练习5

把协作式调度算法fifo改造为抢占式调度算法。让测试应用通过

1 准备

首先，在apps下面增加一个测试应用ex5，它的main.rs

```
#![cfg_attr(feature = "axstd", no_std)]
#![cfg_attr(feature = "axstd", no_main)]

#[macro_use]
#[cfg(feature = "axstd")]
extern crate axstd as std;

use std::sync::atomic::{AtomicUsize, Ordering};
use std::thread;

static FLAG: AtomicUsize = AtomicUsize::new(0);

#[cfg_attr(feature = "axstd", no_mangle)]
fn main() {
    thread::spawn(move || {
        println!("Spawned-thread is waiting ...");
        while FLAG.load(Ordering::Relaxed) < 1 {
            // For cooperative scheduler, we must yield here!
            // For preemptive scheduler, just relaxed! Leave it for scheduler.
        }

        let _ = FLAG.fetch_add(1, Ordering::Relaxed);
    });

    // Give spawned thread a chance to start.
    thread::yield_now();

    println!("Main thread set FLAG to notify spawned-thread to continue.");
    let _ = FLAG.fetch_add(1, Ordering::Relaxed);
    println!("Main thread waits spawned-thread to respond ...");
    while FLAG.load(Ordering::Relaxed) < 2 {
        thread::yield_now();
    }
    println!("Preempt test run OK!");
}
```

ex5的Cargo.toml的内容

```
[package]
name = "ex5"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-
lang.org/cargo/reference/manifest.html

[dependencies]
axstd = { path = "../../ulib/axstd", features = ["alloc", "multitask", "irq"],
optional = true }
```

另外需要把"apps/ex5"加到**工程根目录**Cargo.toml的[workspace]下的members中。

然后，尝试运行一下，`make ARCH=riscv64 A=apps/ex5 run`

```
       d8888                            .d88888b.   .d8888b.
      d88888                           d88P" "Y88b d88P  Y88b
     d88P888                           888     888 Y88b.
    d88P 888 888d888  .d8888b  .d88b.  888     888  "Y888b.
   d88P  888 888P"   d88P"    d8P  Y8b 888     888     "Y88b.
  d88P   888 888     888      88888888 888     888       "888
 d8888888888 888     Y88b.    Y8b.     Y88b. .d88P Y88b  d88P
d88P     888 888      "Y8888P  "Y8888   "Y88888P"   "Y8888P"

arch = riscv64
platform = riscv64-qemu-virt
target = riscv64gc-unknown-none-elf
smp = 1
build_mode = release
log_level = warn

Spawned-thread is waiting ...
```

这样**会卡住!!!**

原因就是默认调度策略fifo是协作式，大家可以参照ex5的应用逻辑想想。

如果希望运行下去，有两个办法：

一是加上yield_now，如下

```
        while FLAG.load(Ordering::Relaxed) < 1 {
            // For cooperative scheduler, we must yield here!
            // For preemptive scheduler, just relaxed! Leave it for scheduler.
            // 在这里加 thread::yield_now();
        }
```

大家可以试试！

二就是修改调度算法，也就是本练习的题目。**注意**：如果刚刚加上yield_now做实验了，现在别忘了删除这行:)

2 题目

要求:

直接修改crates/scheduler/src/fifo.rs，让ex5通过。

提示:

1. 可以比照crates/scheduler/src/round_robin.rs的实现进行修改。(请务必要做一遍!!!)
2. 虽然本题目简单，但是如何为修改后的fifo调度算法启用feature "preempt"，可能有点小麻烦，看看rr/cfs是怎么传递的。

预期:

运行`make ARCH=riscv64 A=apps/ex5 run`

```
arch = riscv64
platform = riscv64-qemu-virt
target = riscv64gc-unknown-none-elf
smp = 1
build_mode = release
log_level = warn

Spawned-thread is waiting ...
Main thread set FLAG to notify spawned-thread to continue.
Main thread waits spawned-thread to respond ...
Preempt test run OK!
```

练习5是自主练习，成功后发截图到群，**不用**发邮件。