Project2 A Simple Kernel 设计文档

中国科学院大学
刘杰
2018.10.29

1. 任务启动与 Context Switching 设计流程

（1）完善 PCB 结构体设计，PCB 包含的信息有：用户态和内核态上下文，用户态和内核态栈顶指针，用于队列操作的两个指针，进程号，存储的光标信息，进程/线程类型，进程/线程状态，优先级，获取锁失败时进入的阻塞队列（用于传参），处于睡眠状态的睡眠时间和开始时间（用于记时，调度时唤醒）

（2）初始化 PCB，除初始化进程号，进程类型，优先级等任务基本信息，还需要给每一个任务分配栈空间，具体实现是给每个任务合适的用户态和内核态栈顶，并将栈顶指针存入上下文中的 sp 寄存器。

在非抢占式调度中，需要将内核态上下文中 ra 寄存器初始化为任务的入口地址，第一次调度时，current_running 从上一个任务指向该任务，恢复现场，然后用 jr ra 指令跳转到该任务入口地址开始执行；在抢占式调度中，需要将用户态 cp0_epc 寄存器初始化为任务入口地址，将内核态 ra 寄存器初始化为一个用于模拟从内核态返回用户态的函数(task_start())的入口地址，第一次调度时，current_running 从上一个任务指向该任务，恢复现场，然后用 jr ra 指令跳转到 task_start()函数的入口地址，task_start 函数首先恢复用户态现场，然后开中断，最后 eret 指令返回用户态，即任务入口地址，开始第一次运行。

另外，需要将初始化 PCB 后的任务根据优先级 push 到相应的就绪队列，并将任务状态改为就绪状态。为防止 os 因为没有任务运行跑死，所以需要初始化一个空任务进行保护，并把 current_running 初始化指向空任务。

（3）context switching 时保存除 zero，k0，k1 寄存器外其余通用寄存器，HI/LO 寄存器，控制寄存器中 CP0_EPC，CP0_STATUS，CP0_BADVADDR.寄存器。保存在 PCB 数组中，在内存静态存储区（.bss 段），使得进程再切换回来后能正常运行

2. 时钟中断、系统调用与 blocking sleep 设计流程

（1）发生例外，硬件自动跳转到例外总入口（0x80000180），关中断然后保存用户态上下文，再根据 CP0_STAUS 寄存器 ExcCode 判断例外类型，如果 ExcCode=0，则为中断，跳转到中断处理函数入口（handle_int()），再根据 CP0_CAUSE 的 IM7-IM0 来区分具体中断类型，如果 IM7 为 1，IM0-IM6 都为 0，则为时钟中断，跳转到时钟中断处理函数（irq_timer()），调用 do_scheduler()完成调度，重置 CP0_COUNT 和 CP0_COMPARE，恢复用户态上下文，开中断，eret 跳转回用户态。

（2）在每次调用 do_scheduler 时，首先检查 sleep 的任务是否达成 wake up 条件，如果 sleep 任务达成唤醒条件，则将其从阻塞队列取出，放入就绪队列再开始调度。

调用 do_sleep()时，将睡眠时间保存在 PCB，同时调用 get_timer()获得睡眠开始时间保存到 PCB，之后在 check_sleep()函数中再调用 get_timer()获得当前时间，将其与 PCB 中初始时间相减，再将结果与 PCB 睡眠时间比较，如果超过则唤醒这任务。

（3）时钟中断和系统调用都属于例外，例外发生后都会跳转到例外总入口处理，关中断，保存用户态上下文，等例外处理完成，再恢复用户态上下文，开中断，eret 指令返回用户态。

不同之处在于例外触发原因，时钟中断是 COUNT 和 COMPARE 相等时触发，sysycall 指令触发系统调用。此外，例外处理具体流程也有不同，在例外总入口根据 CP0_STATUS 跳转到各自例外处理函数，时钟中断跳转到 handle_int，系统调用跳转到 handle_syscall。

3. 基于优先级的调度器设计

（1）在基于优先级的抢占式调度中，优先级定义为 high，medium，low 三级，在初始化 PCB 时给每个任务一个优先级，根据优先级将任务送入不同的就绪队列。在调度时，每次任务切换出来都会降低一个优先级，避免任务长时间占据资源。另外，从阻塞队列出来的任务会先 push 进最高优先级就绪队列，优先运行。

4. Mutex lock 设计流程

（1）自旋锁线程进入隔离区失败会不断尝试，浪费 CPU 资源；互斥锁线程进入隔离区失败会自动挂起到阻塞队列，锁释放前都不会被调度，能节约 CPU 资源。

（2）线程获得互斥锁成功会将标志位置为 LOCKED，访问隔离区直到离开再将标志位设置为 UNLOCKED，同时将该锁对应阻塞队列的任务送入就绪队列。线程获得互斥锁失败会将任务挂起到阻塞队列。

（3）被阻塞的任务在相应锁被释放时从阻塞队列出来，再送入就绪队列等待调度。

5. 关键函数功能

例外处理代码拷贝到总入口

```
static void init_exception()
{
    // 1. Get CP0_STATUS
    // 2. Disable all interrupt
    // 3. Copy the level 2 exception handling code to 0x80000180
    memcpy((void*)0x80000180,exception_handler_entry,exception_handler_end-exception_handler_begin);
    // 4. reset CP0_COMPARE & CP0_COUNT register
    reset_com_and_cnt();
}
```

例外类型判断

```
void exception_helper(uint32_t cause)
{
	uint32_t cause_excode = cause & 0x7c;
	if(cause_excode == 0x00){
		handle_int();
	}
	if(cause_excode == 0x20){
		handle_syscall();
	}
	else{
		handle_other();
	}
}
```

获得时间

```
uint32_t get_timer()
{
	time_elapsed=time_elapsed+2*get_cp0_count()/MHZ;
	return time_elapsed / (1000000);
}
```

时钟中断处理

```
static void irq_timer()
{
	// TODO clock interrupt handler.
	// scheduler, time counter in here to do, emmmmmm maybe.
	//save cursor and print
	save_cursor();
	screen_reflush();
	//get time_elapsed and rst compare&count
	get_ticks();
	reset_com_and_cnt();
	//switch kernel context and restore cursor
	do_scheduler();
	restore_cursor();
}
```