Report_Lab1

在实验三中，遇到了第一个Lab，要求我们实现一个新的系统调用：

`fn sys_task_info(ti: mut TaskInfo) -> isize` sysid 410

这个系统调用的功能是：

- 查询当前正在执行的任务信息，任务信息包括任务控制块相关信息（任务状态）、任务使用的系统调用及调用次数、系统调用时刻距离任务第一次被调度时刻的时长（单位ms）。

对应有一个数据结构：

```
structTaskInfo {
    status:TaskStatus,
    syscall_times: [u32; MAX_SYSCALL_NUM],
    time: usize
}
```

那么需要做的就是，修改相关的模块，在运行时候添加一些信息记录的逻辑，在这个系统调用被调用的时候呈现出来就好。

那么分开来看这个系统调用的要求：

1. 首先是查询任务状态，这里有点奇妙，因为由于查询的是当前任务的状态，因此 TaskStatus 一定是 Running。根据文档说这是助教涉及实验时候的一个小缺陷哈哈。那么这里就无需多说直接赋Running给status就好。
2. 其次是任务使用的系统调用及调用次数。这里就需要修改一些内核关键数据结构了。根据文档的说法，这里对于调用次数直接使用桶计数，因为是针对每个任务的，所以这个桶计数的数组必然是在任务控制块里面，于是在任务控制块中添加计数数组（这里的代码包含了后面添加的时间信息）。

   ```
   /// The task control block (TCB) of a task.
   #[derive(Copy, Clone)]
   pub struct TaskControlBlock {
       /// The task status in it's lifecycle
       pub task_status: TaskStatus,
       /// The task context
       pub task_cx: TaskContext,
       /// The numbers of syscall called by task
       pub task_syscall_times: [u32; MAX_SYSCALL_NUM],
       /// The time of this task be called First
       pub task_first_start: usize,
   }
   ```

   另外，显然的一点是，如果想要统计系统调用的使用，显然要在进入分发逻辑后，在分发之前进行记录，但这里会发现，无法通过直接获取TASK_MANAGER来修改对应的信息，这样不仅性能差，更会违背模块化原则甚至触发可变引用错误。

   由于各个组件之间的可见性以及模块化问题，我选择修改Task模块中的内容，向外提供修改/获取相关信息的接口。这里实现了TaskManager的一些功能，并且提供暴露的接口。

   ```
   /// set the current task syscalls times
   ```

```
    fn set_current_task_syscall_times(&self, syscall_id: usize) {
        let mut inner = self.inner.exclusive_access();
        let current = inner.current_task;
        inner.tasks[current].task_syscall_times[syscall_id] += 1;
    }

    /// get the syscalls_times about current task
    fn get_current_task_syscall_times(&self) -> [u32; MAX_SYSCALL_NUM]{
        let inner = self.inner.exclusive_access();
        let current = inner.current_task;
        // dont need clone because [u32; ..] have copy trait
        inner.tasks[current].task_syscall_times
    }
```

```
/// set the current task syscalls times
pub fn set_current_task_syscall_times(syscall_id: usize) {
    TASK_MANAGER.set_current_task_syscall_times(syscall_id);
}

/// get the current task syscalls times
pub fn get_current_task_syscall_times() -> [u32; MAX_SYSCALL_NUM]{
    TASK_MANAGER.get_current_task_syscall_times()
}
```

这样直接在syscall中分发前加上一行：

```
pub fn syscall(syscall_id: usize, args: [usize; 3]) -> isize {
    // set current task's syscall times
    set_current_task_syscall_times(syscall_id);
    ...
```

就可以完成记录。

3. 对于要求的时间记录，与上面的逻辑大致相同，添加了一个第一次被调用的时间信息，随后修改run_next_task函数中的逻辑，检查这个值，如果是初始化值（0），那么赋上当前的时间，在系统调用中只要实时获取一下时间，减去记录值即可（获取记录值的方式同样由TaskManager封装）。

```
fn run_next_task(&self) {
        if let Some(next) = self.find_next_task() {
            ...
            // give the first call time for this task
            if inner.tasks[next].task_first_start == 0 {
                inner.tasks[next].task_first_start = get_time_ms();
            }
            ...
    }
```

最后呈现的系统调用如下所示：

```
pub fn sys_task_info(ti: *mut TaskInfo) -> isize {
    trace!("kernel: sys_task_info");

    let status = TaskStatus::Running;
```

```
    let syscall_times = get_current_task_syscall_times();
    let time = get_time_ms() - get_first_time_becalled();

    unsafe {
        *ti = TaskInfo {
            status: status,
            syscall_times: syscall_times,
            time: time,
        };
    }
    0
}
```

这样就完成了Lab1的要求。