1. Run some commands that successfully run and run others that do not work. That is, try commands that crash or prematurely terminate as well as commands that complete their execution. What happens to your shell? Does your shell crash when a command crash?

When running commands that does not exist such as “ls -all”, which does not exist on linux, the shell reports that

```
Sh: 1: ls -all: not found
```

However the shell does not crash and continue to execute, for example, for a input file, I put in some genuine shell command, and some made-up ones. The log file is as followed:

```
Typed a command: ls
input.txt  tshell  tshell.c

Typed a command: sadd
sh: 1: sadd: not found

Typed a command: mkdir test

Typed a command: rm -r test
Typed a command:
No more command, exiting
```

2. Determine the system calls that are used in implementing the system() library function

```
strace ./tshell < input.txt
….
rt_sigaction(SIGINT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7fc3aea5df20}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_IGN, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7fc3aea5df20}, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=0}, 8) = 0
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
clone(child_stack=NULL, flags=CLONE_PARENT_SETTID|SIGCHLD, parent_tidptr=0x7ffdd12dd82c) =
13120
wait4(13120,
….
[{WIFEXITED(s) && WEXITSTATUS(s) == 0}], 0, NULL) = 13120
rt_sigaction(SIGINT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7fc3aea5df20}, NULL, 8) = 0
rt_sigaction(SIGQUIT, {sa_handler=SIG_DFL, sa_mask=[], sa_flags=SA_RESTORER,
sa_restorer=0x7fc3aea5df20}, NULL, 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=13120, si_uid=1000, si_status=0,
si_utime=0, si_stime=0} ---
write(1, "Typed a command: ls -la\n", 24Typed a command: ls -la
) = 24
read(0, "", 4096)                   = 0                          = ?
```

```
+++ exited with 0 +++
```

The system() is implemented by cloning a child process and use the following line to execute the command in the child process

```
execl("/bin/sh", "sh", "-c", command, (char *) NULL);
```

3. Implementation of fork()
In my implementation of system() using fork() and waitpid(), the output is exactly the same as the system() function, this is because waitpid() is used to make sure that the parent process, which is the process that reads the shell commands will wait for the child process, which is the process that executes the shell command, to finish.

```
Typed a command: ls -la
total 44
drwxr-xr-x 2 jamestang97 jamestang97 4096 Oct  6 21:30 .
drwxr-xr-x 3 jamestang97 jamestang97 4096 Oct  6 17:08 ..
-rw-r--r-- 1 jamestang97 jamestang97   22 Oct  6 21:30 input.txt
-rwxr-xr-x 1 jamestang97 jamestang97 8520 Oct  6 20:16 tshell
-rwxr-xr-x 1 jamestang97 jamestang97  417 Oct  6 21:00 tshell.c
-rwxr-xr-x 1 jamestang97 jamestang97 8784 Oct  6 21:30 tshell_vf
-rw-rw-r-- 1 jamestang97 jamestang97  938 Oct  6 21:29 tshell_vf.c

Typed a command: echo "hello"
hello

input.txt  tshell  tshell.c  tshell_vf    tshell_vf.c
Typed a command: ls
Typed a command:
No more command, exiting
```

4. Measuring between versions of system() for executing 1000 lister commands

Version	Time in Seconds
System default: system()	4.7877 s
Fork version: systemf()	4.8756 s
vFork version: systemvf()	4.6739 s
Clone version: systemc()	3.0317 s

5. Improve performance with FLAG

In my case, since the command being executed is lister command, which accesses the file system, I used CLONE_FS as my flag, this improved the performance quite a bit. CLONE_FS allows the parent and child to share the same file system information. This is useful for cases where a command in child affects the parent.

```
Int pid = clone(child_func,stackTop,CLONE_FS,cmd)
```