**一、多核或共享内存的多CPU硬件环境中，是否可采用关中断实现内核代码的临界区管理?为什么?**

**答：**

**（1）不可以。**

**（2）原因：**

**1. 核心原因是关中断只在单核上有效，在多核或者共享内存的多CPU硬件环境中无效：**

单核CPU如果进临界区之前关了中断，那么临界区的代码确实是不受干扰的单独运行的。但是多核或者共享内存的多CPU硬件环境中，关中断只能关代码当前所在运行的CPU上的中断，完全避免不了代码被其他CPU上的线程所运行，即避免不了别的核进入临界区，所以是无效的。

**2. 次要原因还有：**

（1）性能下降：关中断会阻止中断处理，这对实时性要求高的系统来说是不利的。

（2）复杂性和可扩展性问题：在多核系统中，通过关中断来管理临界区会增加系统的复杂性。为了保证同步，需要在所有核之间进行协调，这通常需要更复杂的机制，如跨核的中断屏蔽，这会导致代码难以维护和扩展。

**3. 正确做法：**

采用加锁、PV操作、原子操作等等来解决。

**二、采用gcc编译多线程程序时，为何需要加上-pthread选项，其主要作用是什么?**

**答：**

**（1）不加带来的后果：**

如果在编译和链接阶段不加上 -pthread 选项，程序可能会出现以下问题：

(1) 无法找到 pthread 库中的函数，从而导致链接错误。

(2) 没有定义 \_REENTRANT 宏，可能导致标准库函数在多线程环境中表现不正确或不安全。

**（2）-pthread的主要作用：**

其主要作用如下：

**1.启用 POSIX 线程库：**

-pthread 选项会告诉编译器和链接器使用 POSIX 线程库（也称为 pthreads）。这会影响编译和链接的多个方面：

**2.编译阶段的作用：**

编译器会定义 \_REENTRANT 宏，启用线程安全代码的编译。该宏通知标准库和其他依赖库以线程安全的方式进行编译。

**3.链接阶段的作用：**

链接器会将 POSIX 线程库（通常为 libpthread.so 或 libpthread.a）链接到最终的可执行文件中。这样，程序就可以调用 pthread 库提供的各种线程相关函数，例如 pthread\_create、pthread\_join 等。

**4.线程安全的作用：**

启用线程库和定义 \_REENTRANT 宏，确保库中的函数（如 malloc、printf 等）在多线程环境中是线程安全的。

**5.编译和链接的兼容性的作用：**

有些函数和功能只有在启用 POSIX 线程支持时才可用，使用 -pthread 选项可以确保代码正确编译和链接，以使用这些功能。

**三、两个并发进程P1和P2，其中A、B、C、D、E是五个不同的机器指令，P1进程的执行序列为A, B, C，P2进程的执行序列为D, E，试给出两个进程并发执行时可能的执行序列。**

**答：**

**一共有5！/（2！ \* 3！）= 10种，分别如下：**

**1. ABCDE**

**2. ABDCE**

**3. ABDEC**

**4. ADBCE**

**5. ADBEC**

**6. ADEBC**

**7. DABCE**

**8. DABEC**

**9. DAEBC**

**10.DEABC**

**四、一座仅有一个车道的东西向桥梁，为保证交通安全，如果桥上无车，则允许一方车辆过桥，待其全部过完后，才允许另一方车辆过桥。试用信号量和PV操作描述上述问题的同步算法。**

**答：**

要解决这个问题，首先定义如下的信号量和变量：

semaphore bridge= 1； // 独木桥的数量，独木桥只有一个，这里不区分东西方向，有了这个信号量就代表有过桥的资源

int count1 = 0; int count2 = 0; // 东侧、西侧车辆在独木桥（两端，即还没过完）上的数量

semaphore mutex1 = 1; semaphore mutex2 = 1; // 东侧、西侧车辆的互斥信号量

cobegin

process P东() {

P(mutex1); //想上桥

count1++;

if(count1 == 1) // 东侧第一个准备上桥的车去抢夺独木桥

P(bridge);

V(mutex1);

{过独木桥};

P(mutex1);

count1--;

if(count1 == 0) // 东侧最后一个已经下桥的车去释放独木桥

V(bridge);

V(mutex1);

}

process P西() {

P(mutex2); //想上桥

count2++;

if(count2 == 1) // 西侧第一个准备上桥的车去抢夺独木桥

P(bridge);

V(mutex2);

{过独木桥};

P(mutex2);

count2--;

if(count2 == 0) // 西侧最后一个已经下桥的车去释放独木桥

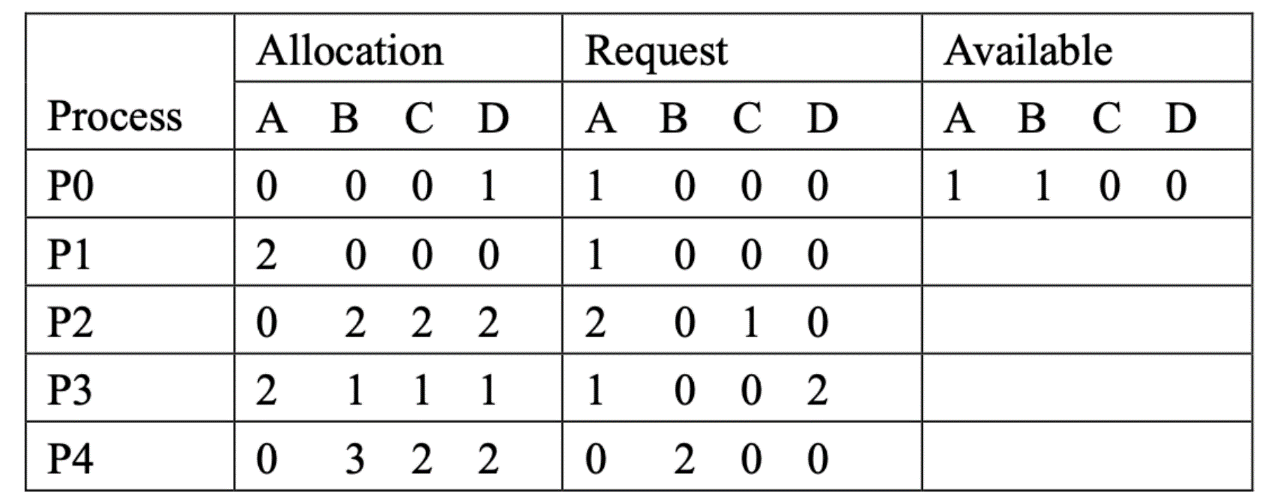
V(bridge);

V(mutex2);

}

coend

**5. 若某个系统中有5个并发进程分别是 P0、P1、P2、P3、P4，四类资源分别标记为 A、B、 C、D，系统目前各进程的资源分配、申请情况如下表所示:**



**该系统是否发生死锁，若发生死锁，则涉及到了哪几个进程？**

**答：**

（1）首先将资源给P0，结果是Available变为 1 1 0 1；

（2）接下来能够给P1，结果是Available变为 3 1 0 1；

（3）接下来以任意顺序都无法满足P2、P3、P4的request，产生不了一个合法序列，资源分配图不可完全简化。

通过以上分析，知道系统发生了死锁，涉及进程P2、P3、P4。