|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
| 北京交通大学软件学院  **《操作系统》课程**  **实验报告** | | |

|  |
| --- |
| **姓名:韩熔** |
| **学号:20301036** |

目录

[1 实验目的 3](#_Toc118133029)

[2 实验过程设计 3](#_Toc118133030)

[3 源代码及注释 3](#_Toc118133031)

[4 运行结果与分析 4](#_Toc118133032)

[5 实验总结 4](#_Toc118133033)

# 1 实验目的

实现操作系统对中断的处理和对应用程序的抢占，并设计实现更加公平和高效交互的抢占式操作系统

# 2 实验过程设计

（1）运行分时多任务系统应用,观察到应用程序在执行过程中确实被抢占了

（2）学习时钟中断与计时器的实现

（3）学习抢占式调度的实现

# 3 源代码及注释

（说明：写出本次实验使用的命令或编写的源代码。请在下列表格中说明源代码的文件名和代码功能概述或者命令名和该命令的主要作用。要求针对核心功能代码进行注释。）

表 3-1 源代码及注释

|  |  |
| --- | --- |
| **文件名:** | os/src/timer.rs |
| **主要功能:** | 实现计时器的控制 |
| **源代码:**  **use riscv::register::time;**  **// get\_time 函数可以取得当前 mtime 计数器的值**  **pub fn get\_time() -> usize {**  **time::read()**  **}**  **use crate::config::CLOCK\_FREQ;**  **const TICKS\_PER\_SEC: usize = 100;**  **//** **timer 子模块的 set\_next\_trigger 函数对 set\_timer 进行了封装**  **pub fn set\_next\_trigger() {**  **//它首先读取当前 mtime 的值，然后计算出 10ms 之内计数器的增量，再将 mtimecmp 设置为二者的和**  **//这样，10ms 之后一个 S 特权级时钟中断就会被触发。**  **set\_timer(get\_time() + CLOCK\_FREQ / TICKS\_PER\_SEC);**  **}**  **const MSEC\_PER\_SEC: usize = 1000;**  **//计时**  **//以毫秒为单位返回当前计数器的值**  **pub fn get\_time\_ms() -> usize {**  **time::read() / (CLOCK\_FREQ / MSEC\_PER\_SEC)**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | os/src/sbi.rs |
| **主要功能:** | 包装SBI 调用 |
| **源代码:**  **const SBI\_SET\_TIMER: usize = 0;**  **//set\_timer 调用，是一个由 SEE 提供的标准 SBI 接口函数，它可以用来设置 mtimecmp 的值。**  **pub fn set\_timer(timer: usize) {**  **sbi\_call(SBI\_SET\_TIMER, timer, 0, 0);**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | os/src/trap/mod.rs |
| **主要功能:** | 处理trap |
| **源代码:**  **//条件分支跳转**  **match scause.cause() {**  **//触发了一个 S 特权级时钟中断时**  **Trap::Interrupt(Interrupt::SupervisorTimer) => {**  **//设置一个 10ms 的计时器，**  **set\_next\_trigger();**  **//暂停当前应用并切换到下一个**  **suspend\_current\_and\_run\_next();**  **}**  **}**  **use riscv::register::sie;**  **pub fn enable\_timer\_interrupt() {**  **unsafe { sie::set\_stimer(); }**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | os/src/main.rs |
| **主要功能:** | 程序入口 |
| **源代码:**  **#[no\_mangle]**  **pub fn rust\_main() -> ! {**  **clear\_bss();**  **println!("[kernel] Hello, world!");**  **trap::init();**  **loader::load\_apps();**  **//设置了 sie.stie 使得 S 特权级时钟中断不会被屏蔽**  **trap::enable\_timer\_interrupt();**  **//设置第一个 10ms 的计时器**  **timer::set\_next\_trigger();**  **task::run\_first\_task();**  **panic!("Unreachable in rust\_main!");**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | user/src/bin/03sleep.rs |
| **主要功能:** | 应用程序，等待 3000ms 然后退出 |
| **源代码:**  **#[no\_mangle]**  **fn main() -> i32 {**  **let current\_timer = get\_time();**  **let wait\_for = current\_timer + 3000;**  **while get\_time() < wait\_for {**  **//主动交出 CPU**  **yield\_();**  **}**  **println!("Test sleep OK!");**  **0**  **}** | |

|  |  |
| --- | --- |
| **文件名:** | os/src/syscall/mod.rs  os/src/syscall/process.rs |
| **主要功能:** | 新增一个系统调用，获取当前的时间 |
| **源代码:**  **//** **os/src/syscall/mod.rs**  **//** **syscall ID：169**  **const SYSCALL\_GET\_TIME: usize = 169;**  **mod fs;**  **mod process;**  **use fs::\*;**  **use process::\*;**  **pub fn syscall(syscall\_id: usize, args: [usize; 3]) -> isize {**  **match syscall\_id {**  **SYSCALL\_GET\_TIME => sys\_get\_time(),**  **\_ => panic!("Unsupported syscall\_id: {}", syscall\_id),**  **}**  **}**  **//** **os/src/syscall/process.rs**  **use crate::task::{exit\_current\_and\_run\_next, suspend\_current\_and\_run\_next};**  **use crate::timer::get\_time\_ms;**  **//以毫秒为单位获取时间**  **pub fn sys\_get\_time() -> isize {**  **get\_time\_ms() as isize**  **}** | |

表 3-2 关键命令解释

|  |  |  |
| --- | --- | --- |
| **序号** | **命令名** | **主要作用** |
| **1** | **make run -j $(nproc)** | 构建并运行代码 |
| **2** |  |  |
| **3** |  |  |
| **…** |  |  |

# 4 运行结果与分析

（1）更新代码仓库



（2）构建并运行代码



三个任务的计算结果交错输出，这说明应用程序在执行过程中确实被抢占了。

# 5 实验总结

上次实验实现了协作式操作系统，依靠应用程序之间的协作来完成任务调度和切换。

但协作式操作系统中任务调度的主动权在于应用程序的“自觉性”上，操作系统自身缺少强制的任务调度的手段，所以这次实验我们通过时钟中断来实现抢占式调度并强行切换应用，从而提高了应用执行的灵活性、公平性和交互效率。