一. 实验步骤

1. 时钟中断与计时器

1.1 实现timer子模块获取mtime的值

```rust
use riscv::register::time;

pub fn get_time() -> usize {
    time::read()
}
```

1.2 实现sbi子模块

```rust
use core::arch::asm;

const SBI_SET_TIMER: usize = 0;
const SBI_CONSOLE_PUTCHAR: usize = 1;
const SBI_CONSOLE_GETCHAR: usize = 2;
const SBI_CLEAR_IPI: usize = 3;
const SBI_SEND_IPI: usize = 4;
const SBI_REMOTE_FENCE_I: usize = 5;
const SBI_REMOTE_SFENCE_VMA: usize = 6;
const SBI_REMOTE_SFENCE_VMA_ASID: usize = 7;
const SBI_SHUTDOWN: usize = 8;
const SBI_SET_TIMER: usize = 0;

pub fn set_timer(timer: usize) {
    sbi_call(SBI_SET_TIMER, timer, 0, 0);
}


#[inline(always)]
fn sbi_call(which: usize, arg0: usize, arg1: usize, arg2: usize) -> usize {
    let mut ret;
```

1.3 在timer子模块进行封装

```
@5738fcf47534:/mnt

use riscv::register::time;
use crate::sbi::set_timer;
use crate::config::CLOCK_FREQ;

const TICKS_PER_SEC: usize = 100;

pub fn set_next_trigger() {
    set_timer(get_time() + CLOCK_FREQ / TICKS_PER_SEC);
}

pub fn get_time() -> usize {
    time::read()
}
~
~
~
~
~
~
~
~
~
-- INSERT --                                         9,2          All
```

```
@5738fcf47534:/mnt

use riscv::register::time;
use crate::sbi::set_timer;
use crate::config::CLOCK_FREQ;

const TICKS_PER_SEC: usize = 100;
const MSEC_PER_SEC: usize = 1000;

pub fn get_time_ms() -> usize {
    time::read() / (CLOCK_FREQ / MSEC_PER_SEC)
}

pub fn set_next_trigger() {
    set_timer(get_time() + CLOCK_FREQ / TICKS_PER_SEC);
}

pub fn get_time() -> usize {
    time::read()
}
~
~
~
~
:wq
```

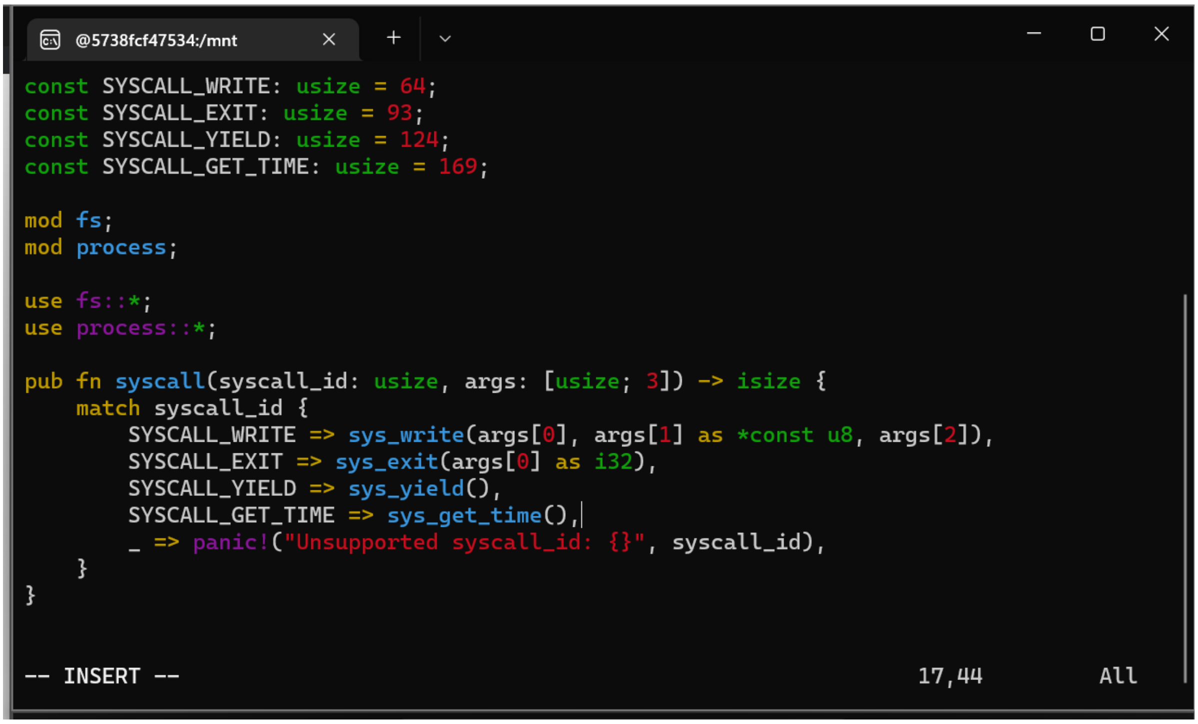
1.4 修改config.js增加常量

```
@5738fcf47534:/mnt

pub const USER_STACK_SIZE: usize = 4096 * 2;
pub const KERNEL_STACK_SIZE: usize = 4096 * 2;
pub const MAX_APP_NUM: usize = 4;
pub const APP_BASE_ADDRESS: usize = 0x80400000;
pub const APP_SIZE_LIMIT: usize = 0x20000;
pub const CLOCK_FREQ: usize = 12500000;
~
-- INSERT --                                  6,40        All
```

1.5 修改syscall子模块

```
@5738fcf47534:/mnt

use crate::task::{
    suspend_current_and_run_next,
    exit_current_and_run_next,
};

pub fn sys_exit(exit_code: i32) -> ! {
    println!("[kernel] Application exited with code {}", exit_code);
    exit_current_and_run_next();
    panic!("Unreachable in sys_exit!");
}

pub fn sys_yield() -> isize {
    suspend_current_and_run_next();
    0
}

use crate::timer::get_time_ms;

pub fn sys_get_time() -> isize {
    get_time_ms() as isize
}
~
-- INSERT --                                  21,2        All
```

1.6 修改mod.rs增加获取时间的系统调用



```
const SYSCALL_WRITE: usize = 64;
const SYSCALL_EXIT: usize = 93;
const SYSCALL_YIELD: usize = 124;
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_WRITE => sys_write(args[0], args[1] as *const u8, args[2]),
        SYSCALL_EXIT => sys_exit(args[0] as i32),
        SYSCALL_YIELD => sys_yield(),
        SYSCALL_GET_TIME => sys_get_time(),
        _ => panic!("Unsupported syscall_id: {}", syscall_id),
    }
}

-- INSERT --                                        17,44          All
```

2. 修改应用程序

2.1 增加get_time系统调用

```
use core::arch::asm;

const SYSCALL_WRITE: usize = 64;
const SYSCALL_EXIT: usize = 93;
const SYSCALL_GET_TIME: usize = 169;

pub fn sys_get_time() -> isize {
    syscall(SYSCALL_GET_TIME, [0, 0, 0])
}


fn syscall(id: usize, args: [usize; 3]) -> isize {
    let mut ret: isize;
    unsafe {
        asm!("ecall",
             in("x10") args[0],
             in("x11") args[1],
             in("x12") args[2],
             in("x17") id,
             lateout("x10") ret
        );
    }
:wq
```

2.2 增加get_time用户库封装

```
#![no_std]
#![feature(linkage)]
#![feature(panic_info_message)]

#[macro_use]
pub mod console;
mod syscall;
mod lang_items;

use syscall::*;

pub fn write(fd: usize, buf: &[u8]) -> isize { sys_write(fd, buf) }
pub fn exit(exit_code: i32) -> isize { sys_exit(exit_code) }
pub fn get_time() -> isize { sys_get_time() }


fn clear_bss() {
    extern "C" {
        fn start_bss();
        fn end_bss();
    }
    (start_bss as usize..end_bss as usize).for_each(|addr| {
                                                    14,45         Top
```

2.3 实现新测试应用

`00power_3.rs`

```
#![no_std]
#![no_main]

#[macro_use]
extern crate user_lib;

const LEN: usize = 100;

#[no_mangle]
fn main() -> i32 {
    let p = 3u64;
    let m = 998244353u64;
    let iter: usize = 200000;
    let mut s = [0u64; LEN];
    let mut cur = 0usize;
    s[cur] = 1;
    for i in 1..=iter {
        let next = if cur + 1 == LEN { 0 } else { cur + 1 };
        s[next] = s[cur] * p % m;
        cur = next;
        if i % 10000 == 0 {
            println!("power_3 [{}/{}]", i, iter);
                                                    1,1           Top
```

`01power_5.rs`

```
@5738fcf47534:/mnt

const LEN: usize = 100;

#[no_mangle]
fn main() -> i32 {
    let p = 5u64;
    let m = 998244353u64;
    let iter: usize = 200000;
    let mut s = [0u64; LEN];
    let mut cur = 0usize;
    s[cur] = 1;
    for i in 1..=iter {
        let next = if cur + 1 == LEN { 0 } else { cur + 1 };
        s[next] = s[cur] * p % m;
        cur = next;
        if i % 10000 == 0 {
            println!("power_5 [{}/{}]", i, iter);
        }
    }
    println!("{}^{} = {}", p, iter, s[cur]);
    println!("Test power_5 OK!");
    0
}
-- INSERT --                                               22,30         Bot
```

`02power_7.rs`

```
@5738fcf47534:/mnt

const LEN: usize = 100;

#[no_mangle]
fn main() -> i32 {
    let p = 7u64;
    let m = 998244353u64;
    let iter: usize = 200000;
    let mut s = [0u64; LEN];
    let mut cur = 0usize;
    s[cur] = 1;
    for i in 1..=iter {
        let next = if cur + 1 == LEN { 0 } else { cur + 1 };
        s[next] = s[cur] * p % m;
        cur = next;
        if i % 10000 == 0 {
            println!("power_7 [{}/{}]", i, iter);
        }
    }
    println!("{}^{} = {}", p, iter, s[cur]);
    println!("Test power_7 OK!");
    0
:wq
```

`03power_7.rs`

```
#![no_std]
#![no_main]

#[macro_use]
extern crate user_lib;

use user_lib::{get_time, yield_};

#[no_mangle]
fn main() -> i32 {
    let current_timer = get_time();
    let wait_for = current_timer + 3000;
    while get_time() < wait_for {
        yield_();
    }
    println!("Test sleep OK!");
    0
}
```

3. 实现抢占式调度

修改`os/src/trap/mod.rs`

```
            cx.x[10] = syscall(cx.x[17], [cx.x[10], cx.x[11], cx.x[12]]) as usize;
        }
        Trap::Exception(Exception::StoreFault) |
        Trap::Exception(Exception::StorePageFault) => {
            println!("[kernel] PageFault in application, bad addr = {:#x}, bad instruction = {:#x}, core dumped.", stval, cx.sepc);
            exit_current_and_run_next();
        }
        Trap::Exception(Exception::IllegalInstruction) => {
            println!("[kernel] IllegalInstruction in application, core dumped.");
            exit_current_and_run_next();
        }
        Trap::Interrupt(Interrupt::SupervisorTimer) => {
            set_next_trigger();
            suspend_current_and_run_next();
        }
        _ => {
            panic!("Unsupported trap {:?}, stval = {:#x}!", scause.cause(), stval);
        }
    }
    cx
}
```

修改`main.rs`

```
@5738fcf47534:/mnt

fn clear_bss() {
    extern "C" {
        fn sbss();
        fn ebss();
    }
    (sbss as usize..ebss as usize).for_each(|a| {
        unsafe { (a as *mut u8).write_volatile(0) }
    });
}

#[no_mangle]
pub fn rust_main() -> ! {
    clear_bss();
    println!("[kernel] Hello, world!");
    trap::init();
    trap::enable_timer_interrupt();
    timer::set_next_trigger();
    loader::load_apps();
    task::run_first_task();
    panic!("Unreachable in rust_main!");
}
-- INSERT --                                          38,5          Bot
```

4. 运行成功!

> 要在user目录下再次执行make build将应用程序编译成二进制文件

```
@5738fcf47534:/mnt/os

power_3 [90000/200000]
power_3 [100000/200000]
power_3 [110000/200000]
power_3 [120000/200000]
power_3 [130000/200000]
power_3 [140000/200000]
power_3 [150000/200000]
power_3 [160000/200000]
power_3 [170000/200000]
power_3 [180000/200000]
power_3 [power_5 [10000/200000]
power_5 [20000/200000]
power_5 [30000/200000]
power_5 [40000/200000]
power_5 [50000/200000]
power_5 [60000/200000]
power_5 [70000/200000]
power_5 [80000/200000]
power_5 [90000/200000]
power_5 [100000/200000]
power_5 [110000/200000]
power_5 [120000/200000]
power_5 [130000/200000]
power_5 [140000/200000]
power_5 [150000/200000]
power_5 [160000/200000]
power_5 [170000/200000]
power_5 [180000/200000]
power_5 [190000/200000]
power_5 [200000/200000]
5^200000 = 670295496
Test power_5 OK!
[kernel] Application exited with code 0
power_7 [190000/200000]
power_3 [200000/200000]
3^200000 = 871008973
Test power_3 OK!
[kernel] Application exited with code 0
10000/200000]
power_7 [20000/200000]
power_7 [30000/200000]
power_7 [40000/200000]
power_7 [50000/200000]
power_7 [60000/200000]
power_7 [70000/200000]
power_7 [80000/200000]
power_7 [90000/200000]
power_7 [100000/200000]
power_7 [110000/200000]
power_7 [120000/200000]
```

> 箭头指出的地方显示power_3运行时切换为power_5，又在power_5执行一段时间后切换回power_3

二. 思考题

1. 分析分时多任务是如何实现的

首先基于`mtime`和`mtimecmp`实现定时器，当时间超过`mtimecmp`时触发时钟中断，然后通过调用`suspend_current_and_run_next`实现应用的切换，实现分时多任务

2. 分析抢占式调度是如何设计和实现的

本实验中抢占式调度是通过时钟中断实现的，当中断发生时，表示当前任务的时间片已用尽，操作系统将保存当前任务的状态，使用`suspend_current_and_run_next()`选择另一个任务继续执行。这种机制确保了系统对任务的执行有更严格的控制，避免了某些任务过度占用CPU资源。

3. 对比上个实验实现的协作式调度与本实验实现的抢占式调度

实验4中的协作式调度是由应用程序调用`yield`主动让出CPU资源，而本实验中切换任务是在任务运行到达时间限制后触发中断实现的，由操作系统触发

三. Git提交截图