lab1

范逍宇 **2019013273**

实验报告

一共修改了四个文件，每个文件修改的内容如下：

src/task/task.rs

在 TaskControlBlock 中增加两个变量 start_time 和 syscall_times 分别用来记录程序开始运行的时间和每种 syscall 的调用次数。

src/task/mod.rs

在构造 TASK_MANAGER 时加入对 start_time 和 syscall_times 的初始化操作，否则会报错。

修改 TaskManager 的 run_first_task 和 run_next_task，如果这是第一次运行某个任务，就将该任务的 start_time 记录为当前的时间。

src/syscall/mod.rs

修改 syscall 函数，在进入具体的 syscall 函数之前，根据 syscall_id 修改当前任务的 syscall_times，将对应的 syscall 类型的计数加一。

src/syscall/process.rs

在 sys_task_info 函数中将当前进程的 TaskInfo 赋给 ti，而 ti 是用户程序传递的参数的引用，这样就实现了将信息反馈给用户程序。

思考题

1.运行 ch2b_bad_address.rs，ch2b_bad_instructions.rs，ch2b_bad_register.rs 分别报错如下：

```
[ERROR] [kernel] PageFault in application, core dumped.
[ERROR] [kernel] IllegalInstruction in application, core dumped.
[ERROR] [kernel] IllegalInstruction in application, core dumped.
```

这说明访问非法地址、使用 S 态特权指令（sret），访问 S 态寄存器（sstatus）后会报错。

我使用的 RustSBI 版本为 0.2.0-alpha.4。

2.

（1）刚进入 __restore 时，a0 的值为分配 Trap 上下文之后的内核栈栈顶地址，因为 restore 在 trap_handler 函数返回之后调用，而 trap_handler 的返回值为传入的 cx: &mut TrapContext，即 Trap 上下文的内核栈栈顶地址。

在开始运行用户程序和 trap 处理完之后返回用户态都会用到 restore 函数。

（2）这几行代码恢复了 sstatus，sepc，sscratch 寄存器，其中 sstatus 寄存器保存了`SPP`等字段，给出 Trap 发生之前 CPU 处在哪个特权级（S/U）等信息，即将当前状态恢复为用户态，sepc 记录了 Trap 发生之前执行的最后一条指令的下一条指令的地址，确定了用户态将要执行的代码，sscratch 保存了用户栈的栈顶指针，从而找到之前的用户栈继续执行。

（3）因为 x2（sp）在下面通过 sscratch 恢复（且现在 sp 的任务还未完成，下面还要用到 sp 来释放 trap 上下文），而对于 x4 （tp）寄存器来说，除非我们手动出于一些特殊用途使用它，否则它一般也不会被用到，所以也无需处理 x4 。

（4）sp 指向用户程序栈的栈顶，sscratch 指向内核栈的栈顶。

（5）sret，这是 RISCV 的规定，执行 sret 指令后，停止执行当前程序流，转而从 csr 寄存器 sepc 定义的 pc 地址开始执行执行，同时硬件更新 sstatus 的 sie 域为之前的状态（spie 的值），即从内核态进入了用户态。

（6）sp 指向内核栈的栈顶，sscratch 指向用户栈的栈顶。

（7）L13：csrrw sp, sscratch, sp