1. 简述功能

为了实现sys_task_info，我大体上做了如下工作：

1. 在TaskControlBlock中维护每个认为需要的syscall_times等task_info信息（这里忘记了可以直接用TaskInfo，而是直接维护了裸的信息）。
2. 在 os/src/syscall/mod.rs的syscall函数中维护每个系统调用的调用次数，供sys_task_info调用获取。
3. 在sys_task_info中将返回值中的status设置为Running。
4. 在 os/src/task/mod.rs的TaskManager类中，记录每个任务第一次被调度的时间。在sys_task_info中和获取并且和当前时间想见得到所求。注意这里获取时间似乎只能使用get_time_us。
5. 在 os/src/syscall/sys_write中，根据拿到的buf和len、以及当前任务的可用地址范围，判断是否存在越界。

2. 简答题

1. 三个错误样例：

```
[31m[ERROR] [kernel] PageFault in application, bad addr = 0x0, bad instruction = 0x8040008a, core dumped.
[31m[ERROR] [kernel] IllegalInstruction in application, core dumped.[0m
[31m[ERROR] [kernel] IllegalInstruction in application, core dumped.[0m
```

分别对应bad_address/bad_instruction/bad_register

第一个

```
(0x0 as *mut u8).write_volatile(0);
```

这句访问了不合法的地址。

第二个

```
core::arch::asm!("sret");
```

这句在用户态试图执行sret，但这是s态的特权级指令。

第三个

```
core::arch::asm!("csrr {}, status", out(reg) sstatus);
```

这句访问了不合法的寄存器 sstatus。

2.

1. L40：刚进入 `__restore` 时，`a0` 代表了什么值。请指出 `__restore` 的两种使用情景。

   __restore是从 _switch跳转过来的。考察switch中的代码可知，a0寄存器应当是指向cur_task的TaskContext的指针。

   两个使用情景：

   1. 用户程序执行系统调用后，从内核态返回到用户态的时候，用于恢复之前程序的上下文。
   2. 用户程序被中断打断或者触发其他异常时，内核切换到其他任务执行时，用于恢复该任务的上下文。

2. L46-L51：这几行汇编代码特殊处理了哪些寄存器？这些寄存器的的值对于进入用户态有何意义？请分别解释。

```
csrw sstatus, t0
csrw sepc, t1
csrw sscratch, t2
```

这部分代码分别恢复了sstatus、sepc和sscratch。其中：

sepc:指向发生异常/中断时的指令，对应于sret返回的位置（具体取决于是中断还是异常）。

sstatus:保存全局中断以及其他的状态，例如是否开了中断之类的。

sscratch:可以用来中转栈指针sp等数据（在ch4也用来中转其他指针）

3. L53-L59：为何跳过了 `x2` 和 `x4`？

   x4：表示线程指针tp，根本就没存。

   x2：表示sp，我们在48行的时候已经将其从栈上暂存到了sscratch中，然后在倒数第二行的时候:

```
csrrw sp, sscratch, sp
```

   将其恢复到了sp上。

4. L63：该指令之后，`sp` 和 `sscratch` 中的值分别有什么意义?

```
csrrw sp, sscratch, sp
```

   此时sp为用户栈sp，sscratch为内核栈sp。

5. `__restore`：中发生状态切换在哪一条指令？为何该指令执行之后会进入用户态?

   sret前为s态；sret后为u态。因为cpu对sret的实现即返回到异常切出去的位置并且将状态转换为u态。

6. L13：该指令之后，`sp` 和 `sscratch` 中的值分别有什么意义?

```
csrrw sp, sscratch, sp
```

   执行前sp为用户栈sp，sscratch为内核栈sp。这一步相当于交换两者中存的数据，执行后sp为内核栈sp而sscratch则变为用户栈sp。

7. 从 U 态进入 S 态是哪一条指令发生的?

   U态进入S态应当由CPU设置，或者说当CPU在U下检测到中断/异常的时候就会将状态切换为S。因此当我们进入trap.S后理应已经进入S态了。