操作系统 lab5

实验要求

1. 实现分支：ch5。

2. 实现进程控制，可以运行 usershell。

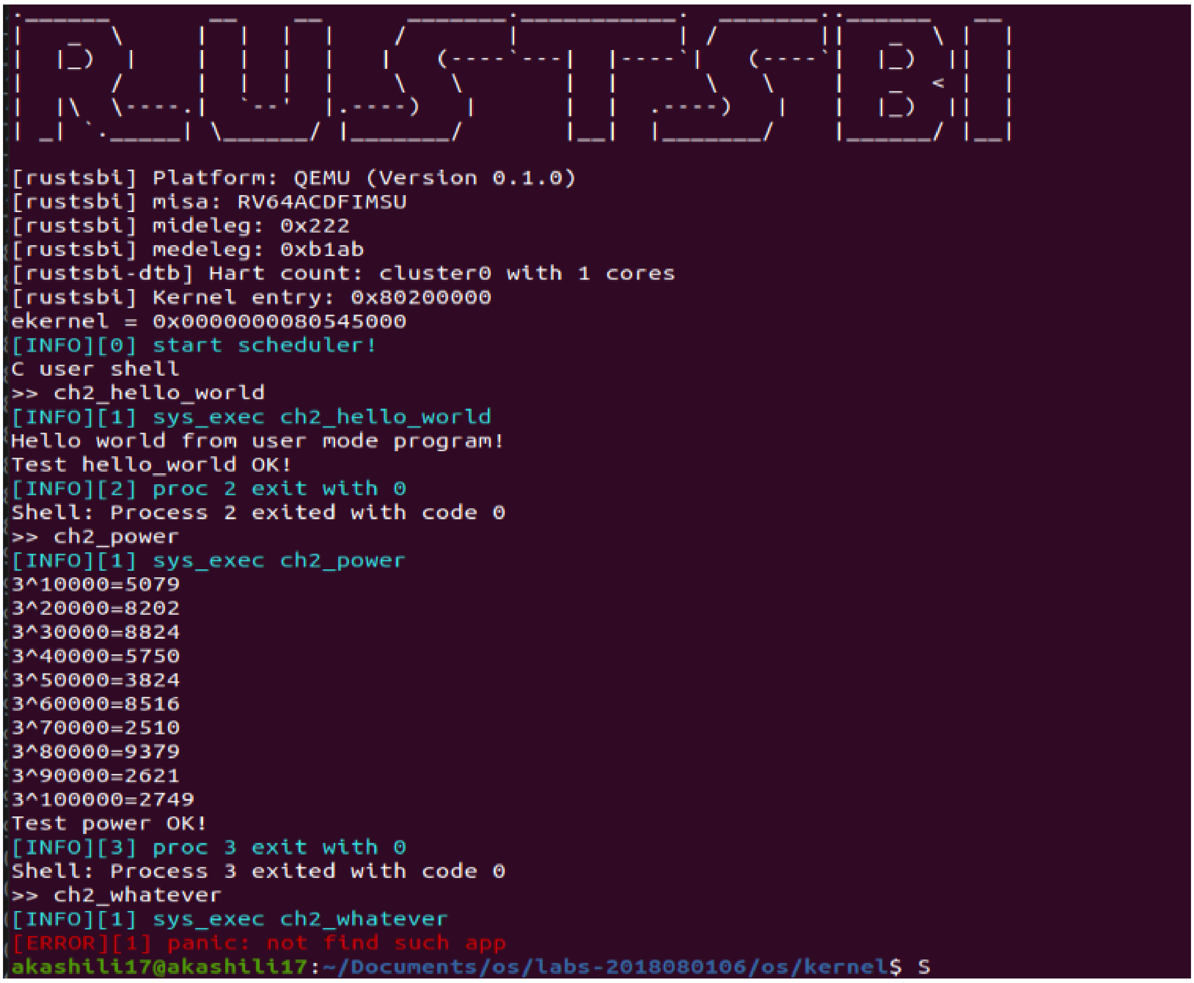
3. 实现自定义系统调用 spawn，并通过 C测例中chapter5对应的所有测例。

实验结果

In this experiment, we had to implement a more flexible process control system. In order to realize such a system, we had to implement 4 new system call functions, including

- sys_read: read bytes from the input (where we input the app name)
- sys_fork: create process that is the same as the current process
- sys_exec: modify the current process to execute the specified program from scratch.
- sys_wait: wait for one or any of the subprocesses to end and get its exit_ code

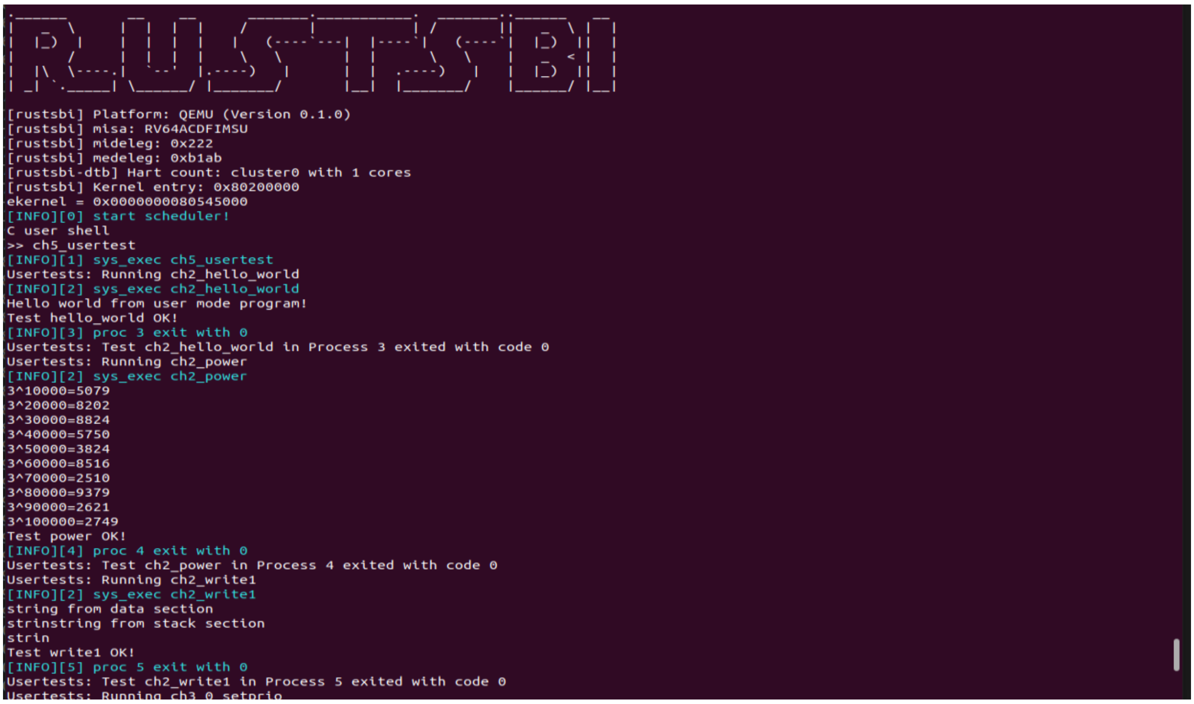
After implementing these function located in `syscall.c` , and `make all CHAPTER=5` in /user, and `make run` in /kernel.



```
.______       __    __      _______.___________.  _______..______   __
|   _  \     |  |  |  |    /       |           | /       ||   _  \ |  |
|  |_)  |    |  |  |  |   |   (----`---|  |----`|   (----`|  |_)  ||  |
|      /     |  |  |  |    \   \       |  |      \   \    |   _  < |  |
|  |\  \----.|  `--'  |.----)   |      |  |  .----)   |   |  |_)  ||  |
| _| `._____| \______/ |_______/       |__|  |_______/    |______/ |__|

[rustsbi] Platform: QEMU (Version 0.1.0)
[rustsbi] misa: RV64ACDFIMSU
[rustsbi] mideleg: 0x222
[rustsbi] medeleg: 0xb1ab
[rustsbi-dtb] Hart count: cluster0 with 1 cores
[rustsbi] Kernel entry: 0x80200000
ekernel = 0x0000000080545000
[INFO][0] start scheduler!
C user shell
>> ch2_hello_world
[INFO][1] sys_exec ch2_hello_world
Hello world from user mode program!
Test hello_world OK!
[INFO][2] proc 2 exit with 0
Shell: Process 2 exited with code 0
>> ch2_power
[INFO][1] sys_exec ch2_power
3^10000=5079
3^20000=8202
3^30000=8824
3^40000=5750
3^50000=3824
3^60000=8516
3^70000=2510
3^80000=9379
3^90000=2621
3^100000=2749
Test power OK!
[INFO][3] proc 3 exit with 0
Shell: Process 3 exited with code 0
>> ch2_whatever
[INFO][1] sys_exec ch2_whatever
[ERROR][1] panic: not find such app
akashili17@akashili17:~/Documents/os/labs-2018080106/os/kernel$ S
```

If we run one of the chapter 5 tests, such as `ch5_usertest.bin` , we get



```
.______       __    __      _______.___________.  _______..______   __
|   _  \     |  |  |  |    /       |           | /       ||   _  \ |  |
|  |_)  |    |  |  |  |   |   (----`---|  |----`|   (----`|  |_)  ||  |
|      /     |  |  |  |    \   \       |  |      \   \    |   _  < |  |
|  |\  \----.|  `--'  |.----)   |      |  |  .----)   |   |  |_)  ||  |
| _| `._____| \______/ |_______/       |__|  |_______/    |______/ |__|

[rustsbi] Platform: QEMU (Version 0.1.0)
[rustsbi] misa: RV64ACDFIMSU
[rustsbi] mideleg: 0x222
[rustsbi] medeleg: 0xb1ab
[rustsbi-dtb] Hart count: cluster0 with 1 cores
[rustsbi] Kernel entry: 0x80200000
ekernel = 0x0000000080545000
[INFO][0] start scheduler!
C user shell
>> ch5_usertest
[INFO][1] sys_exec ch5_usertest
Usertests: Running ch2_hello_world
[INFO][2] sys_exec ch2_hello_world
Hello world from user mode program!
Test hello_world OK!
[INFO][3] proc 3 exit with 0
Usertests: Test ch2_hello_world in Process 3 exited with code 0
Usertests: Running ch2_power
[INFO][2] sys_exec ch2_power
3^10000=5079
3^20000=8202
3^30000=8824
3^40000=5750
3^50000=3824
3^60000=8516
3^70000=2510
3^80000=9379
3^90000=2621
3^100000=2749
Test power OK!
[INFO][4] proc 4 exit with 0
Usertests: Test ch2_power in Process 4 exited with code 0
Usertests: Running ch2_write1
[INFO][2] sys_exec ch2_write1
string from data section
strinstring from stack section
strin
Test write1 OK!
[INFO][5] proc 5 exit with 0
Usertests: Test ch2_write1 in Process 5 exited with code 0
Usertests: Running ch3_0_setprio
```

问答作业

1. fork + exec 的一个比较大的问题是 fork 之后的内存页/文件等资源完全没有使用就废弃了，针对这一点，有什么改进策略？

   【解答】

   On one hand, we can utilize `spawn` instead of `fork` + `exec`

   On the other hand we can adopt `copy on write (COW)`. Meaning, only necessary data can be copied when `fork` occurs, and resources such as memory pages/files can be copied when the behavior ofof changing the corresponding segment occurs in the parent and child processes.

2. 其实使用了题1的策略之后，fork + exec 所带来的无效资源的问题已经基本被解决了，但是近年来 fork 还是在被不断的批判，那么到底是什么正在"杀死"fork？可以参考论文，**注意**：回答无明显错误就给满分，出这题只是想引发大家的思考，完全不要求看论文，球球了，别卷了。

   【解答】

   - if `fork` uses `COW` technology, extracurricular resources are needed to monitor and process the COW process
   - `fork` may use security risks

3. fork 当年被设计并称道肯定是有其好处的。请使用**带初始参数**的 spawn 重写如下 fork 程序，然后描述 fork 有那些好处。注意:使用"伪代码"传达意思即可，spawn接口可以自定义。可以写多个文件。

   ```
   int main() {
       int a = get_a();
       if(fork() == 0) {
           int b = get_b();
           printf("a + b = %d", a + b);
           exit(0);
       }
       printf("a = %d", a);
       return 0;
   }
   ```

```
// child_program
int main() {
    let b = get_b();
    printf("a + b = %d", a + b);
    return 0;
}
```

4. 描述进程执行的几种状态，以及 fork/exec/wait/exit 对与状态的影响。

```
fork : child process is set as READY, parent is set as RUNNING
exec : set as RUNNING
wait : set as READY
exit : set as ZOMBIE
```