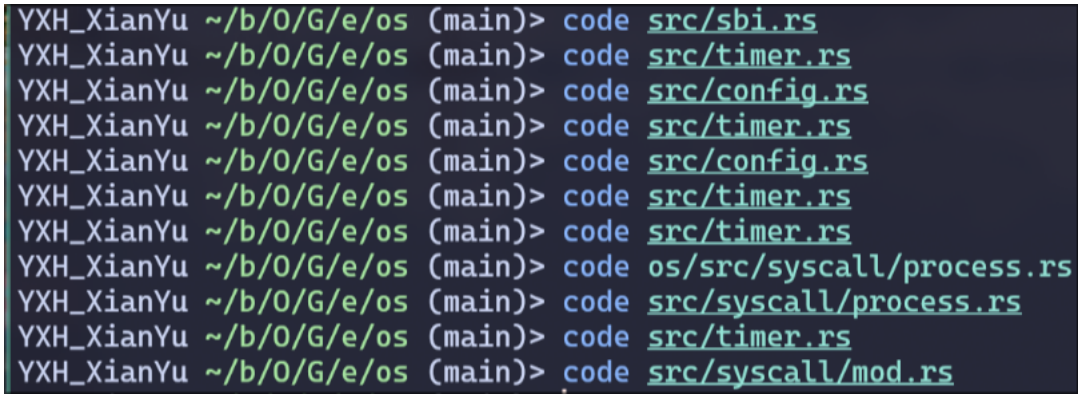
操作系统 实验5

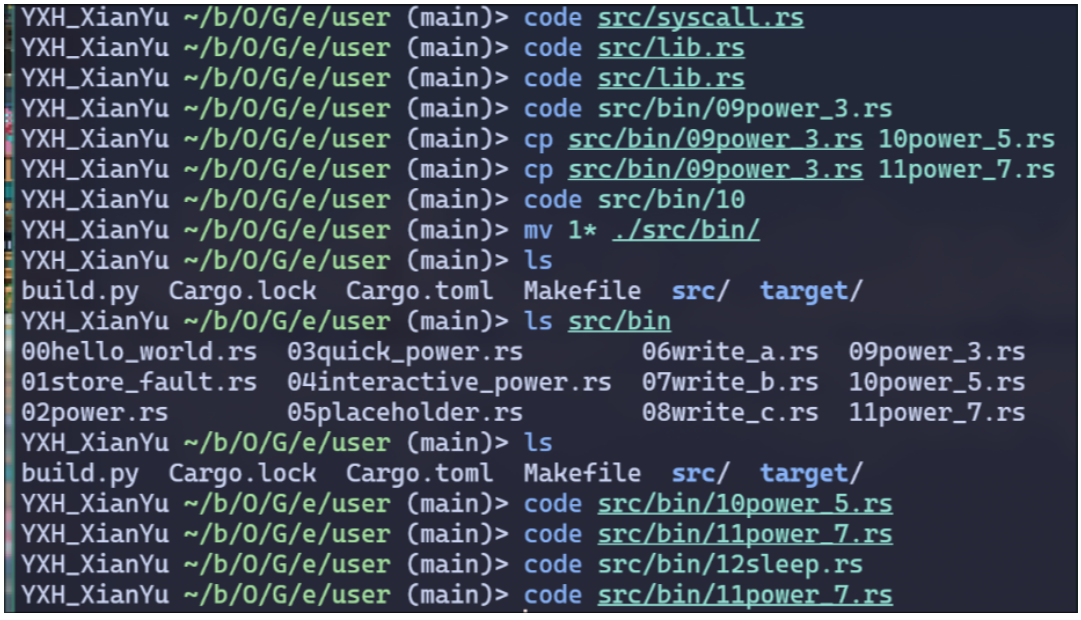
> 21301114 俞贤皓
>
> 环境（实验0~2, 4~5）： Arch Linux 6.5.3-arch1-1
>
> 环境（实验3~4）： Ubuntu 22.04.3 LTS (WSL)

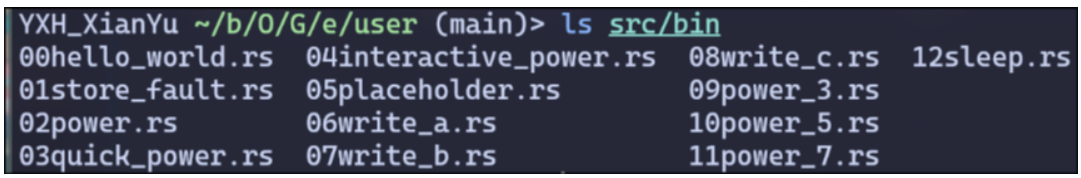
1. 实验步骤

1.1 时钟中断与计时器

- 根据文档编写代码
  - 

1.2 修改应用程序

- 根据文档编写代码
  - 

- 应用程序列表
  - 

1.3 抢占式调度

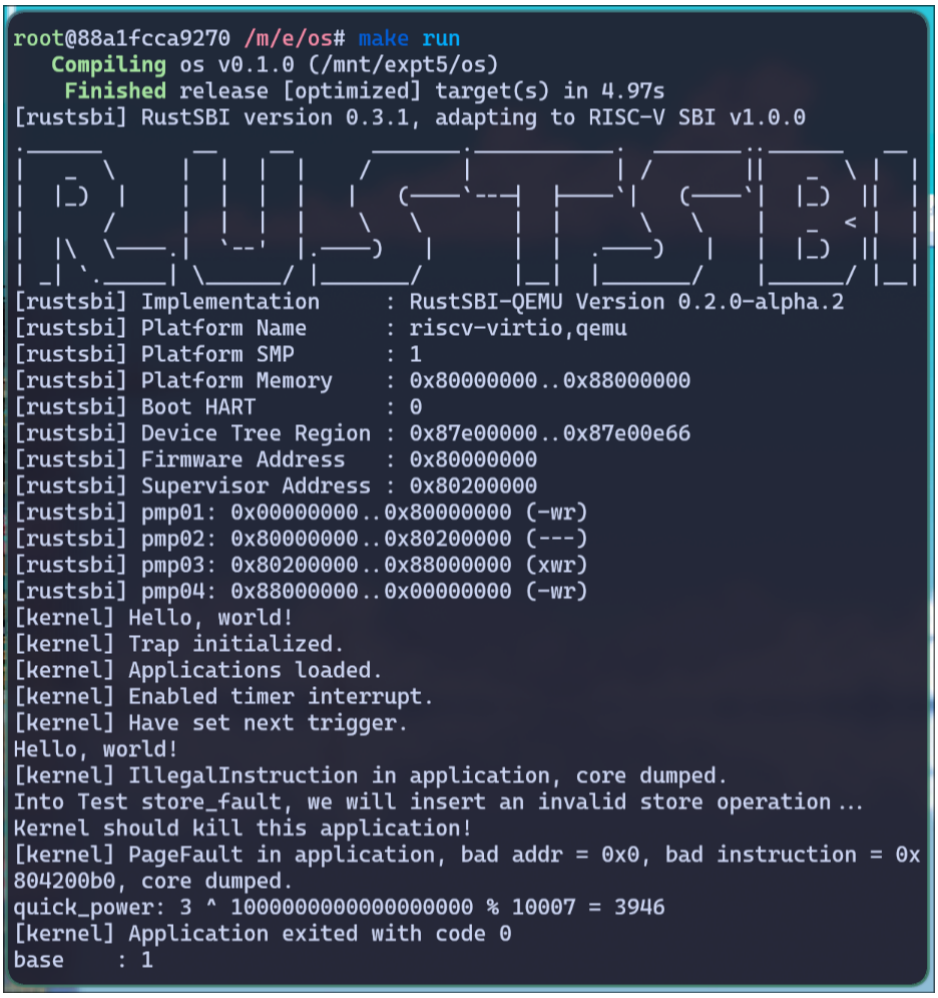
- 根据文档编写代码
  -
```
YXH_XianYu ~/b/O/G/e/user (main)> cd ../os
YXH_XianYu ~/b/O/G/e/os (main)> code src/trap/mod.rs
YXH_XianYu ~/b/O/G/e/os (main)> code src/timer.rs
YXH_XianYu ~/b/O/G/e/os (main)> code src/timer.rs
YXH_XianYu ~/b/O/G/e/os (main)> code src/sbi.rs
YXH_XianYu ~/b/O/G/e/os (main)> code src/main.rs
YXH_XianYu ~/b/O/G/e/os (main)> code src/timer.rs
```

1.4 执行结果

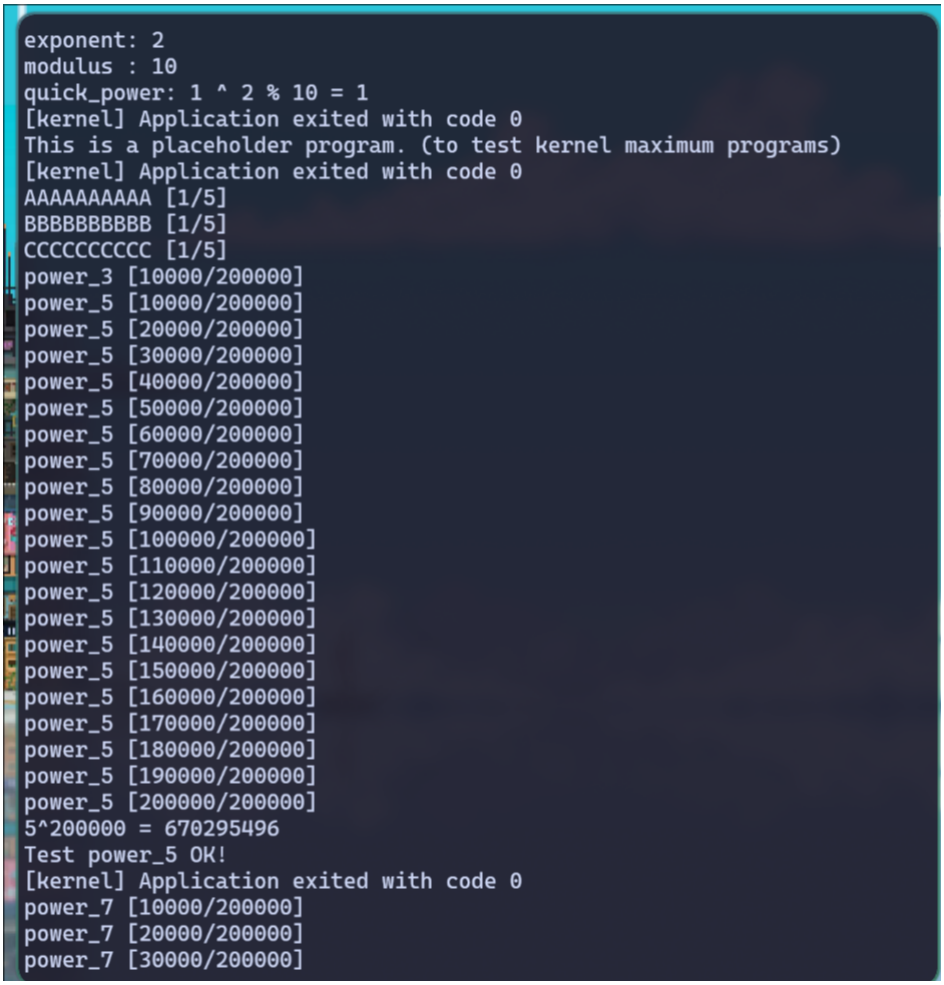
- 编译应用程序
  -
```
root@88a1fcca9270 /m/e/user [2]# make build
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.37s
[build.py] application 00hello_world start with address 0x80400000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.18s
[build.py] application 01store_fault start with address 0x80420000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.20s
[build.py] application 02power start with address 0x80440000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.19s
[build.py] application 03quick_power start with address 0x80460000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
warning: unused import: `crate::user_lib::console::read_u64`
 --> src/bin/04interactive_power.rs:5:5
  |
5 | use crate::user_lib::console::read_u64;
  |     ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  |
  = note: `#[warn(unused_imports)]` on by default

warning: `user_lib` (bin "04interactive_power") generated 1 warning
    Finished release [optimized] target(s) in 0.18s
[build.py] application 04interactive_power start with address 0x8048000
0
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.19s
[build.py] application 05placeholder start with address 0x804a0000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.21s
[build.py] application 06write_a start with address 0x804c0000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
    Finished release [optimized] target(s) in 0.21s
[build.py] application 07write_b start with address 0x804e0000
   Compiling user_lib v0.1.0 (/mnt/expt5/user)
```

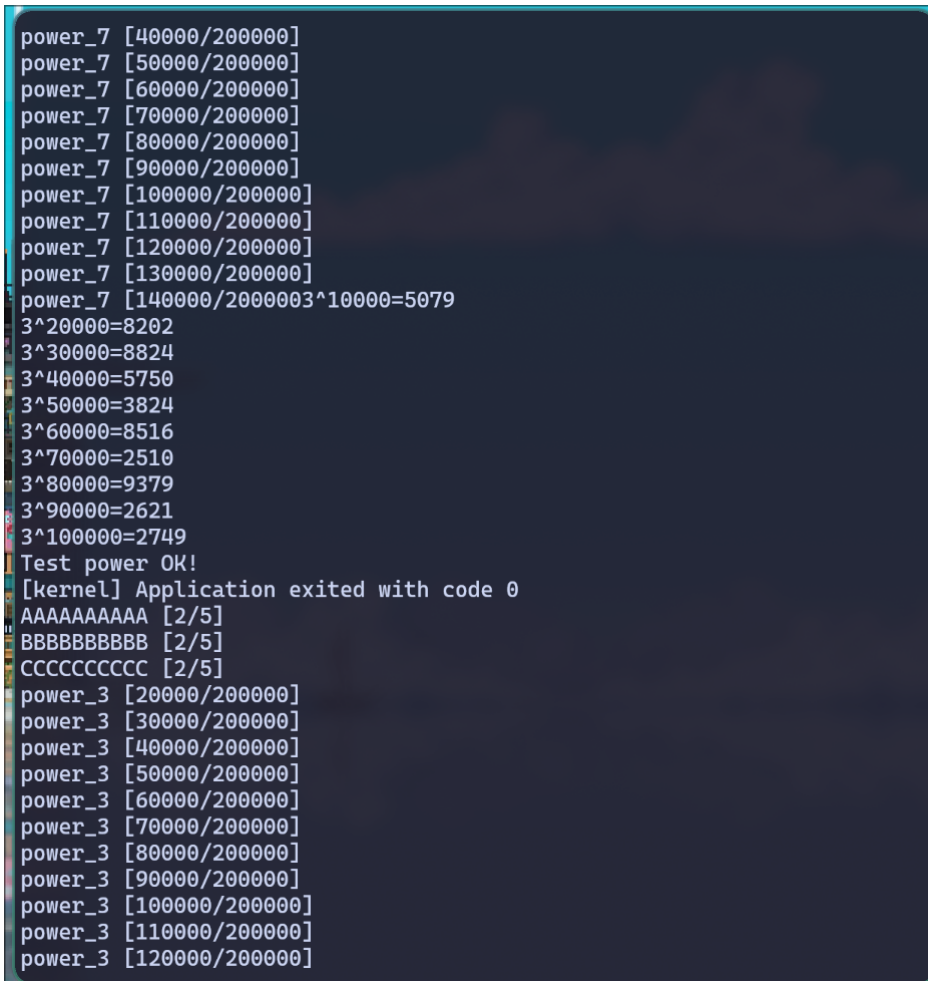
- 执行结果（分五页）

- 

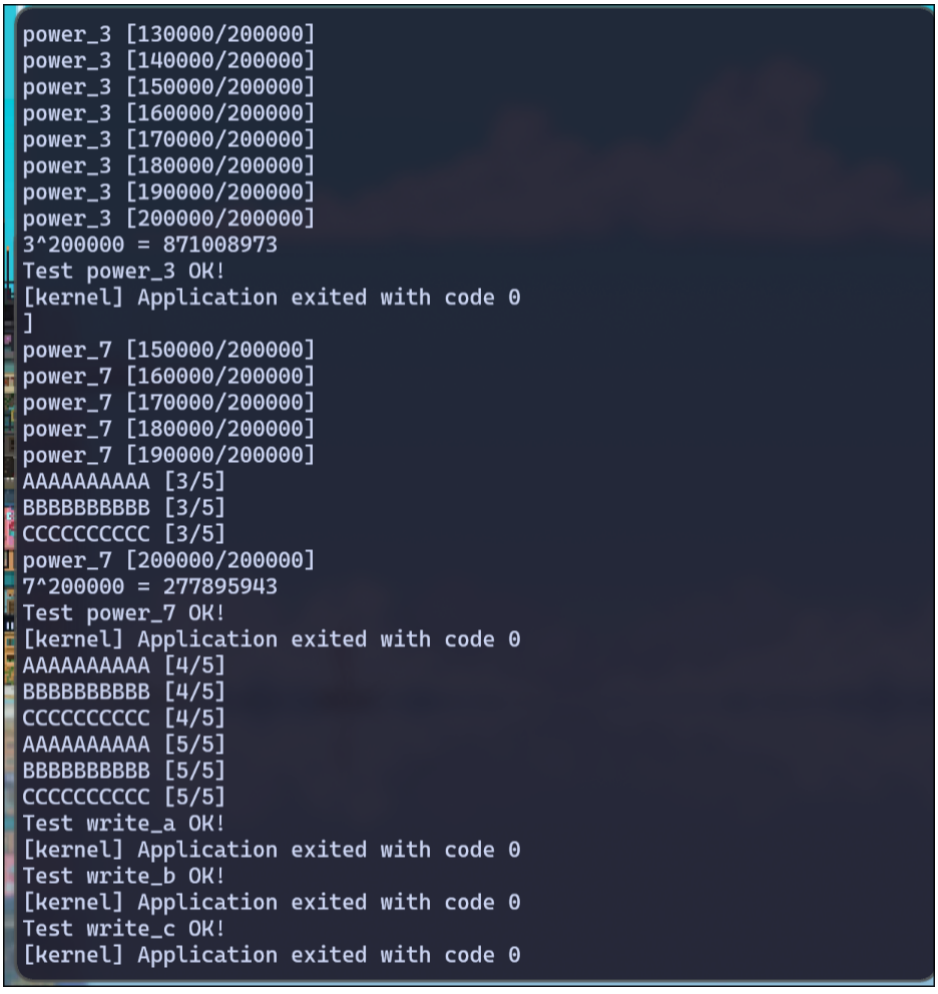
```
root@88a1fcca9270 /m/e/os# make run
   Compiling os v0.1.0 (/mnt/expt5/os)
    Finished release [optimized] target(s) in 4.97s
[rustsbi] RustSBI version 0.3.1, adapting to RISC-V SBI v1.0.0
.______       __    __      _______.___________.  _______..______   __
|   _  \     |  |  |  |    /       |           | /       ||   _  \ |  |
|  |_)  |    |  |  |  |   |   (----`---|  |----`|   (----`|  |_)  ||  |
|      /     |  |  |  |    \   \       |  |      \   \    |   _  < |  |
|  |\  \----.|  `--'  |.----)   |      |  |  .----)   |   |  |_)  ||  |
| _| `._____| \______/ |_______/       |__|  |_______/    |______/ |__|
[rustsbi] Implementation     : RustSBI-QEMU Version 0.2.0-alpha.2
[rustsbi] Platform Name      : riscv-virtio,qemu
[rustsbi] Platform SMP       : 1
[rustsbi] Platform Memory    : 0x80000000..0x88000000
[rustsbi] Boot HART          : 0
[rustsbi] Device Tree Region : 0x87e00000..0x87e00e66
[rustsbi] Firmware Address   : 0x80000000
[rustsbi] Supervisor Address : 0x80200000
[rustsbi] pmp01: 0x00000000..0x80000000 (-wr)
[rustsbi] pmp02: 0x80000000..0x80200000 (---)
[rustsbi] pmp03: 0x80200000..0x88000000 (xwr)
[rustsbi] pmp04: 0x88000000..0x00000000 (-wr)
[kernel] Hello, world!
[kernel] Trap initialized.
[kernel] Applications loaded.
[kernel] Enabled timer interrupt.
[kernel] Have set next trigger.
Hello, world!
[kernel] IllegalInstruction in application, core dumped.
Into Test store_fault, we will insert an invalid store operation...
Kernel should kill this application!
[kernel] PageFault in application, bad addr = 0x0, bad instruction = 0x
804200b0, core dumped.
quick_power: 3 ^ 1000000000000000000 % 10007 = 3946
[kernel] Application exited with code 0
base    : 1
```

- 

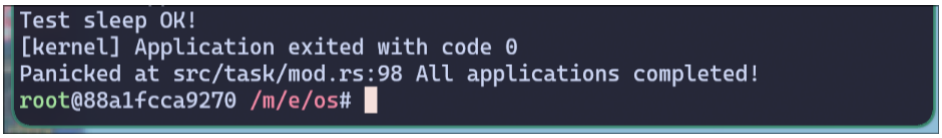
```
exponent: 2
modulus : 10
quick_power: 1 ^ 2 % 10 = 1
[kernel] Application exited with code 0
This is a placeholder program. (to test kernel maximum programs)
[kernel] Application exited with code 0
AAAAAAAAAA [1/5]
BBBBBBBBBB [1/5]
CCCCCCCCCC [1/5]
power_3 [10000/200000]
power_5 [10000/200000]
power_5 [20000/200000]
power_5 [30000/200000]
power_5 [40000/200000]
power_5 [50000/200000]
power_5 [60000/200000]
power_5 [70000/200000]
power_5 [80000/200000]
power_5 [90000/200000]
power_5 [100000/200000]
power_5 [110000/200000]
power_5 [120000/200000]
power_5 [130000/200000]
power_5 [140000/200000]
power_5 [150000/200000]
power_5 [160000/200000]
power_5 [170000/200000]
power_5 [180000/200000]
power_5 [190000/200000]
power_5 [200000/200000]
5^200000 = 670295496
Test power_5 OK!
[kernel] Application exited with code 0
power_7 [10000/200000]
power_7 [20000/200000]
power_7 [30000/200000]
```

- 

```
power_7 [40000/200000]
power_7 [50000/200000]
power_7 [60000/200000]
power_7 [70000/200000]
power_7 [80000/200000]
power_7 [90000/200000]
power_7 [100000/200000]
power_7 [110000/200000]
power_7 [120000/200000]
power_7 [130000/200000]
power_7 [140000/2000003^10000=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!
[kernel] Application exited with code 0
AAAAAAAAAA [2/5]
BBBBBBBBBB [2/5]
CCCCCCCCCC [2/5]
power_3 [20000/200000]
power_3 [30000/200000]
power_3 [40000/200000]
power_3 [50000/200000]
power_3 [60000/200000]
power_3 [70000/200000]
power_3 [80000/200000]
power_3 [90000/200000]
power_3 [100000/200000]
power_3 [110000/200000]
power_3 [120000/200000]
```

- 

```
power_3 [130000/200000]
power_3 [140000/200000]
power_3 [150000/200000]
power_3 [160000/200000]
power_3 [170000/200000]
power_3 [180000/200000]
power_3 [190000/200000]
power_3 [200000/200000]
3^200000 = 871008973
Test power_3 OK!
[kernel] Application exited with code 0
]
power_7 [150000/200000]
power_7 [160000/200000]
power_7 [170000/200000]
power_7 [180000/200000]
power_7 [190000/200000]
AAAAAAAAAA [3/5]
BBBBBBBBBB [3/5]
CCCCCCCCCC [3/5]
power_7 [200000/200000]
7^200000 = 277895943
Test power_7 OK!
[kernel] Application exited with code 0
AAAAAAAAAA [4/5]
BBBBBBBBBB [4/5]
CCCCCCCCCC [4/5]
AAAAAAAAAA [5/5]
BBBBBBBBBB [5/5]
CCCCCCCCCC [5/5]
Test write_a OK!
[kernel] Application exited with code 0
Test write_b OK!
[kernel] Application exited with code 0
Test write_c OK!
[kernel] Application exited with code 0
```

- 

- 可以看到，执行结果乱序，比如power_3执行到[10000/200000]就切换为了power_5，power_5执行到一半又切换回了在实验3中实现的计算3次幂程序。这足以说明，除了主动让出进程所有权以外，操作系统在抢占式地，以分时的方式，给每个进程分配资源！
- 太酷了！

2. 思考问题

2.1 分析分时多任务是如何实现的

- 首先，我们需要在riscv架构中设置一个计时器，设置cpu执行多少秒后触发一次中断。
  - `os/src/timer.rs`
- 接着，我们需要使用riscv的sie::set_stimer()，来使得操作系统会定时对当前的进程产生一次中断。
  - `os/src/trap/mod.rs`
- 接着，我们要对中断异常进行处理。中断异常和其他异常有区别，不应该退出当前程序，所以我们**不调用** `exit_current_and_run_next()`，而是调用 `suspend_current_and_run_next()`。
  - `os/src/trap/mod.rs`
- 最后，在操作系统入口处启用 `sie::set_stimer()` 的中断，并且设置第一次中断的时间间隔，然后就可以实现分时多任务。
  - `os/src/main.rs`

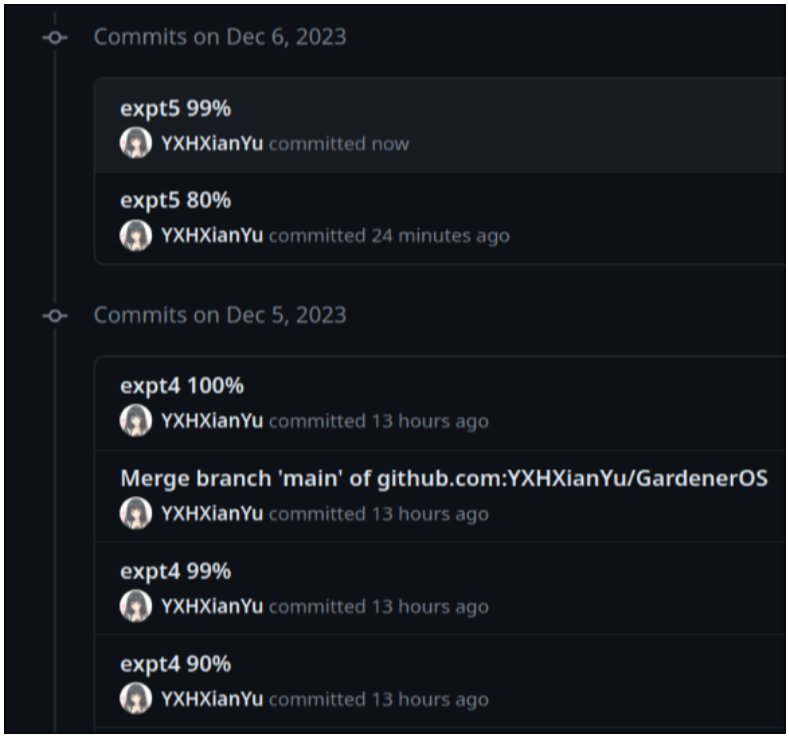
2.2 分析抢占式调度是如何设计和实现的

- 抢占式调度，我们沿用上次实验实现的任务上下文切换的机制，通过将当前任务的上下文储存起来，并读取下一个任务的上下文，来实现任务切换。而这个任务切换（挂起），是由操作系统发出的，并不是程序主动发出的，所以是抢占式调度。
- 核心函数为实验4实现的 `suspend_current_and_run_next()`，该函数分为两步
  - 第一步，挂起当前Task（将Task的状态设置为：Ready）
  - 第二步，执行下一个Task。
- 除此之外，我们是用定时器来定时中断当前任务，执行下一个任务。

2.3 对比协作式调度与抢占式调度

- 本操作系统中，协作式调度和抢占式调度的实现方法非常类似。
- 联系
  - 这两种调度方法均基于Task，中断或yield时，将当前Task的上下文进行储存，并切换至下一个Task。
- 区别
  - 协作式调度，为主动让出CPU资源，在代码中主动使用yield来触发。
  - 抢占式调度，为被动让出CPU资源，在操作系统中，通过计时器导致的中断来触发。
  - 抢占式调度，需要额外实现一个计时器。（也是本次实验的主要代码量）

3. Git提交截图

- 仓库链接
- 

4. 其他说明

4.1 关于rustsbi-qemu新版的问题

- 我在实验2中，使用了0.2.0-alpha.2版本的rustsbi-qemu完成了实验，所以今天的实验我也均用最新版的rustsbi-qemu完成。
- 根据文档实现代码，并不做任何修改，在运行后，会产生 **卡死** 的现象。
  - 经过分析，代码在执行到 `task::run_first_task()` 语句时，程序卡死。
- 因为代码与文档一致，所以考虑rustsbi-qemu的问题。故查阅 riscv-sbi文档，发现旧的set_timer api已经被废弃。
  - 证据1：EID 0x00 已经被废弃，应该使用 0x54494D45
    - ### 5.10. Function Listing

      Table 5. Legacy Function List

Function Name	SBI Version	FID	EID	Replacement EID
sbi_set_timer	0.1	0	0x00	0x54494D45
sbi_console_putchar	0.1	0	0x01	N/A
  - 证据2：专门介绍新API的文档
    - ### Chapter 6. Timer Extension (EID #0x54494D45 "TIME")

      This replaces legacy timer extension (EID #0x00). It follows the new calling convention defined in v0.2.
- 根据riscv-sbi文档对代码进行修改
  - 在 `os/src/sbi.rs` 中添加以下代码，并替换旧代码：

- ```
  const SBI_EID_SET_TIMER: usize = 0x54494D45;

  #[inline(always)]
  fn sbi_call(eid: usize, fid: usize, arg0: usize, arg1: usize, arg2:
  usize) -> usize {
      let mut ret;
      unsafe {
          asm!("ecall",
               in("x10") arg0,
               in("x11") arg1,
               in("x12") arg2,
               in("x16") fid,
               in("x17") eid,
               lateout("x10") ret
          );
      }
      ret
  }

  pub fn set_timer(timer: usize) {
      sbi_call(SBI_EID_SET_TIMER, 0, timer, 0, 0);
  }
  ```
  - 请注意，**其他系统调用也要同步修改**。但我在实验2中已经修改了对应代码，这里就不深入描述。
- 于是，我们就得到了正确的输出结果～(∠・ω<)⌒☆
  - 如本实验手册1.4节所示。
  ```