简要总结

在系统任务信息（sys_task_info）模块中，对任务的三个信息进行统计与展示。首先，追踪任务的运行状态，即任务是处于运行中、就绪状态还是被阻塞。其次，记录每个任务进行系统调用（syscall）的次数，用于分析任务与操作系统交互的频繁程度。最后，测量并统计任务开始运行到当前的时长。

问题简答

1. PageFault, IllegalInstruction, rustsbi 0.2.0-alpha.1
2.

1) 刚进入 _restore 时，a0 代表了任务上下文（taskContext）的指针。这是因为在 _alltraps 处理函数中，a0 被用来传递 TrapContext 的地址给 trap_handler 函数，而在 _restore 函数中，a0 被用来恢复任务上下文。__restore 的两种使用情景包括：系统启动时，从内核态进入用户态，开始运行应用程序。中断或异常处理完成后，从内核态返回用户态，继续执行被中断的应用程序。

2) sstatus：保存了陷阱发生前的 CPU 状态信息，包括当前的特权级别（S/U 模式）。sepc：保存了陷阱发生前执行的最后一条指令的地址，用于返回到发生陷阱的下一条指令继续执行。sscratch：保存了用户栈指针（sp），用于恢复用户态的栈环境。这些寄存器使得从内核态返回到用户态时，程序可以从正确的上下文位置继续正确执行。

3) 在 trap.S 中的代码段 L50-L56 中跳过了 x2 和 x4 的原因是：x2 对应的用户栈指针（sp）保存到了 sscratch 寄存器中，不需要从内核栈中进行恢复。x4（tp）一般不会被用到，除非手动使用它，因此没有必要保存；而 sp（x2）后面还要使用，需要依靠栈指针加偏移量来找到其他寄存器应该保存的正确位置。

4) sp 指向用户栈，这是因为该指令将 sscratch（保存用户栈指针）的值加载到 sp 中，使得 sp 指向用户栈。sscratch 指向内核栈，因为这条指令实际上是交换了 sscratch 和 sp 的值，执行前 sp 指向内核栈指针，sscratch 指向用户栈指针；执行后，sp 指向用户栈，为回到用户态做准备。

5) 状态切换发生在执行 csrw sstatus, t0 这条指令时。这条指令将 t0（从栈中加载的值，保存了用户态的 sstatus 寄存器值）写入 sstatus 寄存器，从而改变了处理器的状态，使得处理器进入用户态（U 模式）。这是因为 sstatus 寄存器控制着处理器的当前特权级别，通过设置正确的值，可以切换到用户态。

6) 第一个 sp 是目标寄存器，它将接收 sscratch CSR 的旧值。sscratch 是被操作的 CSR 寄存器。第二个 sp 是源寄存器，它当前的值将被写入 sscratch CSR

7)ecall

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：

   郑超群

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：

   参考了《Rust 编程语言》官方文档

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。