实现功能

对于维护`TaskInfo`的`status`，直接返回`running`即可。

对于维护`TaskInfo`的`syscall_times`，我先为全局的`TASK_MANAGER`添加了`update_task_info`方法，里面包装了更新`syscall_times`的逻辑。这样每次在调用syscall时就直接调用`update_task_info`方法即可。

对于维护`TaskInfo`的`time`，直接调用了`timer`里面的`get_time_ms`函数。

简答作业

1. SBI版本：`RustSBI-QEMU Version 0.2.0-alpha.2`
    - 使用S态特权指令：`[kernel] IllegalInstruction in application, kernel killed it.`直接判断指令不合法，随后kernel关闭了对应的task。
    - 访问S态寄存器：`[kernel] IllegalInstruction in application, kernel killed it.`直接判断指令不合法，随后kernel关闭了对应的task。
    - 访问错误地址(0x0)：`[kernel] PageFault in application, bad addr = 0x0, bad instruction = 0x804003ac, kernel killed it.`出现了PageFault,于是kernel直接关闭了task.
2.
    1. `a0`代表了内核栈的栈顶。`__restore`既可以用来启动新的应用（在系统启动或者一个应用结束需要启动另一个程序时），也可以用来从`trap_handler`运行后恢复`context`
    2. 特殊处理了`sstatus`,`sepc`和`sscratch`。`sstatus`保存了Trap发生前cpu所处于的特权级；`sepc`保存了用户态从trap恢复后继续运行的指令地址，因此很重要。`sscratch`保存了进入内核栈的栈顶地址，对于下一次触发trap很重要。
    3. `x2`为sp，因为在`__alltrap`触发之前就已经被保存在了`sscratch`中，因此无需保存。`x4`为`tp`，因为不常用所以也没被保存。
    4. 该指令交换了`sscratch`和`sp`的值，即在进入用户态前把`sp`设置为用户态栈顶，并将`sscratch`设置为内核态栈顶。
    5. 应该是`sret`,这个是riscV规定的吧。
    6. 发生之后，`sp`为内核栈栈顶，`sscratch`为用户栈栈顶。
    7. 不太确定，应该是`call trap_handler`吧。

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：

    > ChatGPT：关于Rust一些语法相关的内容。

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：

    > rcoreTutorial文档：https://learningos.cn/rCore-Tutorial-Guide-2024S。

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。