Programming Assignment #1 – A Simple Shell

Introduction to Operating Systems

Prof. Li-Pin Chang

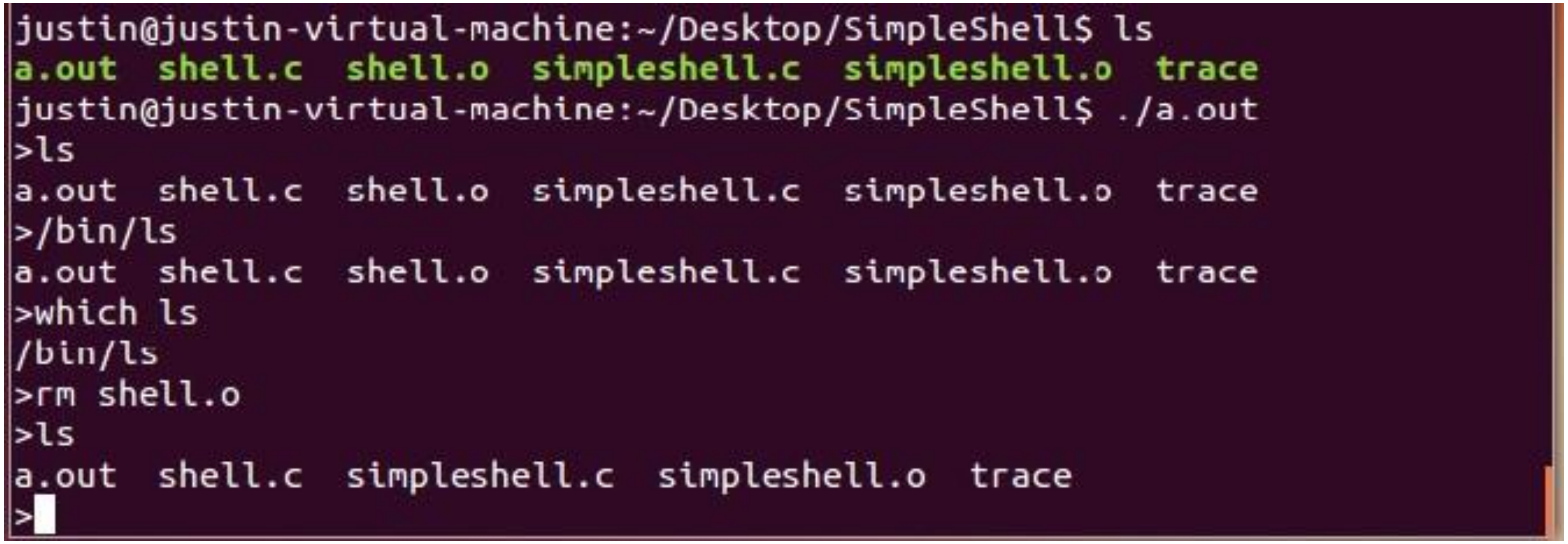
CS@NYCU

A Simple Shell

Control flow of your simple shell:

1. Display the prompt sign “>” and take a string from user
2. Parse the string into a program name and arguments
3. Fork a child process
4. Have the child process execute the program
5. Wait until the child terminates
6. Go to the first step

Example Output

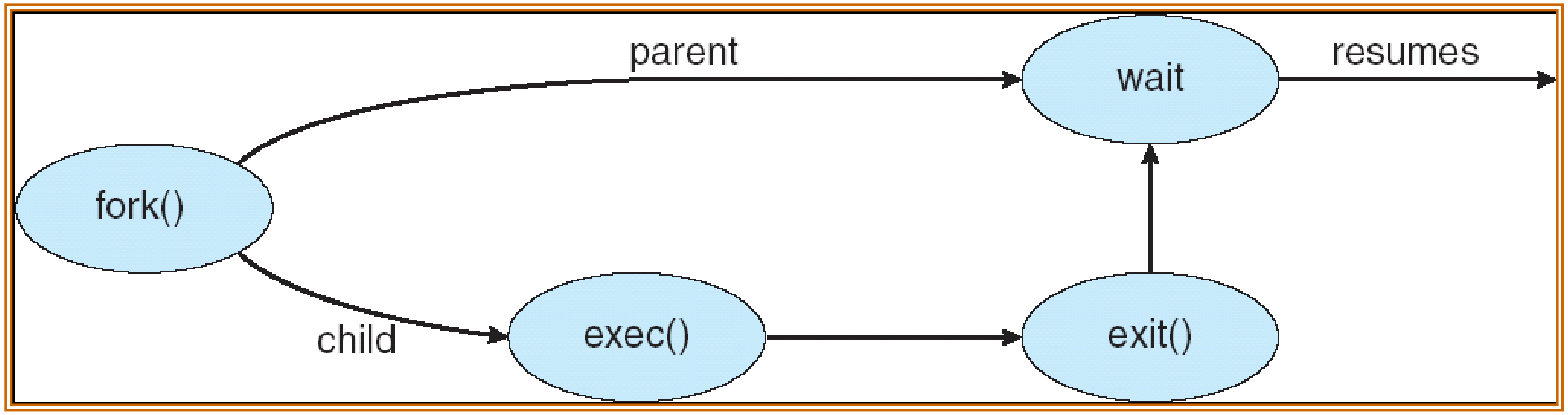


```
justin@justin-virtual-machine:~/Desktop/SimpleShell$ ls
a.out  shell.c  shell.o  simpleshell.c  simpleshell.o  trace
justin@justin-virtual-machine:~/Desktop/SimpleShell$ ./a.out
>ls
a.out  shell.c  shell.o  simpleshell.c  simpleshell.o  trace
>/bin/ls
a.out  shell.c  shell.o  simpleshell.c  simpleshell.o  trace
>which ls
/bin/ls
>rm shell.o
>ls
a.out  shell.c  simpleshell.c  simpleshell.o  trace
>
```

Core Logic of Your Shell

```
pid_t  pid;


 …
/* fork another process */
pid = fork();
if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
}
else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
}
else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
}
…
```



Tip: You don’t have to implement “ls”, “cp”, etc in your shell;
they are programs in GNU coreutils.
Your shell just forks a child to execute them.

Important System Calls

- fork()
  - Create a child process
  - http://man7.org/linux/man-pages/man2/fork.2.html
- exec() family
  - Have the current process execute the program specified in the pathname
  - http://man7.org/linux/man-pages/man3/exec.3.html
- wait() family
  - wait() wait the termination of anyone of the child processes
  - waitpid() waits the termination of the specified child process
  - http://man7.org/linux/man-pages/man2/waitpid.2.html

Waiting on Child Processes

- If a command is ended with “&”, then the shell will not wait on a child process
- For example:
  - sleep 10s → The prompt re-appears after 10 seconds
  - sleep 10s & → The prompt re-appears immediately
- A child process becomes a zombie if it is not waited by its parent process. Get rid of zombies using
  - signal(SIGCHLD) or
  - double fork, etc.

Bonus

- I/O redirection (+5 pts)
  - `ls –l > a.txt`
  - `echo “zzz” > b.txt`
  - `more < b.txt`
- Pipe (+5 pts)
  - `ls –l | more`

- Related APIs: pipe(), dup2()

Input Commands

- Your shell must correctly handle test cases of the following format
  - `<program> <arg1> <arg2> <…>`
- Test cases will be like the following:
  - `clear`
  - `ls -l`
  - `cp a.txt b.txt`
  - `cat c.txt &`
- No multiple pipes and redirections; no appending “>>”
  - (x) a | b | c
  - (x) a < b > c
  - (x) a | b > c

Header of your .c or .cpp

```
/*
Student No.: 31415926
Student Name: John Doe
Email: xxx@yyy.zzz
SE tag: xnxcxtxuxoxsx
Statement: I am fully aware that this program is not
supposed to be posted to a public server, such as a
public GitHub repository or a public web page.
*/
```

Caution

You receive a score penalty for

- Use system() → get 0 point
- Use popen() → no bonus for pipe
- Zombie processes exist before or after your shell terminates
- The header is absent from your source program

Testing OS Environment

- Ubuntu 22.04
- Install as a VM or on a physical machine