实验一: rcore代码注释和测试实验报告

1. 重现rcore的交叉编译和在qemu上的x86-64和riscv32的运行步骤重现, 依据重现过程中遇到的问题，对相关文档进行完善

按照rcore的相关文档, 在x86_64和riscv32运行rcore, 经过一段时间的调试后都成功运行, 运行截图如下

```
BdsDxe: loading Boot0001 "UEFI QEMU HARDDISK QM00001 " from PciRoot(0x0)/Pci(0
x1,0x1)/Ata(Primary,Master,0x0)
BdsDxe: starting Boot0001 "UEFI QEMU HARDDISK QM00001 " from PciRoot(0x0)/Pci(
0x1,0x1)/Ata(Primary,Master,0x0)
INFO: bootloader is running
INFO: opening file: \EFI\Boot\rboot.conf
INFO: loading file to memory
INFO: switching graphic mode
INFO: config: Config {
INFO:     physical_memory_offset: 0xffff800000000000,
INFO:     kernel_path: "\\EFI\\rCore\\kernel.elf",
INFO:     resolution: Some(
INFO:         (
INFO:             0x400,
INFO:             0x300,
INFO:         ),
INFO:     ),
INFO: }
INFO: acpi2: 0xbfbfa014
INFO: opening file: \EFI\rCore\kernel.elf
INFO: loading file to memory
INFO: mapping ELF
INFO: mapping physical memory
INFO: starting application processors
Hello world! from CPU 1!
Hello world! from CPU 2!
Hello world! from CPU 3!
INFO: exit boot services
Hello world! from CPU 0!
```

遇到的一个问题是当运行原来 ucore 中的测试程序, 不能通过测试, 而且 `printf` 得到的结果也和规范不同. 对已经提供的 `rust` 测试程序进行执行, 发现一些程序也存在一些问题.

`rust/shell` 运行之后即无响应, 在查看程序代码之后, 发现是存在无法读取输入,

ucore 的测试程序只有少数可以正确运行. 在对 `printf` 异常输入结果的分析当中, 我意识到是和机器位数导致的bug. 测试的 rust 是运行在64bits环境下, 而 ucore 的程序都是面向 32bits, 因此系统调用的相关接口不符合.

2.参考ucore+的测试用例，给rcore添加可能的测试用例，并对发现的小问题进行可能的修改，对发现的大问题提交完善建议

仿照 ucore 中的 sleep.c 样例, 编写了 `sleep.rs` 的测试用例, 测试了 sleep, get_time, waituid 这些系统调用.

```
#![no_std]
#![no_main]
#![feature(alloc)]
```

```rust
extern crate alloc;
#[macro_use]
extern crate rcore_user;

use alloc::vec::Vec;
use core::ptr;
use rcore_user::io::get_line;
use rcore_user::syscall::{sys_sleep, sys_vfork, sys_wait, sys_get_time,
sys_exit};

pub fn sleep(time: usize) -> i32 {
    sys_sleep(time)
}

pub fn gettime_msec() -> u32{
    sys_get_time() as u32
}

pub fn fork() -> i32 {
    sys_vfork()
}

pub fn waitpid(pid: usize, code: *mut i32) -> i32 {
    sys_wait(pid, code)
}

pub fn exit(error_code: usize) {
    sys_exit(error_code);
    println!("BUG: exit failed.");
    while true {};
}

fn sleepy(pid: usize) {
    let time: usize = 1;
    for i in 0..10 {
        sleep(time);
        println!("sleep {} x {} slices.", i + 1, time);
    }
    exit(0);
}

#[no_mangle]
pub fn main(){
    let time = gettime_msec();
    let mut pid1: usize = 0;
    let mut exit_code = 0;

    pid1 = fork() as usize;

    if pid1 == 0 {
        sleepy(pid1);
    } else {
        println!("child id is {}", pid1);
    }

    assert_eq!(waitpid(pid1, &mut exit_code), pid1 as i32);
    assert_eq!(exit_code, 0);
```

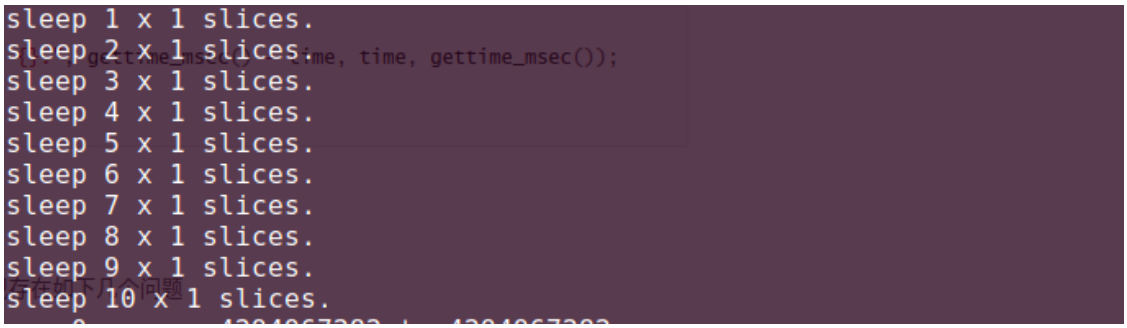
```
    println!("use {} msecs: {} to {}.", gettime_msec() - time, time,
gettime_msec());
    println!("sleep pass.");
}
```

在用 rust 完成用例的编写后运行测试, 发现存在如下几个问题

1. rCore 中实现的 sleep 系统调用是以秒为单位, 而之前的 ucore 是毫秒.

   如之前ucore运行 sleep.c 时, 系统调用参数为 100, 运行的结果是 100ms, 截图如下

   移植到 rCore 之后, 为了加速测试, 只能将参数改为 1, 即每次 sleep 一秒钟. 运行截图如下



   分析 `rcore` 的内部代码可以发现, 在提供的 sys_sleep 接口处, 直接设置了毫秒数为 0

```
pub fn sys_sleep(time: usize) -> i32 {
    let ts = TimeSpec {
        sec: time as u64,
        nsec: 0,
    };
    sys_call(
        SyscallId::Sleep,
        &ts as *const TimeSpec as usize,
        0,
        0,
        0,
        0,
        0,
    )
}
```

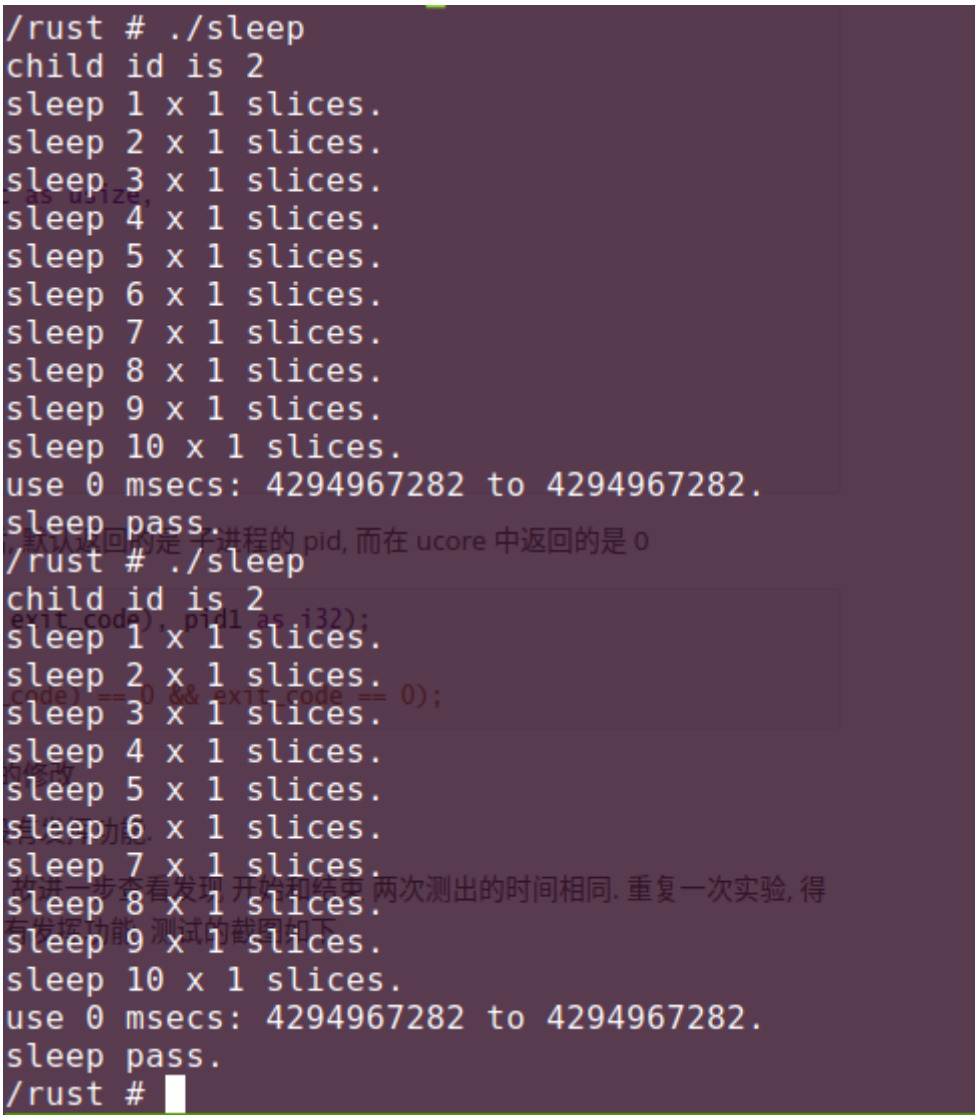
2. rCore 中 `sys_wait` 在子进程退出之后, 默认返回的是 子进程的 pid, 而在 ucore 中返回的是 0

```
assert_eq!(waitpid(pid1, &mut exit_code), pid1 as i32);
assert_eq!(exit_code, 0);
// assert(waitpid(pid1, &exit_code) == 0 && exit_code == 0);
```

可以看到, 在测试程序中需要进行相应的修改

3. rCore 中的 `sys_get_time` 接口似乎没有发挥功能.

   运行新的测试程序得到花费的时间为0, 故进一步查看发现 开始和结束 两次测出的时间相同. 重复一次实验, 得到的时间仍不变. 故认为 此系统调用没有发挥功能, 测试的截图如下



3.熟悉和理解rcore的代码结构和函数功能，并进行可能的注释补充

我后面的实验目标已经确认为完善和实现 `pool, select, epoll` 接口, 故注释的补充集中在现有的 `poll` 实现原理上, 如 `fs.rs` 中的 `sys_ppoll`, `sys_poll` 等