操作系统原理课程实验报告

姓　　名：　　孙溪

学　　院：　　计算机科学与技术学院

专　　业：　　计算机科学与技术

班　　级：　　CS2007

学　　号：　　U202015512

指导教师：　　谢美意

分数	
教师签名	

2022 年 12 月 31 日

目　录

实验一 外部中断

1.1 实验目的

完善 PKE 操作系统的时钟中断处理，执行 obj/app_long_loop 时获得预期输出。

1.2 实验内容

在这个实验中，我们遇到的时钟中断也是在机器模式下触发的。因为 S 模式下的处理函数 handle_mtimer_trap 还没有完成。handle_mtimer_trap 需要完成以下操作：首先，我们可以看到在这个函数的上方定义了一个全局变量 g_ticks，用它来记录时钟中断的次数。在第 33 行，会输出这个计数。为了确保系统能正常运行，这个计数应该每次都会增加 1。所以，handle_mtimer_trap 需要首先将 g_ticks 加 1。其次，由于在处理完中断后，SIP（Supervisor Interrupt Pending，即 S 模式的中断等待寄存器）寄存器中的 SIP_SSIP 位仍然为 1（由 M 模式的中断处理函数设置），如果这个位置一直是 1，会导致模拟的 RISC-V 机器一直处于中断状态。所以，handle_mtimer_trap 还需要将 SIP 的 SIP_SSIP 位设置为 0，以便在下一次发生时钟中断时，M 模式的函数能够将该位置 1，以触发 S 模式的下一次中断。核心代码如图 1.3 所示：

```
//
// global variable that store the recorded "ticks". added @lab1_3
static uint64 g_ticks = 0;
//
// added @lab1_3
//
void handle_mtimer_trap() {
  sprint("Ticks %d\n", g_ticks);
  // TODO (lab1_3): increase g_ticks to record this "tick", and then clear the
"SIP"
  // field in sip register.
  // hint: use write_csr to disable the SIP_SSIP bit in sip.
  // panic( "lab1_3: increase g_ticks by one, and clear SIP field in sip
register.\n" );
  g_ticks++;
  write_csr(sip, 0);
}
```

图 1.3 时钟中断处理完善

1.3 实验调试及心得

学习到了操作系统的中断处理机制，包括一些细节上的处理，保证每一次中断的产生都能有效地被回应。

实验二 复杂缺页异常

1.1 实验目的

通过修改 PKE 的内核代码，能够处理不同情况的缺页异常。

1.2 实验内容

测试程序会对给定的 n 计算 0 到 n 的和，每一步的递归结果保存在 ans 数组。函数调用时，在递归执行时会触发用户栈的缺页，需要对其进行正确处理，确保程序正确运行；访问数组越界地址，由于该处虚拟地址尚未有对应的物理地址映射，因此属于非法地址的访问，对于这种缺页异常，提示用户并退出程序执行。

实验思路：为了解决这些问题，我们需要在 handle_user_page_fault 函数中添加一个判断，判断当前访问的地址是否在合法的能进行分配的地址空间内。如果是，则分配一个新的页面；否则提示用户并退出程序。关键在于利用 sp 寄存器存储的栈顶地址来判断地址是否越界，关键核心代码如图 2.1 所示：

```
void handle_user_page_fault(uint64 mcause, uint64 sepc, uint64 stval) {
  sprint("handle_page_fault: %lx\n", stval);
  switch (mcause) {
    case CAUSE_STORE_PAGE_FAULT:
    if (stval + PGSIZE < current->trapframe->regs.sp) {
      panic("this address is not available!");
    }else {
      map_pages(current->pagetable, ROUNDDOWN(stval,PGSIZE), PGSIZE, (uint64)
alloc_page(), prot_to_type(PROT_READ|PROT_WRITE, 1));
    }
      break;
    default:
      sprint("unknown page fault.\n");
      break;
  }
}
```

图 2.1 复杂缺页异常

1.3 实验调试及心得

挑战实验的思路比较清晰，只需掌握好栈的结构和操作系统的栈管理方式就可以进行相应的程序编写。因为异常条件比较容易判断和处理，只需要进行条件区分就可以完成，没有繁杂的调试过程。

实验三 循环轮转调度

1.1 实验目的

实现 kernel/strap.c 文件中的 rrsched()函数，使得执行 obj/app_two_long_loops 时获得预期结果。

1.2 实验内容

修改 PKE 内核并实现循环轮转调度算法，以使用指定的“时间片”长度来调度进程。在实现时，在 rrsched()函数中完善当前进程的 tick_count 加 1 后是否大于等于 TIME_SLICE_LEN 的判断逻辑，并根据结果执行相应的操作。如果 tick_count 加 1 后大于等于 TIME_SLICE_LEN，则应将当前进程的 tick_count 清零，并将当前进程加入就绪队列，然后进行转进程调度。如果 tick_count 加 1 后仍然小于 TIME_SLICE_LEN，则应将 tick_count 加 1 并返回。代码如图 3.1 所示：

```
void rrsched() {
  // TODO (lab3_3): implements round-robin scheduling.
  // hint: increase the tick_count member of current process by one, if it is
bigger than
  // TIME_SLICE_LEN (means it has consumed its time slice), change its status
into READY,
  // place it in the rear of ready queue, and finally schedule next process to
run.
  if( current->tick_count + 1 >= TIME_SLICE_LEN ){
      current->tick_count = 0;
      current->status = READY;
      insert_to_ready_queue( current );
      schedule();
    }else{
      current->tick_count ++;
    }

}
```

图 3.3 rrsched 内容完善

1.3 实验调试及心得

明晰好调用关系按照相关流程，通过利用时钟中断来实现进程的循环轮转调度，避免由于一个进程的执行体过长，导致系统中其他进程无法得到调度的问题。