19271169-张东植-实验报告5

19271169-张东植-实验报告5
1. 时钟中断与计时器
2. 修改程序
    2.1 get_time
    2.2 Test Demo
3. 抢占式调度

- **Gitlab repo: http://202.205.102.126:88/ZhangDongZhi/os-lab.git**

The main purpose of this experiment is to implement a time-sharing multi-task and preemptive scheduling operating system. Through this experiment, we can realize a time-sharing multi-task and preemptive scheduling operating system. This includes clock interrupt and timer, application modification, preemptive scheduling, etc.

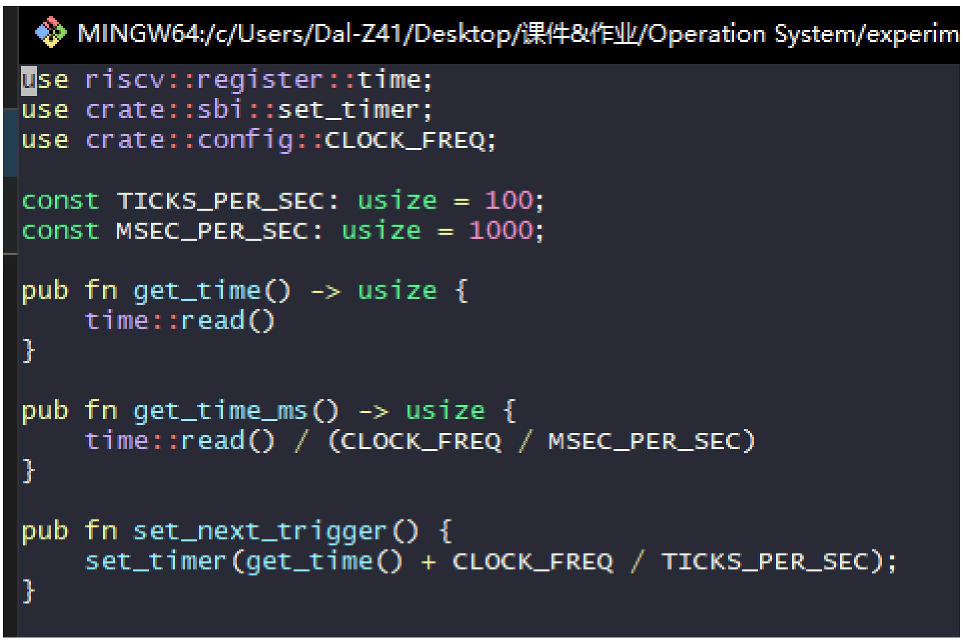
1. 时钟中断与计时器

In risc-v 64 architecture, there are two status registers, **mtime** and **mtimecmp**. mtime counts the clock period of the built-in clock since power-on. mtimecmp triggers a clock interrupt when mtime exceeds mtimecmp.

1. First, implement the timer submodule to get the value of mtime.

```
//os /src/timer.rs
use riscv::register::time;

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



2. Next, the value of mtimECMP is set in the SBI submodule implementation and encapsulated in the Timer submodule.

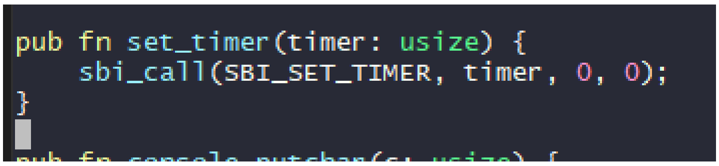
```
//os/src/sbi.rs
const SBI_SET_TIMER: usize = 0;

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

//os/src/timer.rs
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 set_timer(timer: usize) {
    sbi_call(SBI_SET_TIMER, timer, 0, 0);
}
```

3. At the same time, for subsequent timing operations, we also need to wrap another function in the timer submodule that returns the value of the current counter in milliseconds.

```
//os/src/timer.rs
const MSEC_PER_SEC: usize = 1000;

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

4. Since the above two functions use constants in "config.rs", so we should also modify file "**config.rs**".

```
pub const CLOCK_FREQ: usize = 12500000;
```

5. Finally, we need to modify **"syscall"** submodule to add the implementation of the **get_time** system call. And add the following codes to file **"process.rs"**. Also, change OS/SRC /syscall/mod.rs to add get_time system call handling.

```
use crate::timer::get_time_ms;

pub fn sys_get_time() -> isize {
    get_time_ms() as isize
}

const SYSCALL_GET_TIME: usize = 169;

SYSCALL_GET_TIME => sys_get_time(),
```

```
pub fn sys_get_time() -> isize {
    get_time_ms() as isize
}
~
```

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/gardeneros/os/s

```
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),
    }
}
```

2. 修改程序

This section includes some modifications on the program to make it preemptive and time-sharing.

2.1 get_time

1. First, add the get_time system call to **"syscall.rs"** with the following code.

```
//user/src/syscall.rs
const SYSCALL_GET_TIME: usize = 169;

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

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

2. Then add the get_time user library wrapper to **"lib.rs"** and add the following code.

```
//user/src/lib.rs
pub fn get_time() -> isize { sys_get_time() }
```

```
pub fn yield_() -> isize { sys_yield() }
pub fn get_time() -> isize { sys_get_time() }
```

2.2 Test Demo

- **00power_3.rs :**

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/co

```rust
#![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);
        }
    }
    println!("{}^{} = {}", p, iter, s[cur]);
    println!("Test power_3 OK!");
    0
}
```

- **01power_5.rs :**

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/

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

#[macro_use]
extern crate user_lib;

const LEN: usize = 100;

#[no_mangle]
fn main() -> i32 {
    let p = 5u64;
    let m = 998244353u64;
    let iter: usize = 140000;
    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
}
~
```

- **02power_7.rs :**

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/

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

#[macro_use]
extern crate user_lib;

const LEN: usize = 100;

#[no_mangle]
fn main() -> i32 {
    let p = 7u64;
    let m = 998244353u64;
    let iter: usize = 160000;
    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
}
~
```

- **03sleep.rs :**

MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operati

```rust
#![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. 抢占式调度

Once the clock interrupts and timers are complete, it's easy to implement preemptive scheduling. Modify file **"mod.rs"** code as below.

```rust
//os/src/trap/mod.rs

use riscv::register::{
```

```
    mtvec::TrapMode,
    stvec,
    scause::{
        self,
        Trap,
        Exception,
        Interrupt,
    },
    stval,
    sie,
};

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

use crate::timer::set_next_trigger;


pub fn enable_timer_interrupt() {
    unsafe { sie::set_stimer(); }
}

#[no_mangle]
pub fn trap_handler(cx: &mut TrapContext) -> &mut TrapContext {
    let scause = scause::read();
    let stval = stval::read();
    match scause.cause() {
        Trap::Exception(Exception::UserEnvCall) => {
            cx.sepc += 4;
            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
}
```

```
MINGW64:/c/Users/Dal-Z41/Desktop/课件&作业/Operation System/experiment/code/
mod context;

use riscv::register::{
    mtvec::TrapMode,
    stvec,
    scause::{
        self,
        Trap,
        Exception,
        Interrupt,
    },
    stval,
    sie,
};
use crate::syscall::syscall;
use crate::task::{
    exit_current_and_run_next,
    suspend_current_and_run_next,
};

use crate::timer::set_next_trigger;

global_asm!(include_str!("trap.S"));

pub fn init() {
    extern "C" { fn __alltraps(); }
    unsafe {
        stvec::write(__alltraps as usize, TrapMode::Direct);
    }
}

pub fn enable_timer_interrupt() {
    unsafe { sie::set_stimer(); }
}

#[no_mangle]
pub fn trap_handler(cx: &mut TrapContext) -> &mut TrapContext {
    let scause = scause::read();
    let stval = stval::read();
    match scause.cause() {
        Trap::Exception(Exception::UserEnvCall) => {
            cx.sepc += 4;
mod.rs [dos] (20:22 12/11/2021)
"mod.rs" [dos] 65L, 1703B
```

In addition, we need to do some initialization work in "main.rs" before the first application executes. We need to add some codes to file **"main.rs"**.

```
mod timer;

trap::enable_timer_interrupt();
timer::set_next_trigger();
```

```
trap::enable_timer_interrupt();
timer::set_next_trigger();
```