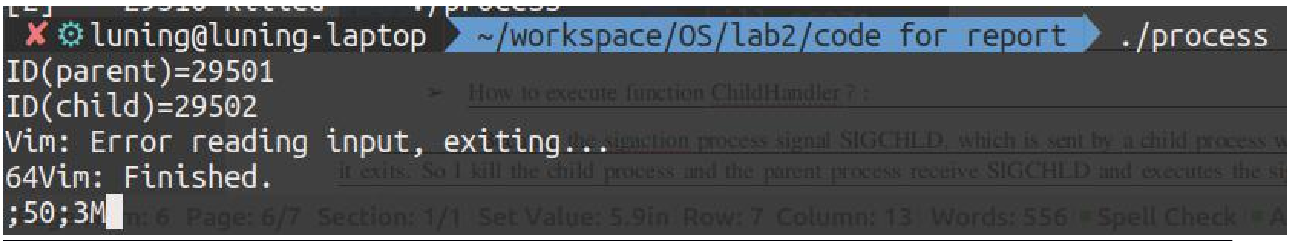
- What happens after killing the main process:

FIND AND KILL:



```
luning@luning-laptop  ~/workspace/OS/lab3/Lab3_exercise_code  ps -a | grep vi
29502 pts/1    00:00:00 vi
luning@luning-laptop  ~/workspace/OS/lab3/Lab3_exercise_code  kill 29501
```

RESULT:



```
luning@luning-laptop  ~/workspace/OS/lab2/code for report  ./process
ID(parent)=29501
ID(child)=29502
Vim: Error reading input, exiting...
64Vim: Finished.
;50;3M
```

The two processes is killed

Conclusion:

I know the concepts of process, process group, foreground and background. And learn a lot of functions about them like getpg(), tcsetpgrp().

As for the debug part, the first bug is the directory fault. And the second is about tcsetpgrp() function. The manufacture file says that **If tcsetpgrp() is called by a member of a background process group in its session, and the calling process is not blocking or ignoring SIGTTOU, a SIGTTOU signal is sent to all members of this background process group.** So ignore the SIGTTOU and the parent process group will be the foreground group.

Submission:

-lab2_report_studentID.pdf **(pdf version report)**

Suggestion and Feedback for Lab：