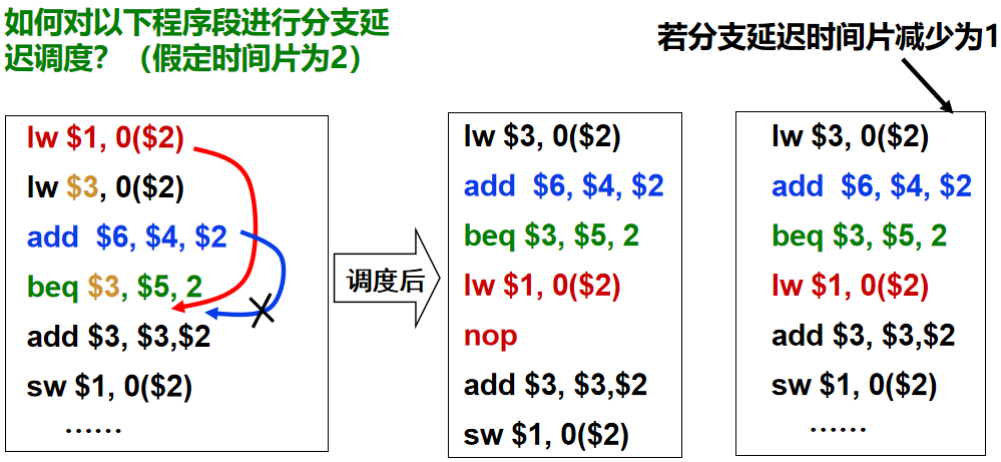
计算机系统设计平时作业六

朱浩泽 1911530

May 11, 2022

1 请描述 PPT 中左侧代码如何通过编译器优化来解决控制冒险的。



以常见的五级流水线为例，如果代码按照顺序执行，不进行调度，分支延迟时间片为 2，则要在 beq 指令之后插入两个 nop，将浪费两个时间片。进行编译器优化之后，将对分支指令没有影响且移动后不会对程序造成影响的指令移动到分支指令后执行，相当于替代一个 nop，在分支指令判断是否进行跳转的同时对该指令进行计算。这样既保证了程序的正确性，又节省了一个时间片的时间。可以看出，若分支延迟时间减少为 1 时，便不需要插入额外的 nop，可以在一定程度上认为通过乱序执行解决了结构冒险的问题。同时我们可以看出，我们选择 lw $1,0($2) 这条指令是因为他对调度到相应位置后对续指令没有影响，但如果调度 add 指令会产生额外的 nop，使整个调度失去意义。

2 请结合代码详述 fork 的由来和作用（勿抄袭网上混乱的讲述）

首先，我们从 Linus 的 github 仓库中找到 linux 的最新代码，地址为 https://github.com/torvalds/linux；找到源代码后，我们进入 kernal/fork.c 文件查看这一部分的代码。

笼统的来说，在 Linux 中 fork 函数的作用就是创建一个子进程作为当前进程的克隆。返回到父节点和子节点。将子 pid 返回给父进程，将 0 返回给子进程。换句话说就是 fork 通过拷贝当前

进程创建一个子进程。子进程与父进程的区别仅仅在于 PID（每个进程唯一）、PPID（父进程的进程号和挂起的信号等某些特定的统计量。fork 具体执行时使用写实拷贝，内核此时并不复制整个进程地址空间，资源的复制只有在需要写人的时候才进行，在此之前，只是以只读方式共享，只有在需要写人的时候，数据才会被复制，从而使各个进程拥有各自的拷贝。

该函数调用 copy_process 函数可以说是完成了大部分的操作，其调用 dup task_ struct 为新进程创建一个内核栈和 thread_info 等，让其与当前进程的值相同，然后将进程描达符内的一些统计信息重新初始化。task_struct 保持不动，调用 copy_flags 以更新 task_struct 的 flags 成员，调用 alloc_pid 为新进程分配一个有效的 PID。根据传递给 clone 的参数标志，拷贝或共享打开的文件、文件系统信息、信号处理函数、进程地址空间和命名空间等信息。到此子进程基本创建完毕。

通过阅读《Linux 内核设计与实现原书第 3 版中文版》可以知道，fork 的思想主要来自 Melvin Conway 的论文 A Multiprocessor System Design

3 关于异常和中断，经常会被提到异常嵌套中断嵌套，请结合 linux 实际代码说明在实际系统中是否存在嵌套？为什么？

我们在 github 仓库中查找这一部分的代码，可以看到其是在 kernel/irq/manage.c 文件中。通过查阅资料我们可以知道，旧版本的 Linux 在申请申请的时候不带 IRQF_DISABLED 标记，则 IRQ HANDLER 里面允许新的其他中断嵌套进来（《Linux 内核设计与实现原书第 3 版中文版》）。然后我们阅读 request_threaded_irq 函数

```
int request_threaded_irq(unsigned int irq, irq_handler_t handler,
          irq_handler_t thread_fn, unsigned long irqflags,
          const char *devname, void *dev_id)
{
   struct irqaction *action;
   struct irq_desc *desc;
   int retval;

   if (irq == IRQ_NOTCONNECTED)
      return -ENOTCONN;
    if (((irqflags & IRQF_SHARED) && !dev_id) ||
      ((irqflags & IRQF_SHARED) && (irqflags & IRQF_NO_AUTOEN)) ||
      (!(irqflags & IRQF_SHARED) && (irqflags & IRQF_COND_SUSPEND)) ||
      ((irqflags & IRQF_NO_SUSPEND) && (irqflags & IRQF_COND_SUSPEND)))
      return -EINVAL;

   desc = irq_to_desc(irq);
   /*
   ''''''''
   ''''''''
```

```
   ''''''''
   */

    if (retval) {
       irq_chip_pm_put(&desc->irq_data);
       kfree(action->secondary);
       kfree(action);
    }

#ifdef CONFIG_DEBUG_SHIRQ_FIXME
    if (!retval && (irqflags & IRQF_SHARED)) {
       /*
        * It's a shared IRQ -- the driver ought to be prepared for it
        * to happen immediately, so let's make sure....
        * We disable the irq to make sure that a 'real' IRQ doesn't
        * run in parallel with our fake.
        */
       unsigned long flags;

       disable_irq(irq);
       local_irq_save(flags);

       handler(irq, dev_id);

       local_irq_restore(flags);
       enable_irq(irq);
    }
#endif
    return retval;
}
```

我们可以看到，IRQF_DISABLED 标记已经消失，中断发生后，在硬件层面上，硬件设备会自动屏蔽 CPU 对中断的响应；软件层面上，设置中断管理器只有处理完一个中断，才能开始处理下一个中断，所以是不允许进行中断嵌套的。

异常一般也是不嵌套的，有一个东西叫 tripple fault，就是说如果异常里再异常，有个 double fault handler 会试图处理，double fault 里再 fault，就被认为是无法恢复了，就成为 tripple fault，只能关机了。