E06：进程控制（UNIX 时钟中断与系统调用）

一、单项选择题

1. 用户要在程序一级获得系统帮助，必须通过__D__。
   A. 进程调度　B. 作业调度　C. 键盘命令　D. 系统调用

2. 以下关于一般函数调用与系统调用的表述中，错误的是：__B__。
   A. 一般函数调用使用 call 指令，系统调用使用 INT 指令。
   B. 二者都通过向当前堆栈压入栈帧传递参数。
   C. 系统调用的调用函数和被调用函数分别链接在两个不同的可执行文件中。
   D. 一般函数调用中的调用函数和被调用函数链接在同一个可执行文件中。

3. 下列关于系统调用的叙述中，正确的是：__C__。
   Ⅰ. 在执行系统调用服务程序的过程中，CPU 处于内核态
   Ⅱ. 操作系统通过提供系统调用避免用户程序直接访问外设
   Ⅲ. 不同的操作系统为应用程序提供了统一的系统调用接口
   Ⅳ. 系统调用是操作系统内核为应用程序提供服务的接口
   A. 仅Ⅰ、Ⅳ　B. 仅Ⅱ、Ⅲ
   C. 仅Ⅰ、Ⅱ、Ⅳ　D. 仅Ⅰ、Ⅲ、Ⅳ

4. 异常是指令执行过程中在处理器内部发生的特殊事件，中断是来自处理器外部的请求事件。下列关于中断或异常情况的叙述中，错误的是：__A__。
   A. “访存时缺页” 属于中断
   B. “整数除以 0” 属于异常
   C. “DMA 传送结束” 属于中断　（直接内存访问）
   D. “存储保护错” 属于异常

5. 执行系统调用的过程包括如下主要操作：
   ①返回用户态　②执行陷入(trap)指令
   ③传递系统调用参数　④执行相应的服务程序
   正确的执行顺序是：__A__。
   A. ②→③→④→①　B. ①→②→④→③
   C. ④→③→②→①　D. ③→②→④→①

D，先传递系统调用参数再执行trap

6.p_cpu反映了进程占用处理机的程度，对于其值的修改，首先是每一次时钟中断，现运行进程p_cpu+1,而每一秒结束时，检查所有进程的p_cpu，如果其值没有超过调度魔数，则将其值置零，否则减去其调度模数。

结果是：连续占用处理机较长时间的进程，p_cpu值越来越大，p_pri增大，优先权降低，在进程切换调度的时候被选中的机会越来越小；而在较长时间内未使用处理机或者虽然频繁使用处理机，但每次使用时间都很短的进程，进程优先数降低，优先权变大，更容易在下一次进程切换调度时被选中。

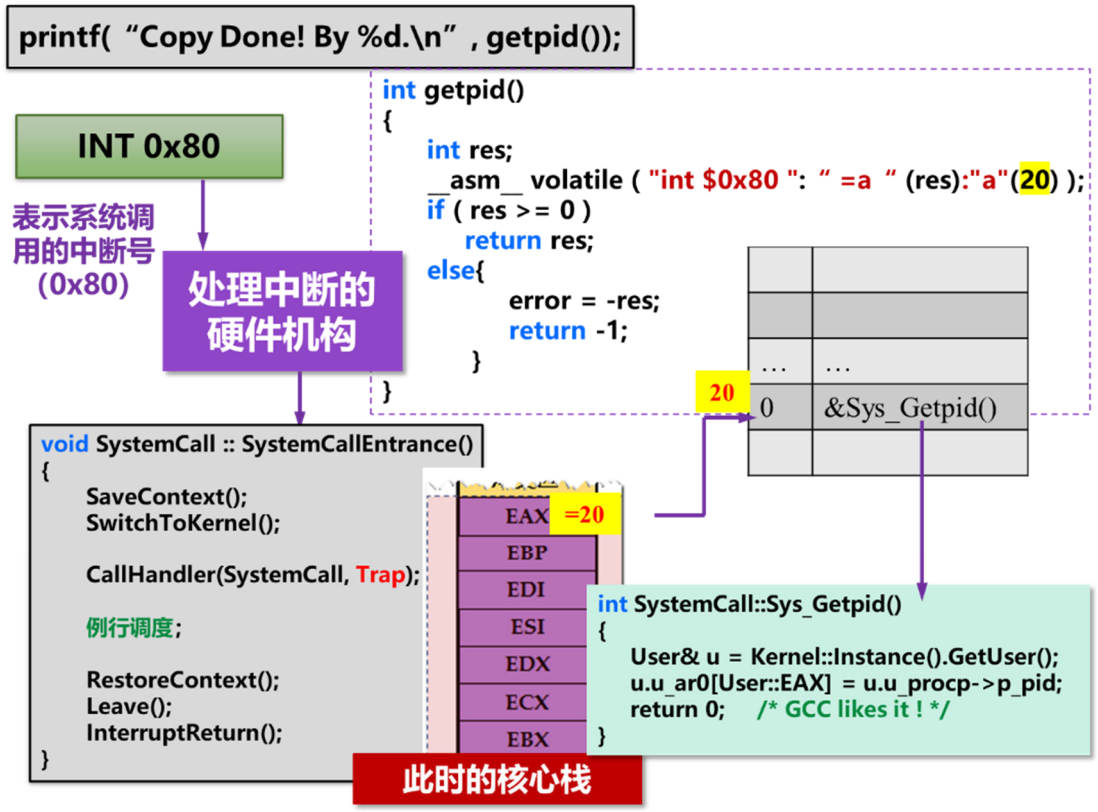
二、简答题：

6. 请说明 UNIX V6++中 p_cpu 这个变量是如何被修改的？对其修改的最终结果是怎样的？
7. 请写出 UNIX V6++中几个重新计算进程优先数的时机。

1. 系统调用处理程序即将结束时，在Trap（）函数末尾计算先运行进程优先数
2. 每秒结束时重算进程优先数
3. 每秒结束时，计算所有p_pri大于PUSER的进程的优先数

三、应用分析题

8. 参照下图，尽量详细的描述下述语句：printf(“Copy Done! By %d.\n”, getpid())中，getpid()的执行过程。



用户在用户程序中调用了库函数getpid()，其作用为获得进程标识数。
函数中有一个内联汇编语句，把系统调用号20送入EAX寄存器，然后引发INT 80指令，最后系统调用的返回值由EAX寄存器保留在res变量中。
执行系统调用的过程如下：
由硬件执行中断处理，现运行进程进入核心态，执行系统调用入口程序SystemCall::System CallEntrance，然后调用系统调用处理程序Trap,然后根据EAX寄存器中的系统调用号在系统调用子程序入口表中查询相应服务子程序的参数和入口地址，得到子程序SystemCall:: Sys_Getpid()，在该函数中访问了User对象然后在User结构中找到了u_procp即进程proc表的地址，然后访问了proc表中p_pid的值得到进程标识数，然后使用EAX寄存器将其返回值带回用户态。系统调用结束，进程返回用户态。

中断返回后，将EAX寄存器的返回值赋值给res，如果res>=0,返回res，否则返回-1并设置出错码
然后getpid()返回

cpu执行中断隐指令，
1.关中断
2.装入中断向量，根据中断号80H，查询IDT表，获取中断向量。将其中的Segment Selector 装入CS，使CPU 转入核心态；Offset 指向的系统调用入口程序SystemCall:: SystemCallEntrance()的入口地址装入EIP，以实现程序跳转。
3. 将CS，EIP，EFLAGES 等寄存器的值压入现运行进程核心栈，形成硬件现场。

执行系统调用入口程序SystemCall::SystemCallEntrance()，完成如下工作：
3.1. 宏SaveContext()继续保存中断现场，形成软件现场如图5 所示。
3.2. 宏SwithToKernel()完成对DS，ES，SS 的赋值，指向核心态数据段描述符。
3.3. 调用中断处理程序SystemCall::Trap(struct pt_regs* regs, struct pt_context* context)，核心栈变化如图6 所示（由于此处为汇编指令call 调用了C 语言函数，所以栈帧中包含返回地址，Old Ebp 和局部变量区）。转向步骤4。
3.4. 例行调度：因为系统调用前为用户态，检查RunRun 的标志位是否为0。如果不为0，进行进程的切换调度。否则，
3.5. 宏RestoreContext()弹出核心栈中的软件现场，恢复CPU 中各个寄存器的值。核心栈回到图4 所示的状态。
3.6. 宏Leave()删除核心栈中的中断入口程序的栈帧。核心栈回到图3 所示的状态。
3.7. 宏InterruptReturn()执行中断返回指令，删除核心栈中的硬件现场。CPU 返回用户态。