实验概述

1. 实现syscall统计

在 `TaskControlBlock` 中添加属性 `syscall_counts: Vec<SyscallRecord>`，其中`SyscallRecord` 结构如下

```
#[derive(Copy, Clone)]
pub struct SyscallRecord {
    pub syscall_id: usize,
    pub count: u32,
}
```

这里使用 Vec 而不是定长数组是因为内核堆空间有限(0x20000)，如果为每个 TaskControlBlock 都分配长度为500的usize数组，则会消耗很多内存。没有直接使用 HashMap 或 BTreeMap 是因为它们都在 Rust 的标准库中，因此无法使用。

在每个 TaskControlBlock 初始化时，设置属性 syscall_counts 为空数组，表示没有任何系统调用发生。每当有系统调用执行时，在根据 syscall_id 进入不同处理函数前，通过TASK_MANAGER找出当前所属的task的id，并更新系统调用的记录。

在进入系统调用 task_info 查询时，由于更新记录的操作发生在进入 sys_task_info 之前，所以此次查询结果会包含本次 task_info 的记录，满足实验要求。

2. 实现运行时间查询

在 TaskControlBlock 中添加属性 `start_time_ms: Option<usize>` ，为了方便，直接使用毫秒作为计时单位，初始设置为None。

所有任务第一次运行时要么作为第一个被运行任务发生在 TaskManager.run_first_task() ，要么发生在 run_next_task 切换任务时。因此在这两处先查询到当前时间，若即将执行的任务还没有设置开始时间，就将其设置为当前时间。task_info 查询时直接用当前时间减去任务开始时间即可。

3. 查询当前任务的状态

直接设置为Running 或 查询 TaskControlBlock 都可。

任务执行完成后没有清除系统调用的统计和开始时间等信息，因为目前一个TaskControlBlock只会被使用一次。

简答作业

1. 正确进入 U 态后，程序的特征还应有：使用 S 态特权指令，访问 S 态寄存器后会报错。 请同学们可以自行测试这些内容 (运行 Rust 三个 bad 测例 (ch2b_bad_*.rs) ， 注意在编译时至少需要指定 `LOG=ERROR` 才能观察到内核的报错信息) ， 描述程序出错行为，同时注意注明你使用的 sbi 及其版本。

版本: RustSBI version 0.3.0-alpha.4, adapting to RISC-V SBI v1.0.0

- bad_address: PageFault in application
- bad_instructions: IllegalInstruction in application 非法指令，用户态不能执行 `sret`
- bad_register: IllegalInstruction in application 非法指令 CSR相关指令或寄存器需要在S模型下执行

2. 深入理解 trap.S 中两个函数 `__alltraps` 和 `__restore` 的作用

1. 刚进入 `__restore` 时，a0 代表指向分配 Trap 上下文之后的内核栈栈顶， `__restore` 可以让 trap_handler 处理完系统调用/异常/中断之后，从内核态返回用户态，也可以作为新任务的初始化助手，让新任务开始在用户态执行。

2. sstatus寄存器是 S 特权级最重要的 CSR，可以从多个方面控制 S 特权级的 CPU 行为和执行状态。比如SPP字段可以给出trap前CPU处于哪一种模式。

sepc 寄存器 当 Trap 是一个异常的时候，记录 Trap 发生之前执行的最后一条指令的地址。当CPU用sret返回时，会回到 sepc 所指向的位置

sscratch 寄存器一般作为中转寄存器，比如存储用户态sp或内核态sp

3. x2就是sp寄存器，这时的sp指向的是内核栈，还要用它来恢复后面的寄存器，所以暂时跳过，之后可以直接用sscratch恢复x2(sp)。 x4寄存器一般不会用到，所以无需保存。

4. 指令执行后 sp 为用户栈 sp， sscratch 为内核栈sp

5. __restore状态切换在sret，执行后进入用户态是因为当前在S模式，硬件会正常执行并自动设置 sstatus 的 SPP字段为U，即进入用户态

6. sp为内核栈sp， sscratch保存用户栈sp

7. 通过ecall指令，或者在U态尝试执行S态特权指令，或者一些错误指令比如除0

个人想法

1. 最好在实验中加入GDB调试的内容，能更直观感受程序在各个阶段比如_restore, _switch是如何流转的。
2. 最好多留点时间，只是为了完成lab的话有点囫囵吞枣，马马虎虎地就要赶下一章了。

荣誉准则

1. 在完成本次实验的过程（含此前学习的过程）中，我曾分别与 **以下各位** 就（与本次实验相关的）以下方面做过交流，还在代码中对应的位置以注释形式记录了具体的交流对象及内容：

   无

2. 此外，我也参考了 **以下资料** ，还在代码中对应的位置以注释形式记录了具体的参考来源及内容：

   阅读了更详细的文档 rCore-Tutorial-Book 第三版

3. 我独立完成了本次实验除以上方面之外的所有工作，包括代码与文档。 我清楚地知道，从以上方面获得的信息在一定程度上降低了实验难度，可能会影响起评分。

4. 我从未使用过他人的代码，不管是原封不动地复制，还是经过了某些等价转换。 我未曾也不会向他人（含此后各届同学）复制或公开我的实验代码，我有义务妥善保管好它们。 我提交至本实验的评测系统的代码，均无意于破坏或妨碍任何计算机系统的正常运转。 我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按“-100”分计。