考试科目名称 操作系统原理与实践 II （A 卷）

考试方式：闭卷 考试日期：2008 年 月 日 教师：

系（专业）： 年级： 班级：

学号： 姓名： 成绩：

题号	一	二	三	四	五	六
分数						

一、名词解释（本题满分 20 分）

1、 孤儿进程

父进程退出后还在继续运行的子进程称为孤儿进程。孤儿进程将被 init 进程(进程号为 1)所收养，并由 init 进程对它们完成状态收集工作。

2、 可靠信号

信号支持排队，不会丢失，其信号值位于信 SIGRTMIN 和 SIGRTMAX 之间。

3、 网络字节顺序

在网络中传输的多字节数的顺序，通常是高字节数据存放在低地址处，低字节数据存放在高地址处。它可能与数据在内存中的存储顺序不一致。

4、 封装例程

屏蔽底层复杂性，将系统调用封装成应用程序能够直接调用的函数(库函数)，供用户在用户空间编程使用。

5、 工作队列

是另外一种将工作推后执行的形式，其核心思想是将推后工作交由一个内核线程去执行，允许被重新调度甚至是睡眠。因此，通过工作队列执行的代码能占尽进程上下文的所有优势。

二、计算题（本题满分 15 分）

某基于 i386 体系结构的逻辑地址的段标志符（即段选择子）为 0x0010，其段内偏移为 0x00000005，CR3 的地址信息为 0x00102000：

A） 试根据下述 GDT 及 LDT 信息，计算出线性地址。

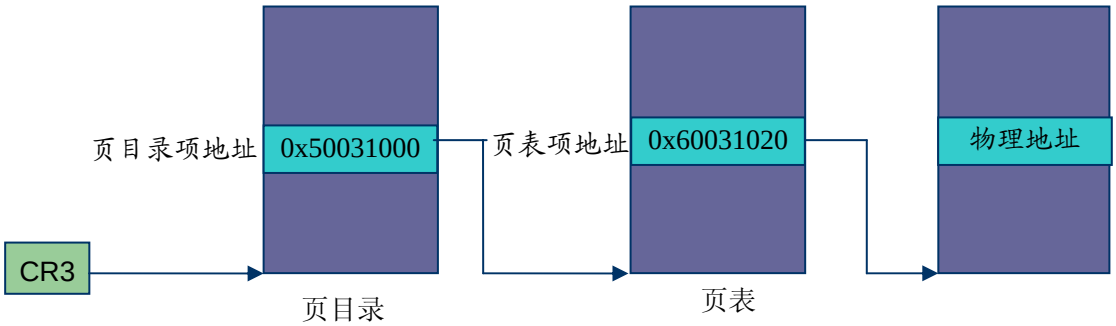
B）试结合下图中的页目录及页表信息，计算出与该线性地址对应的页目录项地址、页表项地址及物理地址。

INDEX	基地址
0	NULL
1	0x20021400
2	0x20021401
3	0x20021402
4	0x20021403
5	0x20021404
6	0x20021405

GDT 中各段描述符的基地址信息

INDEX	基地址
0	NULL
1	0x30021401
2	0x30021402
3	0x30021403
4	0x30021404
5	0x30021405
6	0x30021406

LDT 中各段描述符的基地址信息



段选择子的二进制形式：0000 0000 0001 0000，可得 T = 0，索引值为 2。因此，应从 GDT 中获取基地址信息，基地址值为 0x20021401。

线性地址为 0x20021401+0x00000005 = 0x20021406，其二进制格式为

0010 0000 0000 0010 0001 0100 0000 0110

其页目录索引为 0x80，页表索引为 0x21，页内偏移是 0x406。

页目录项地址：0x00102000 + 0x80*4 = 0x00102200。

页表项地址：0x50031000 + 0x21*4 = 0x 50051084

物理地址：0x60031020 + 0x406 = 0x60031426

三、简答题（本题满分 35 分）

1、 试述 Linux 2.4 与 Linux 2.6 进程调度体系结构的主要差别，并论述 Linux 2.6 O(1)调度算法结构及执行机制。

1）Linux2.4 所有就绪进程存在一个以 runqueue_head 为表头的全局进程队列中，调度器从中选取最适合调度的进程投入运行。整个队列由一个读/写自旋锁保护，多个处理器可以并行访问，但写操作必须互斥访问。 时间片重算算法是在所有的进程都用尽它们的时间片以后才重新计算。

2)Linux 2.6 的每个处理器有两个数组，活动就绪进程队列数组和不活跃就绪进程队列数组。如果一个进程消耗完了它的“时间片”，就进入不活跃就绪进程数组的相应队列的队尾。当所有的进程都“耗尽”了它的“时间片”后，交换活跃与不活跃就绪进程队列数组，因此，不需要任何其他的开销。

主要差别：Linux 2.4 的调度体系基于共享全局队列，其算法属于 O(n)，开销是线性增长的。而 Linux 2.6 每个处理器都有独立的就绪进程队列，各个处理器可以并行运行调度程序来挑选进程运行，不同处理器上的进程可以完全并行地休眠、唤醒和上下文切换。

O(1)调度算法结构：每个数组中有 140 个就绪进程队列(runqueue)，每个队列对应于 140 个优先级的某一个。通过位图标记队列状态，调度时，通过 find_first_bit 找到第一个不为空

的队列，并取队首的进程即可。不管队列中有多少个就绪进程，挑选就绪程的速度是一定的。

2、 试述基于 IRQ 线共享的中断处理实现机制。

Linux 通过 struct irq_desc_t 及 struct irqaction 持 IRQ 线共享。其中 irq_desc_t 用于描述中断源，其成员 action 为指针结构，该成员可将共享同一个中断号的多个设备连接起来。Irqaction 是一个链表结构，包含内核接收到特定 IRQ 后应采取的操作，所有共享一个 irq 的链接在一个 action 表。其成员 dev_id 用来记录由制造厂商定义的设备 ID，从而实现对不同中断设备的识别。

3、 试述消息队列、共享主存及信号量通信机制的特点及在应用场景上的主要差别。

消息队列是一个格式化的可变长信息单位，允许一个进程向任何其他进程发送一个消息。当一个进程收到多个消息时，可将它们排成一个消息队列。消息队列以异步方式为通信频繁、但数据量少的进程通信提供服务。

共享主存的核心思想是让多个进程共享的一块内存区域，不同进程可把共享内存映射到自己的一块地址空间，处于共享内存区的进程对该区域的操作是互见的。共享内存主要用于为数据量大的进程间通信提供服务。

信号量是具有整数值的对象，表示可用资源的数量。申请资源时减 1，资源不足时进程可以睡眠等待、也可以立即返回。信号量主要用于实现与其他进程同步和互斥。

4、 试述 Linux 中的线程实现机制的特点，并说明如何区别内核线程与普通进程。

从内核角度，Linux 中没有线程概念内核没有针对所谓线程的调度算法或数据结构来表征线程，把所有线程当进程来实现。线程被视为与其他进程共享某些资源的进程，都有自己的进程描述符 task_struct。在 linux 中“线程”，仅仅是表示多个进程共享资源的一种说法。

内核线程与普通进程的区分主要通过进程描述符中的 mm 和 active_mm 来实现。 对普通进程而言，这两个字段存放相同的指针。对内核线程而言，由于不拥有任何内核描述符，因此，mm 字段总为 NULL，当内核线程运行时，active_mm 被初始化为前一个运行进程的 active_mm 的值。

5、 试述 Linux 内存管理中的伙伴系统实现机制。

系统将所有空闲页框分组为 10 个块链表，每个块链表分别包含大小为 1，2，4，8，16，32，64，128，256 和 512 个连续的页框，每个块的第一个页框的物理地址是该块大小的整数倍。Linux 为每个 zone 使用独立的伙伴系统，每个伙伴系统使用的主要数据结构包括：

1） 页描述符数组 zone_mem_map，每个管理区都关系到 mem_map 的一个子集；

2） 结构为 free_area_t 的空闲内存管理数组 free_area，数组长度为 10；

伙伴算法每次分配包含一个或者多个物理页面的内存块，页面以 2 的次幂的内存块来分配。首先搜寻满足请求大小的页面，它从满足当前申请大小的 Buddy 数据结构的 free_area 域着手沿链来搜索空闲页面。如果没有这样请求大小的空闲页面，则它搜索两倍于请求大小的内存块。这个过程一直将持续到所有的 Buddy 被搜索完或找到满足要求的内存块为。如果找到的页面块大于请求的块则对其进行分割以使其大小与请求块匹配。由于块大小都是 2 的次幂所以分割过程十分简单：空闲块（低地址）被连进相应大小的队列而这个页面块被分配给调用者。 释放时，先检查该块的伙伴的使用情况，如果没有被使用，则合并这对 Buddy，再检查该两倍大小的块的 Buddy，一直持续到合并后的块没有 Buddy 为止，并将之链到该大小的 free_area 队列中。

得分	

四、程序分析题（本题满分 15 分）

结合__do_softirq()及 ksoftirqd 内核线程的核心代码，分析软中断的处理机制。

__do_softirq()核心代码

```
#define MAX_SOFTIRQ_RESTART 10
asmlinkage void __do_softirq(void){
  struct softirq_action *h;
  __u32 pending;
  int max_restart = MAX_SOFTIRQ_RESTART;
  int cpu;
  pending = local_softirq_pending();
  account_system_vtime(current);
// 屏蔽其他软中断，使每个CPU上只能同时运行一个软中断
  __local_bh_disable((unsigned long)__builtin_return_address(0));
  trace_softirq_enter();
  // 针对 SMP 得到当前正在处理的 CPU
  cpu = smp_processor_id();
restart:
  set_softirq_pending(0);
  // 此时才开中断运行，以前运行状态一直是关中断运行
 //这时当前处理软中断才可能被硬件中断抢占。
  local_irq_enable();
  // 以下代码可被硬件中断抢占，但无法注册相应软中断
  h = softirq_vec;

  do {
    if (pending & 1) {
      h->action(h);
      rcu_bh_qsctr_inc(cpu);
    }
    h++;
    pending >>= 1;
  } while (pending);
  // 以下代码执行过程中硬件中断无法抢占。
  local_irq_disable();

  // 因为刚才开中断执行过程中可能多次被硬件中断抢占，每抢占一次就有可
 // 能注册一个软中断，所以要再重新取一次所有的软中断。
  pending = local_softirq_pending();
  if (pending && --max_restart)
    goto restart;
  // 如果以上步骤重复了 10 次后还有 pending 的软中断的话，调用 ksoftirqd
  if (pending)
    wakeup_softirqd();
  trace_softirq_exit();
  account_system_vtime(current);
  _local_bh_enable();
}
```

ksoftirqd 核心代码

```
static int ksoftirqd(void * __bind_cpu){
  set_user_nice(current, 19);
  // 设置当前进程不允许被挂启
  current->flags |= PF_NOFREEZE;
  set_current_state(TASK_INTERRUPTIBLE);
  while (!kthread_should_stop()) {
  //若可处理，在此处理期间内禁止当前进程被抢占
    preempt_disable();
 // 首先判断系统当前没有需要处理的 pending 状态的软中断
    if (!local_softirq_pending()) {
      // 没有的话在主动放弃 CPU 前先要允许抢占
      preempt_enable_no_resched();
       //将当前进程放入睡眠队列，并切换新的进程执行
      schedule();
      // 再次被调度时，将从调用这个函数的下一条语句开始执行
      preempt_disable();
    }
    __set_current_state(TASK_RUNNING);
    while (local_softirq_pending()) {
      if (cpu_is_offline((long)__bind_cpu))
    // 如果当前被关联的 CPU 无法继续处理则跳转
    // 到 wait_to_die 标记出，等待结束并退出。
        goto wait_to_die;
      do_softirq();
      // 允许当前进程被抢占。
      preempt_enable_no_resched();
      // 该函数有可能间接的调用 schedule() 来切换当前进程
      cond_resched();
      preempt_disable();
    }
    preempt_enable();
    set_current_state(TASK_INTERRUPTIBLE);
  }
  // 如果将会停止则设置当前进程为运行状态后直接返回。
  __set_current_state(TASK_RUNNING);
  return 0;
wait_to_die:
   preempt_enable();
   set_current_state(TASK_INTERRUPTIBLE);
   while (!kthread_should_stop()) {
     schedule();
     set_current_state(TASK_INTERRUPTIBLE);
   }
   __set_current_state(TASK_RUNNING);
   return 0;
}
```

__do_softirq()：固定每次的循环次数，然后就返回；其余挂起的软中断在内核线程 ksoftirqd 中执行。

ksoftirqd 内核线程：被唤醒时，检查 local_softirq_pending 中的软中断并在必要时调用 do_softirq()。如果没有挂起的软中断，把当前进程状态设置为休眠，随后若当前进程需要，就调用 cond_resched()函数实现进程切换。

得分	

五、程序设计题（本题满分 15 分）

基于 TCP/IP socket 网络编程技术，采用 C 语言，实现一个支持多用户并发执行的客户/服务器程序，客户及服务器程序的功能描述如下：

1）服务程序的服务端口为 1234，该程序实现浮点数加减法运算。每当接收到一个用户服务连接请求后，服务程序将为该用户创建一个线程，并通过该线程响应用户服务请求。

a) 该线程不断接收服务请求，根据请求类型（加、减法）执行相应计算，并将计算结果返回给用户。

b) 当用户输入“exit”时，服务程序将断开与当前用户的连接，并退出相应服务线程。

2）客户程序通过端口 1234 与服务器连接，连接成功后将在客户端显示“连接成功”信息，并可通过以下消息格式向服务器程序发送服务请求（其中 op1 及 op2 为操作数）：

a) 加法运算：op1 + op2

b) 减法运算：op1 – op2

c) 退出服务：exit

请根据上述功能描述，给出客户及服务器代码。

说明：程序实现中可能涉及到的字符串处理函数如下：

a）*strchr(char *str, char c); //在一个串中查找给定字符的第一个匹配之处

b）int atof(const char *nptr); //将字符串转换成浮点数