


中断、异常和系统调用比较 随堂练习

单选题 1

(2012统考)下列选项中，不可能在用户态发生的是 ()


- ☐ 系统调用
- ☐ 外部中断
- ☒ 进程切换 
- ☐ 缺页

EXPLANATION

系统调用是提供给应用程序使用的，由用户态发出，进入内核态执行。外部中断随时可能发生；应用程序执行时可能发生缺页；进程切换完全由内核来控制。

单选题 2

(2012统考) 中断处理和子程序调用都需要压栈以保护现场。中断处理一定会保存而子程序调用不需要保存其内容的是 ()


- ☐ 程序计数器
- ☒ 程序状态字寄存器 
- ☐ 通用数据寄存器
- ☐ 通用地址寄存器

EXPLANATION

程序状态字 (PSW) 寄存器用于记录当前处理器的状态和控制指令的执行顺序，并且保留与运行程序相关的各种信息，主要作用是实现程序状态的保护和恢复。所以中断处理程序要将PSW保存，子程序调用在进程内部执行，不会更改PSW。


单选题 3

(华中科技大学) 中断向量地址是 ()

- ☐ 子程序入口地址
- ☒ 中断服务例程入口地址 
- ☐ 中断服务例程入口地址的地址
- ☐ 例行程序入口地址

单选题 4

下列选项中， ____可以执行特权指令？ ()


- ☒ 中断处理例程 
- ☐ 普通用户的程序
- ☐ 通用库函数
- ☐ 管理员用户的程序

EXPLANATION

中断处理例程（也可称为中断处理程序）需要执行打开中断，关闭中断等特权指令，而这些指令只能在内核态下才能正确执行，所以中断处理例程位于操作系统内核中。而1,3,4都属于用户程序和用于用户程序的程序库。以ucore OS为例，在lab1中就涉及了中断处理例程，可查看intr_enable，sti，trap等函数完成了啥事情？被谁调用了？

单选题 5

一般来讲，中断来源于 ____ ()


- ☒ 外部设备 
- ☐ 应用程序主动行为
- ☐ 操作系统主动行为
- ☐ 软件故障

EXPLANATION

中断来源与外部设备，外部设备通过中断来通知CPU与外设相关的各种事件。第2选项如表示是应用程序向操作系统发出的主动行为，应该算是系统调用请求。第4选项说的软件故障也可称为软件异常，比如除零错等。以ucore OS为例，外设产生的中断典型的是时钟中断、键盘中断、串口中断。在lab1中，具体的中断处理例程在trap.c文件中的trap_dispatch函数中有对应的实现。对软件故障/异常的处理也在trap_dispatch函数中的相关case default的具体实现中完成。在lab1的challenge练习中和lab5中，有具体的系统调用的设计与实现。

单选题 6

用户程序通过____向操作系统提出访问外部设备的请求 ()


- ☐ I/O指令
- ☒ 系统调用 
- ☐ 中断
- ☐ 创建新的进程

EXPLANATION

具体内容可参见10.的回答。以ucore OS为例，在lab5中有详细的syscall机制的设计实现。比如用户执行显示输出一个字符的操作，由于涉及向屏幕和串口等外设输出字符，需要向操作系统发出请求，具体过程是应用程序运行在用户态，通过用户程序库函数cputch，会调用sys_putc函数，并进一步调用syscall函数（在usr/libs/syscall.c文件中），而这个函数会执行“int 0x80”来发出系统调用请求。在ucore OS内核中，会接收到这个系统调用号（0x80）的中断（参见 kernel/trap/trap.c中的trap_dispatch函数有关“case T_SYSCALL:”的实现），并进一步调用内核syscall函数（参见 kernel/syscall/syscall.c中的实现）来完成用户的请求。内核在内核态（也称特权态）完成后，通过执行“iret”指令（kernel/trap/trapentry.S中的“__trapret:”下面的指令），返回到用户态应用程序发出系统调用的下一条指令继续执行应用程序。

单选题 7

应用程序引发异常的时候，操作系统可能的反应是 ()

- ☐ 删除磁盘上的应用程序
- ☐ 重启应用程序
- ☒ 杀死应用程序 
- ☐ 修复应用程序中的错误

EXPLANATION

更合适的答案是3。因为应用程序发生异常说明应用程序有错误或bug，如果应用程序无法应对这样的错误，这时再进一步执行应用程序意义不大。如果应用程序可以应对这样的错误（比如基于当前c++或java的提供的异常处理机制，或者基于操作系统的信号（signal）机制（后续章节“进程间通信”会涉及）），则操作系统会让应用程序转到应用程序的对应处理函数来完成后续的修补工作。以ucore OS为例，目前的ucore实现在应对应用程序异常时做的更加剧烈一些。在lab5中有对对用户态应用程序访问内存产生错误异常的处理（参见 kernel/trap/trap.c中的trap_dispatch函数有关“case T_PGFLT:”的实现），即ucore判断用户态程序在运行过程中发生了内存访问错误异常，这是ucore认为重点是查找错误，所以会调用panic函数，进入kernel的监控器子系统，便于开发者查找和发现问题。这样ucore也就不再做正常工作了。当然，我们可以简单修改ucore当前的实现，不进入内核监控器，而是直接杀死进程即可。你能完成这个修改吗？

多选题 1

操作系统处理中断的流程包括____ ()

- ☒ 保护当前正在运行程序的现场
- ☒ 分析是何种中断，以便转去执行相应的中断处理程序
- ☒ 执行相应的中断处理程序
- ☒ 恢复被中断程序的现场

EXPLANATION

中断是异步产生的，会随时打断应用程序的执行，且在操作系统的管理之下，应用程序感知不到中断的产生。所以操作系统需要保存被打断的应用程序的执行现场，处理具体的中断，然后恢复被打断的应用程序的执行现场，使得应用程序可以继续执行。以ucore OS为例（lab5实验），产生一个中断XX后，操作系统的执行过程如下：vectorXX(vectors.S)--> _alltraps(trapentry.S)-->trap(trap.c)-->trap_dispatch(trap.c)-->.....具体的中断处理->_trapret(trapentry.S) 通过查看上述函数的源码，可以对应到答案1-4。另外，需要注意，在ucore中，应用程序的执行现场其实保存在trapframe数据结构中。

多选题 2

下列程序工作在内核态的有____()

- ☒ 系统调用的处理程序
- ☒ 中断处理程序
- ☒ 进程调度
- ☒ 内存管理

EXPLANATION

这里说的“程序”是一种指称，其实就是一些功能的代码实现。而1-4都是操作系统的主要功能，需要执行相关的特权指令，所以工作在内核态。以ucore OS为例（lab5实验），系统调用的处理程序在kern/syscall目录下，中断处理程序在kern/trap目录下，进程调度在kern/schedule目录下，内存管理在kern/mm目录下

系统调用

单选题 1

(西北工业大学)CPU执行操作系统代码的时候称为处理机处于 ()

- ☐ 自由态
- ☐ 目态
- ☒ 管态
- ☐ 就绪态

EXPLANATION

内核态又称为管态

cpu工作状态分为**系统态（或称管理态，管态）**和**用户态（或称目态）**。引入这两个工作状态的原因是：为了避免用户程序错误地使用特权指令，保护操作系统不被用户程序破坏。具体规定为：当cpu处于用户态时，不允许执行特权指令，当cpu处于系统态时，可执行包括特权指令在内的一切机器指令。

单选题 2

（2013统考）下列选项中，会导致用户进程从用户态切换到内核态的操作是（ ） 1) 整数除以0 2) sin()函数调用 3) read系统调用


- ☐ 1、2
- ☒ 1、3 
- ☐ 2、3
- ☐ 1、2、3

EXPLANATION

函数调用并不会切换到内核态，而除零操作引发中断，中断和系统调用都会切换到内核态进行相应处理。

单选题 3

系统调用的主要作用是（ ）

- ☐ 处理硬件问题
- ☐ 应对软件异常
- ☒ 给应用程序提供服务接口 
- ☐ 管理应用程序

EXPLANATION

应用程序一般无法直接访问硬件，也无法执行特权指令。所以，需要通过操作系统来间接完成相关的工作。而基于安全性和可靠性的需求，应用程序运行在用户态，操作系统内核运行在内核态，导致应用程序无法通过函数调用来访问操作系统提供的各种服务，于是通过系统调用的方式就成了应用程序向OS发出请求并获得服务反馈的唯一通道和接口。以ucore OS为例，在lab1的challenge练习中和lab5中，系统调用机制的初始化也是通过建立中断向量表来完成的（可查看lab1的challenge的答案中在trap.c中idt_init函数的实现），中断向量表描述了但应用程序产生一个用于系统调用的中断号时，对应的中断服务例程的具体虚拟地址在哪里，即建立了系统调用的中断号和中断服务例程的对应关系。这样当应用程序发出类似“int 0x80”这样的指令时（可查看lab1的challenge的答案中在init.c中lab1_switch_to_kernel函数的实现），操作系统的中断服务例程会被调用，并完成相应的服务（可查看lab1的challenge的答案中在trap.c中trap_dispatch函数有关“case T_SWITCH_TOK:”的实现）。

单选题 4

下列关于系统调用的说法错误的是（）

- ☐ 系统调用一般有对应的库函数
- ☒ 应用程序可以不通过系统调用来直接获得操作系统的服务 ✓
- ☐ 应用程序一般使用更高层的库函数而不是直接使用系统调用
- ☐ 系统调用可能执行失败

EXPLANATION

更合适的答案是2。根据对当前操作系统设计与实现的理解，系统调用是应用程序向操作系统发出服务请求并获得操作系统服务的唯一通道和结果。如果操作系统在执行系统调用服务时，产生了错误，就会导致系统调用执行失败。以ucore OS为例，在用户态的应用程序（lab5,6,7,8中的应用程序）都是通过系统调用来获得操作系统的服务的。为了简化应用程序发出系统调用请求，ucore OS提供了用户态的更高层次的库函数（user/libs/ulib.[ch]和syscall.[ch]），简化了应用程序的编写。如果操作系统在执行系统调用服务时，产生了错误，就会导致系统调用执行失败。

单选题 5

以下关于系统调用和常规调用的说法中，错误的是（）

- ☐ 系统调用一般比常规函数调用的执行开销大
- ☐ 系统调用需要切换堆栈
- ☐ 系统调用可以引起特权级的变化
- ☒ 常规函数调用和系统调用都在内核态执行 ✓

EXPLANATION

系统调用相对常规函数调用执行开销要大，因为这会涉及到用户态栈和内核态栈的切换开销，特权级变化带来的开销，以及操作系统对用户态程序传来的参数安全性检查等开销。如果发出请求的请求方和应答请求的应答方都在内核态执行，则不用考虑安全问题了，效率还是需要的，直接用常规函数调用就够了。以ucore OS为例，我们可以看到系统调用的开销在执行“int 0x80”和“iret”带来的用户态栈和内核态栈的切换开销，两种特权级切换带来的执行状态（关注 kern/trap/trap.h中的trapframe数据结构）的保存与恢复等（可参看 kern/trap/trapentry.S的__alltraps和__trapret的实现）。而函数调用使用的是“call”和“ret”指令，只有一个栈，不涉及特权级转变带来的各种开销。如要了解call, ret, int和iret指令的具体功能和实现，可查看“英特尔 64 和 iA-32 架构软件开发人员手册卷 2a's,指令集参考 (A-M)”和“英特尔64 和 iA-32 架构软件开发人员手册卷 2B's,指令集参考 (N-Z)”一书中对这些指令的叙述。