三种基本I/O方式

- 程序直接控制方式（最简单的I/O方式）
  - 无条件传送：对简单外设定时（同步）进行数据传送
  - 条件传送：CPU主动查询，也称程序查询或轮询（Polling）方式
- I/O Interrupt (中断I/O方式): 几乎所有系统都支持中断I/O方式
  - 若一个I/O设备需要CPU干预，它就通过中断请求通知CPU
  - CPU中止当前程序的执行，调出OS（中断处理程序）来执行
  - 处理结束后，再返回到被中止的程序继续执行
- Direct Memory Access (DMA方式): 磁盘等高速外设所用的方式
  - 磁盘等高速外设成批地直接和主存进行数据交换
  - 需要专门的DMA控制器控制总线，完成数据传送
  - 数据传送过程无需CPU参与

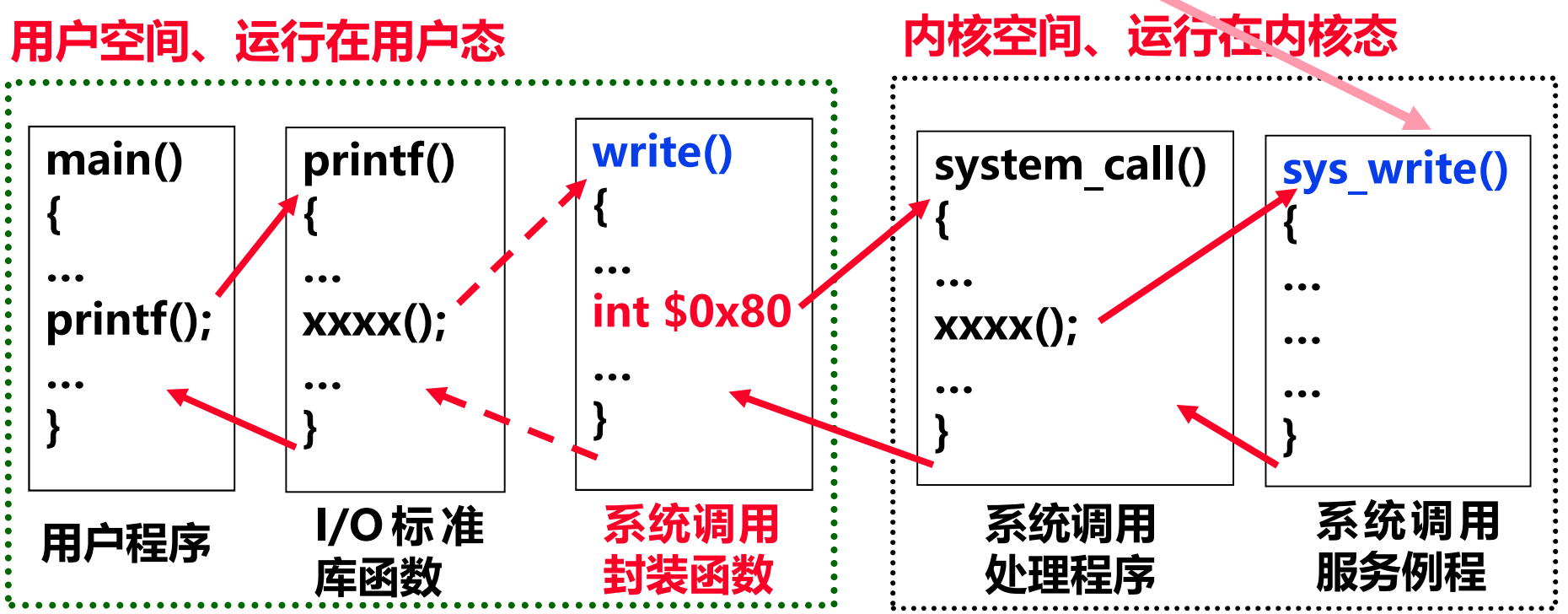
以hello程序为例说明

假定以下用户程序对应的进程为p

```
#include <stdio.h>
int main()
{
    printf("hello, world\n");
}
```

sys_write可用三种I/O方式实现：
程序查询、中断 和 DMA

字符串输出最终是由内核中的
sys_write系统调用服务例程实现



程序查询（Polling）方式

- I/O设备（包括设备控制器）将自己的状态放到状态寄存器中
  - 打印缺纸、打印机忙、未就绪等都是状态
- OS阶段性地查询状态寄存器中的特定状态，以决定下一步动作
  - 如：未“就绪”时，则一直“等待”
- 例如：sys_write进行字符串打印的程序段大致过程如下：

```
copy_string_to_kernel ( strbuf, kernelbuf, n);  // 将字符串复制到内核缓冲区
for (i=0; i < n; i++) {                          // 对于每个打印字符循环执行
  while ( printer_status != READY);              // 等待直到打印机状态为“就绪”
  *printer_data_port=kernelbuf[i];               // 向数据端口输出一个字符
  *printer_control_port=START;                   // 发送“启动打印”命令
}
return_to_user ( );                              // 返回用户态
```

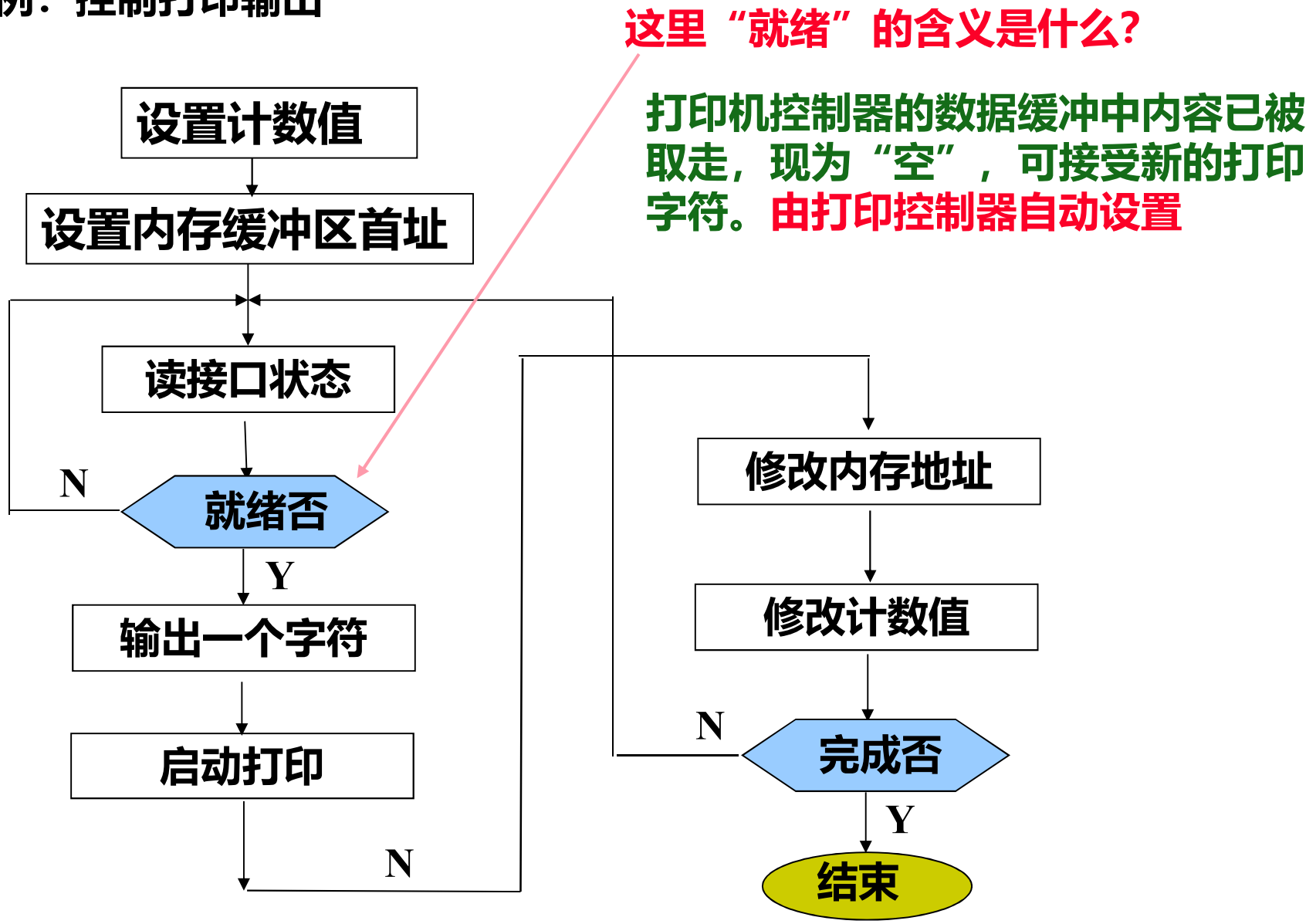
如何判断“就绪”？如何“等待”？

读取状态寄存器，判断特定位（1-就绪；0-未就绪）是否为1

等待：读状态、判断是否为1；不是，则继续读状态、判断、……

程序查询（Polling）方式

举例：控制打印输出



打印输出标准子程序

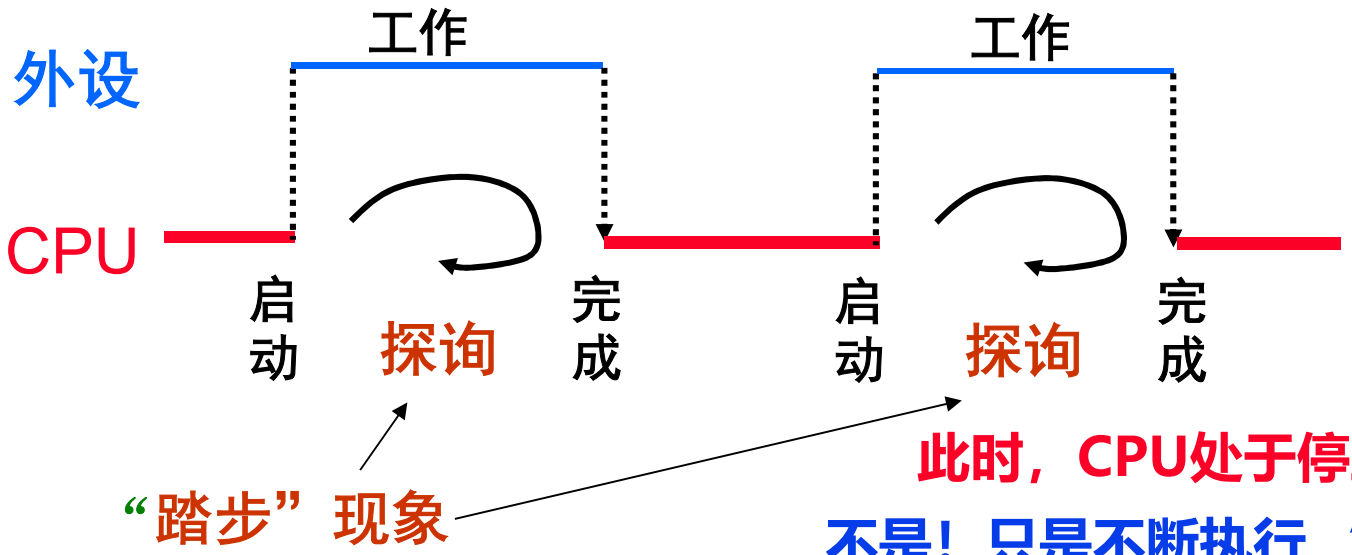
功能：打印AL寄存器中的字符。

```
PRINT   PROC  NEAR                                    SKIP
        PUSH  AX          ;保留用到的寄存器
        PUSH  DX          ;保留用到的寄存器
        MOV   DX, 378H    ;数据锁存器口地址送DX
        OUT   DX, AL      ;输出要打印的字符到数据锁存器
        MOV    DX, 379H   ;状态寄存器口地址送DX
WAIT:   IN    AL, DX      ;读打印机状态位
        TEST  AL, 80H     ;检查忙位
        JE    WAIT        ;等待直到打印机不忙
        MOV    DX, 37AH   ;命令(控制)寄存器口地址送DX
        MOV   AL, 0DH     ;置选通位=1 (表示启动打印)
        OUT   DX, AL      ;使命令寄存器中选通位置1
        POP   DX
        POP   AX          ;恢复寄存器
        RET
PRINT   ENDP
```

回顾：过程/函数/子程序中的开始总是先要保护现场，最后总是要恢复现场！

程序查询I/O方式

sys_write系统调用服务例程



此时，CPU处于停止状态吗?

不是！只是不断执行 “ IN-TEST-JE” 3条指令，称为“忙等待”！

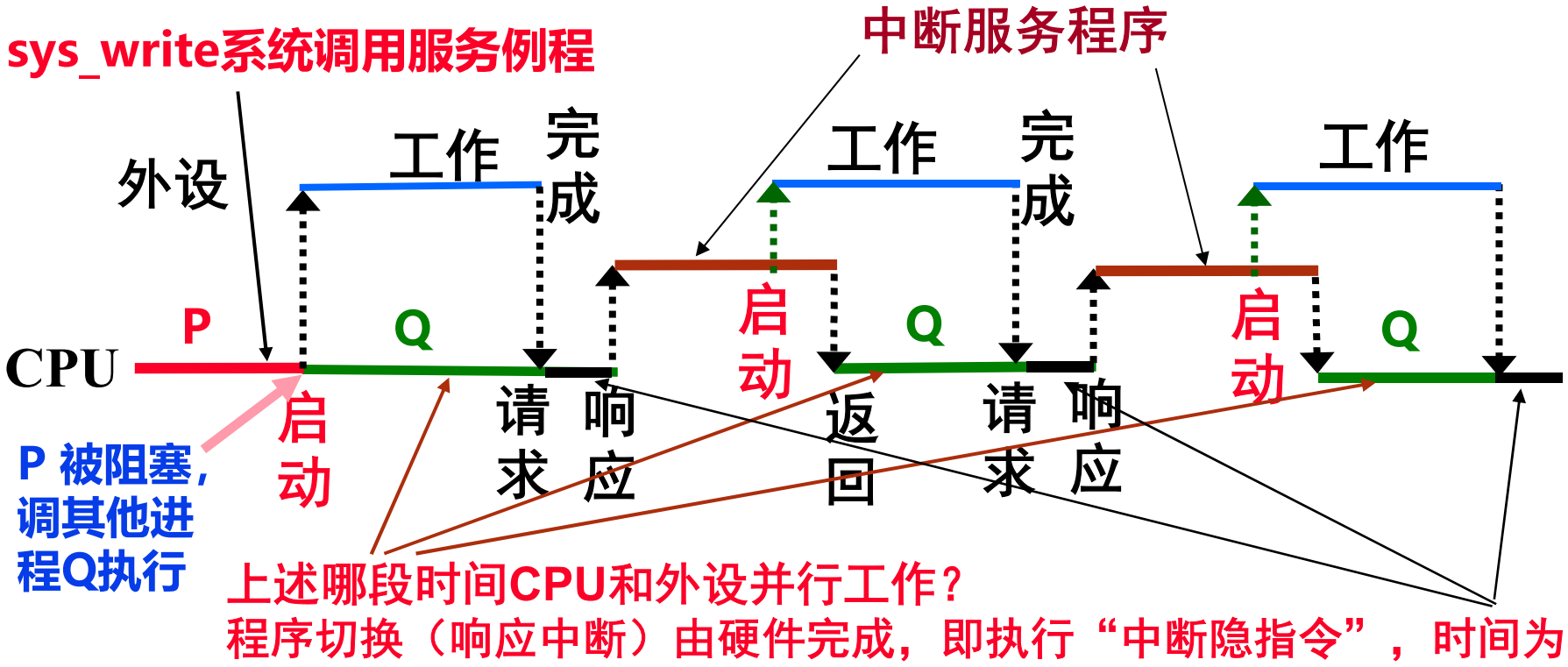
“探询”期间，可一直不断查询（独占查询），也可定时查询（需保证数据不丢失！）。

- **特点:**
  - **简单、易控制、外围接口控制逻辑少;**
  - **CPU与外设串行工作，效率低、速度慢，适合于慢速设备**
  - **查询开销极大 (CPU完全在等待“外设完成”)**
- **工作方式：完全串行或部分串行，CPU用100%的时间为I/O服务!**

中断I/O方式

- **基本思想：**

  当外设准备好（ready）时，便向CPU发中断请求，CPU响应后，中止现行程序的执行，转入“中断服务程序”进行输入/出操作，以实现主机和外设接口之间的数据传送，并启动外设工作。“中断服务程序”执行完后，返回原被中止的程序断点处继续执行。此时，外设和CPU并行工作。



中断I/O方式

例子：采用中断方式进行字符串打印

sys_write进行字符串打印的程序段:

```
copy_string_to_kernel ( strbuf, kernelbuf, n);// 将字符串复制到内核缓冲区
enable_interrupts ( );                        // 开中断，允许外设发出中断请求
while ( printer_status != READY);             // 等待直到打印机状态为“就绪”
*printer_data_port=kernbuf[i];                // 向数据端口输出第一个字符
*printer_control_port=START;                  // 发送“启动打印”命令
scheduler ( );                                // 阻塞用户进程P，调度其他进程执行
```

“字符打印”中断服务程序：

```
if (n==0) {                    // 若字符串打印完，则
unblock_user ( );              // 用户进程P解除阻塞，P进就绪队列
} else {
   *printer_data_port=kernelbuf[i]; // 向数据端口输出一个字符
   *printer_control_port=START;     // 发送“启动打印”命令
    n = n-1;                   // 未打印字符数减1
    i = i+1;                   // 下一个打印字符指针加1
}
acknowledge_interrupt();       // 中断回答（清除中断请求）
return_from_interrupt();       // 中断返回
```

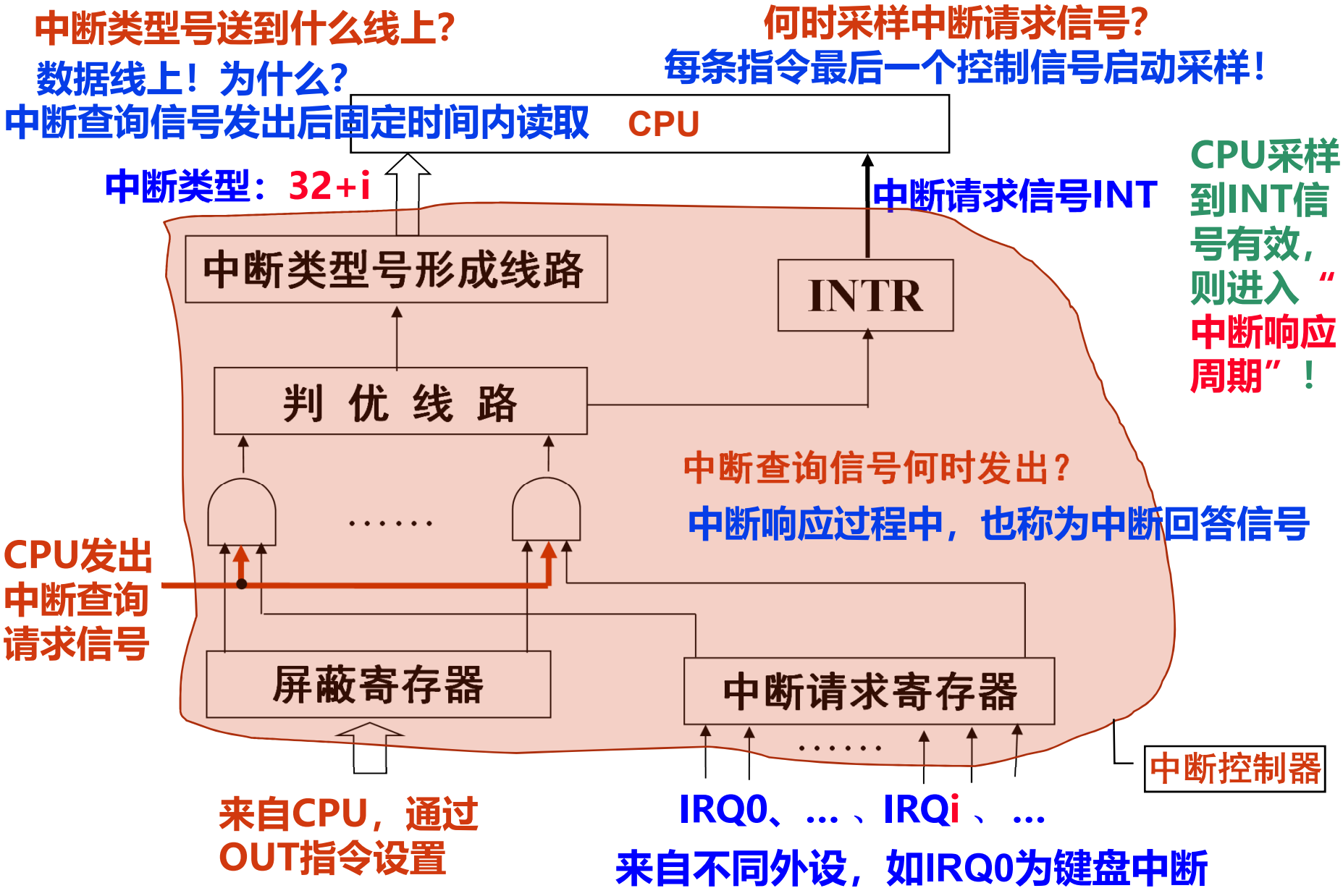
sys_write是如何调出来的?

系统调用!

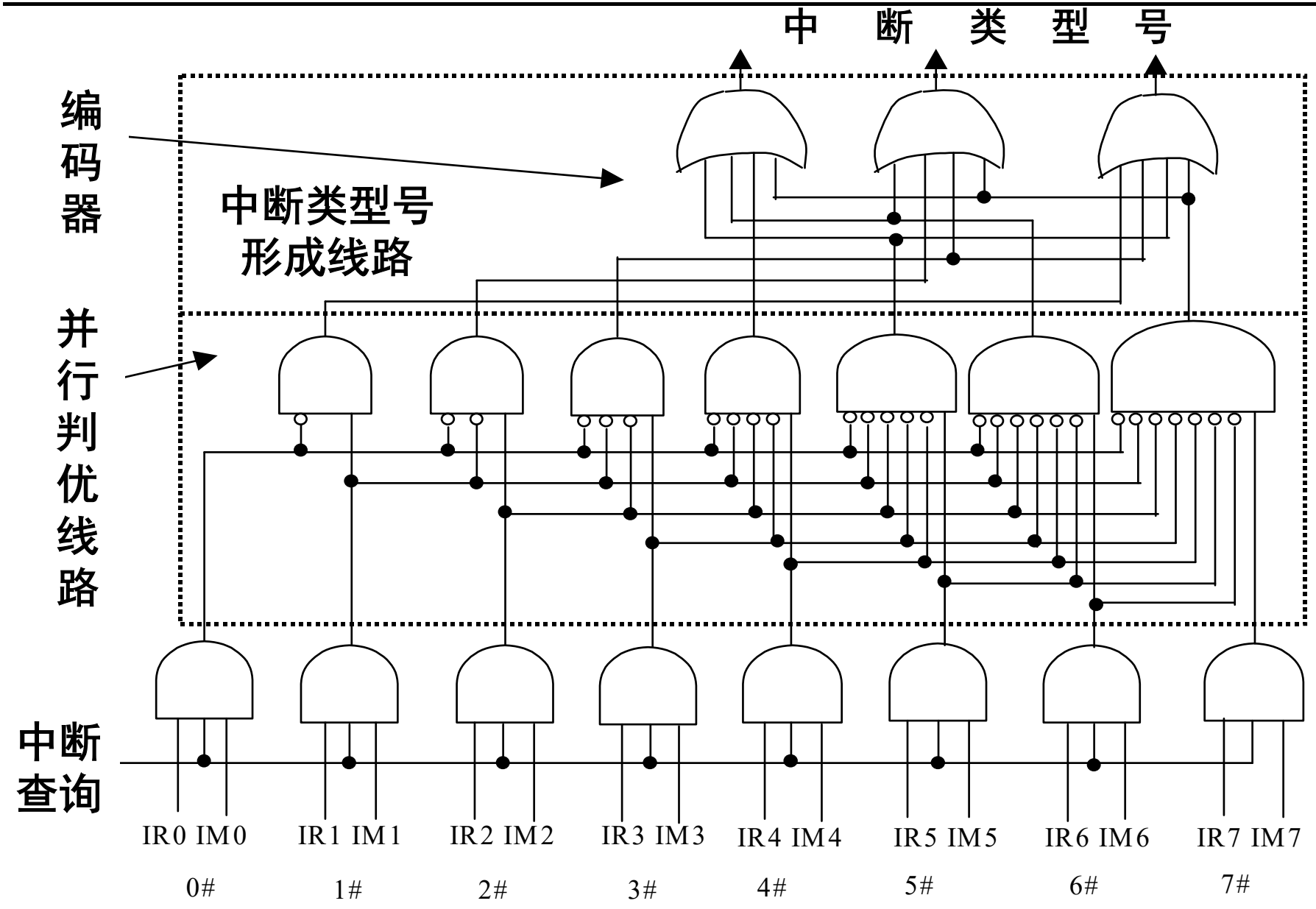
中断服务程序是如何调出来的?

外设完成任务!

中断控制器的基本结构

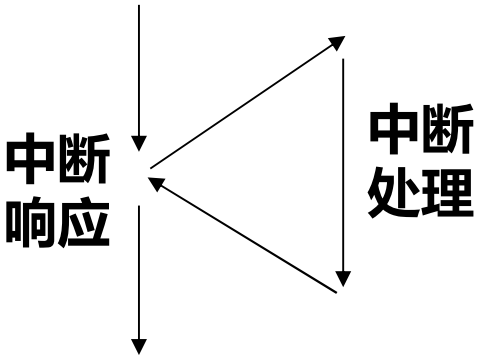

中断类型号送到什么线上?
数据线上！为什么?
何时采样中断请求信号?
每条指令最后一个控制信号启动采样!
中断查询信号发出后固定时间内读取
CPU
中断类型：32+i
中断请求信号INT
CPU采样到INT信号有效，则进入“中断响应周期”！
中断类型号形成线路
INTR
判 优 线 路
中断查询信号何时发出?
中断响应过程中，也称为中断回答信号
CPU发出中断查询请求信号
屏蔽寄存器
中断请求寄存器
中断控制器
来自CPU，通过OUT指令设置
IRQ0、…、IRQi 、 …
来自不同外设，如IRQ0为键盘中断

中断优先权编码器


中 断 类 型 号
编码器
中断类型号
形成线路
并行判优线路
中断查询
IR0 IM0
IR1 IM1
IR2 IM2
IR3 IM3
IR4 IM4
IR5 IM5
IR6 IM6
IR7 IM7
0#
1#
2#
3#
4#
5#
6#
7#

中断I/O方式

- 中断过程
  - 中断检测（硬件实现）
  - 中断响应（硬件实现）
  - 中断处理（软件实现）



- 中断响应
  - 中断响应是指主机发现外部中断请求，中止现行程序的执行，到调出中断服务程序这一过程。

中断响应的条件

① CPU处于开中断状态

② 在一条指令执行完

③ 至少要有一个未被屏蔽的中断请求

SKIP

问题：中断响应的时点与异常处理的时点是否相同？为什么？

通常在指令执行结束时查询有无中断请求，有则立即响应；

而异常发生在指令执行过程中，一旦发现则马上处理。

回顾：异常/中断响应过程

检测到异常或中断时，CPU须进行以下基本处理：

① 关中断（“中断允许位” 清0）：使CPU处于“禁止中断”状态，以防止新中断破坏断点（PC）、程序状态（PSW）和现场（通用寄存器）。

② 保护断点和程序状态：将断点和程序状态保存到栈或特殊寄存器中

PC→栈 或 EPC（专门存放断点的寄存器）

PSWR →栈 或 EPSWR （专门保存程序状态的寄存器）

PSW（Program Status Word）：程序状态字

PSWR（PSW寄存器）：如IA-32中的的EFLAGS寄存器

③ 识别中断事件

有软件识别和硬件识别（向量中断）两种不同的方式。 BACK

IA-32中，响应异常时不关中断，只在响应中断时关中断

中断处理过程

中断响应的结果就是调出相应的中断服务程序

中断处理是指执行相应中断服务程序的过程

- 不同的中断源其对应的中断服务程序不同。
- 典型的多重中断处理（中断服务程序）分为三个阶段：
  - 先行段（准备阶段）
    - 保护现场及旧屏蔽字
    - 查明原因（软件识别中断时）
    - 设置新屏蔽字

    处在“关中断”状态，不允许被打断
    - 开中断

    处在“开中断”状态，可被新的处理优先级更高的中断打断
  - 本体段（具体的中断处理阶段）
  - 结束段（恢复阶段）
    - 关中断
    - 恢复现场及旧屏蔽字
    - 清“中断请求”

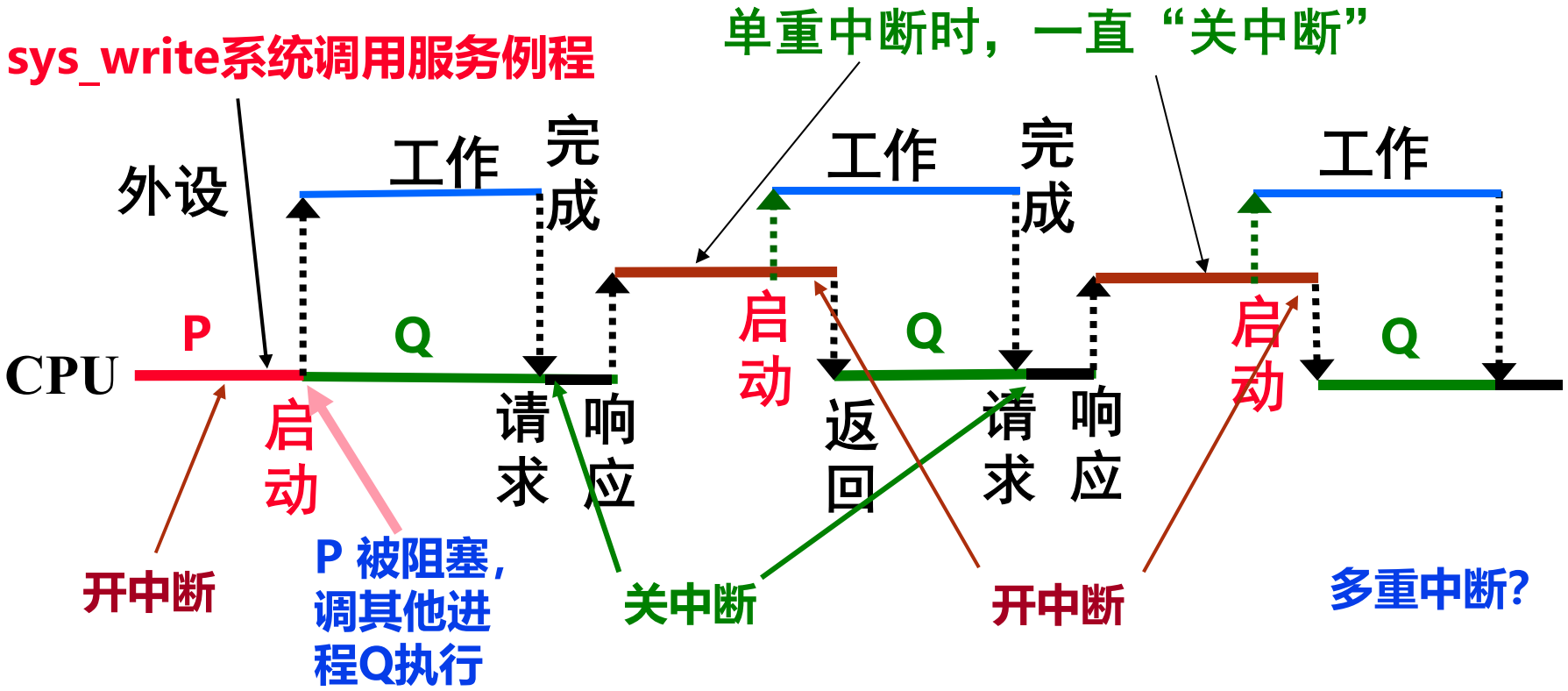
    处在“禁止中断”状态，不允许被打断
    - 开中断
    - 中断返回

单重中断不允许在中断处理时被新的中断打断，因而直到中断返回前才会开中断。单重中断系统无需设置中断屏蔽字。

回顾：中断I/O方式

- **基本思想：**

  当外设准备好（ready）时，便向CPU发中断请求，CPU响应后，中止现行程序的执行，转入“中断服务程序”进行输入/出操作，以实现主机和外设接口之间的数据传送，并启动外设工作。 “中断服务程序”执行完后，返回原被中止的程序断点处继续执行。此时，外设和CPU并行工作。



多重中断的概念

SKIP

- 多重中断和中断处理优先权的动态分配
  - **多重中断的概念：**

    在一个中断处理（即执行中断服务程序）过程中，若又有新的中断请求发生，而新中断优先级高于正在执行的中断，则应立即中止正在执行的中断服务程序，转去处理新的中断。这种情况为多重中断，也称中断嵌套。

  - **中断优先级的概念：**

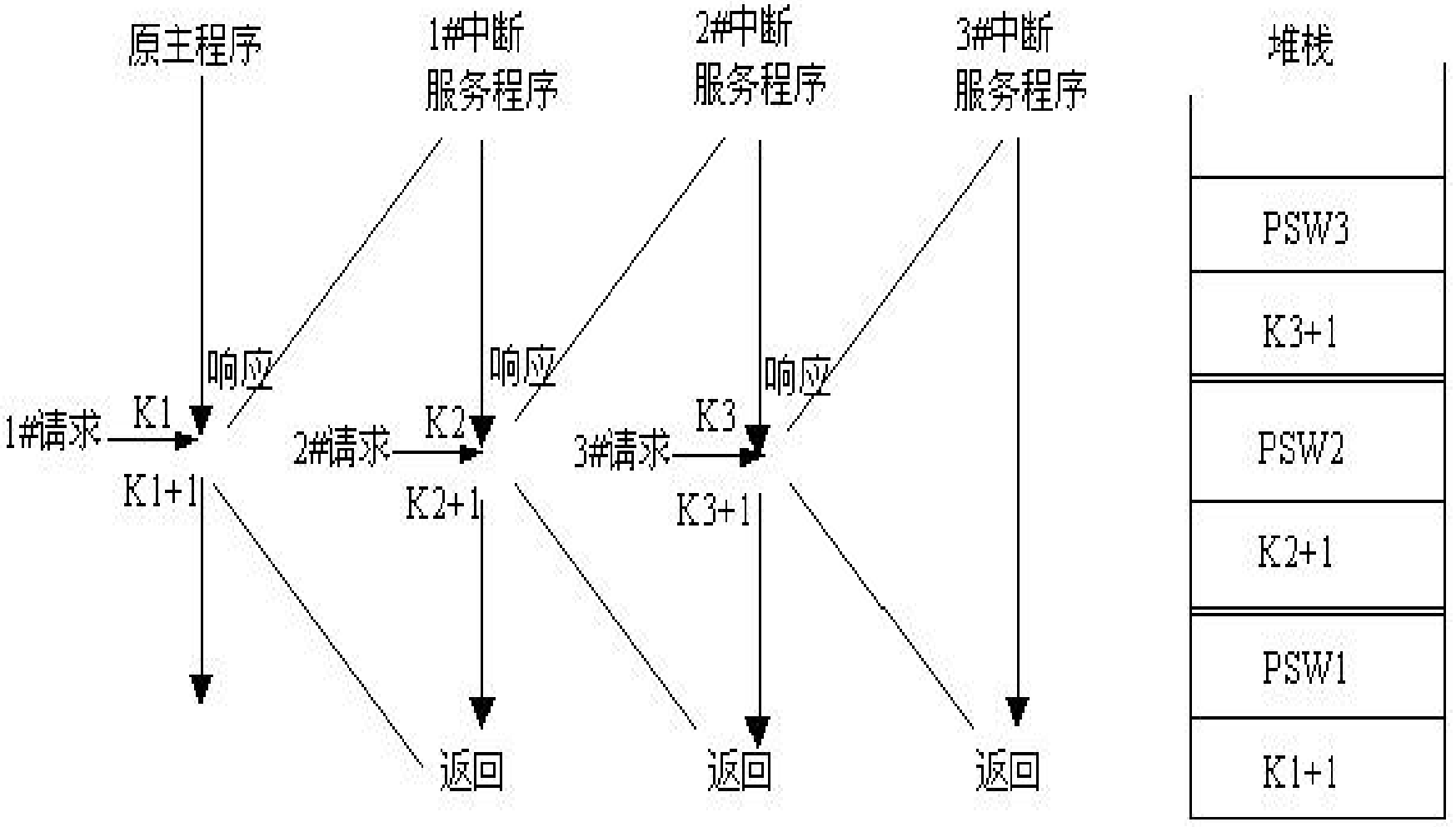
    中断响应优先级----由查询程序或硬联排队线路决定的优先权，反映多个中断同时请求时选择哪个响应。

    中断处理优先级----由各自的中断屏蔽字来动态设定，反映本中断与其它中断间的关系。

    回想一下，中断屏蔽字在何处用到的?

多重中断嵌套

BACK



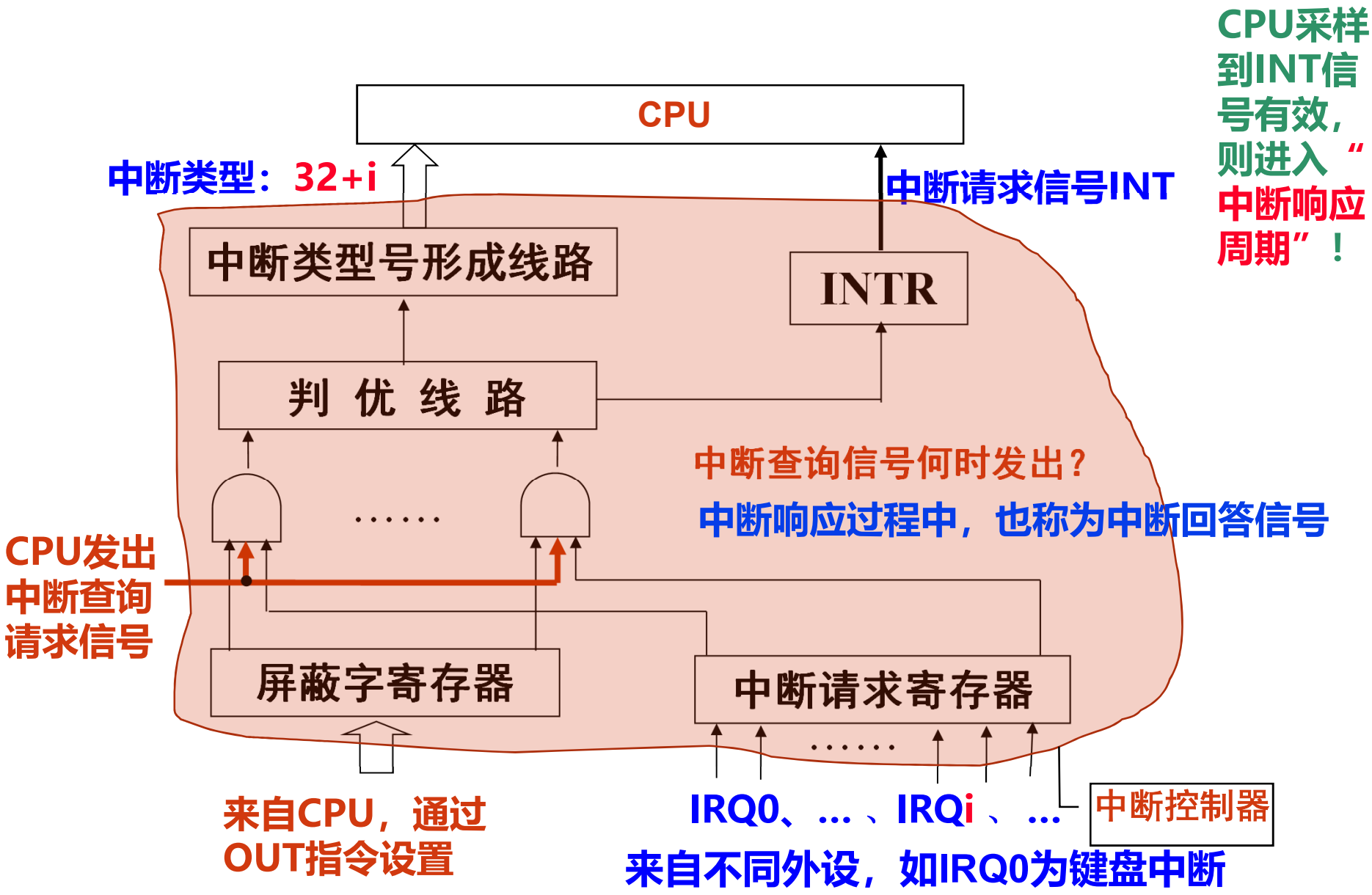
中断处理优先级的顺序是：

3# > 2# > 1#

1# 对 2# 开放（不屏蔽）

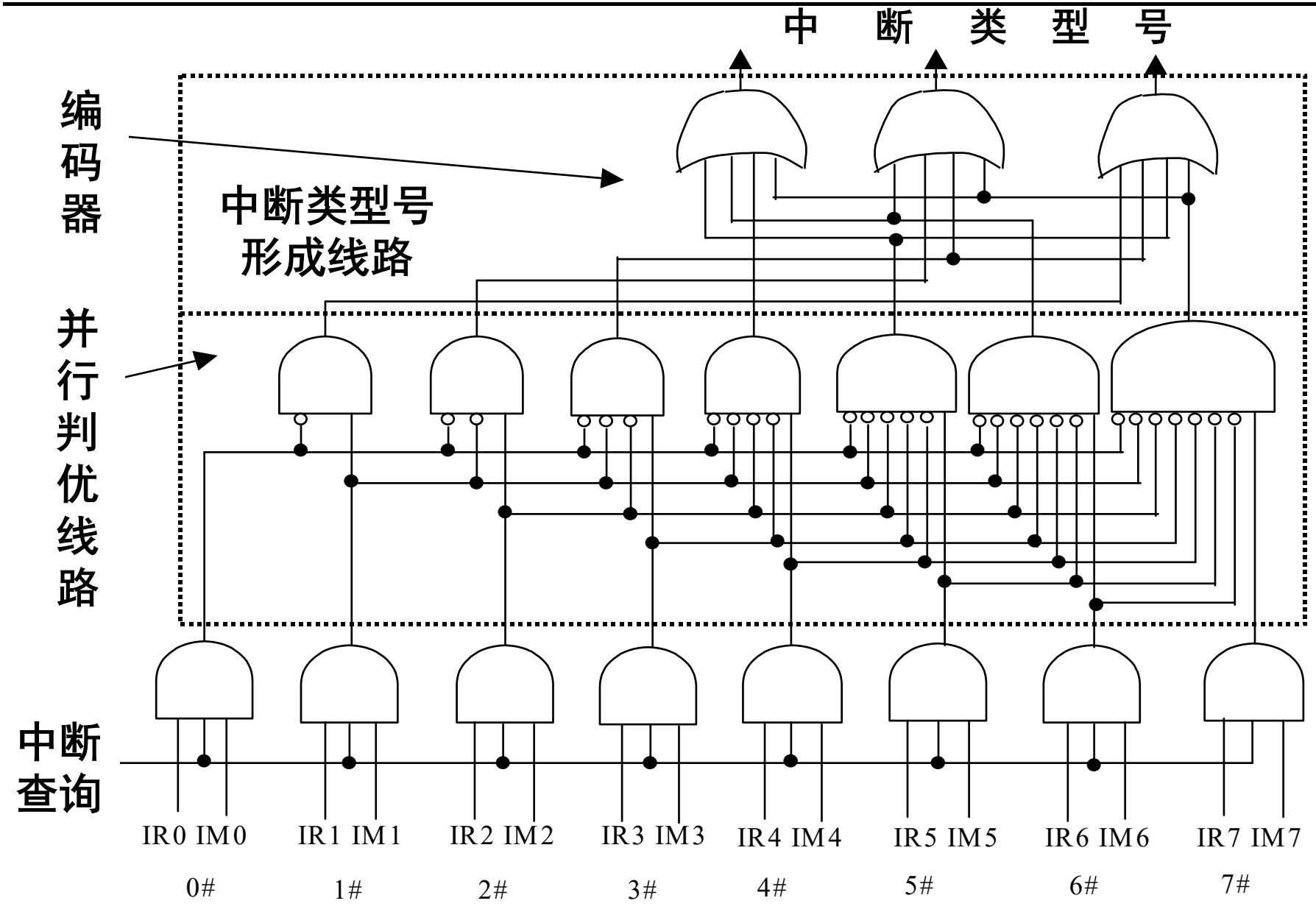
2# 对 3# 开放（不屏蔽）

中断控制器的基本结构


CPU采样到INT信号有效，则进入“中断响应周期”！
CPU
中断类型：32+i
中断请求信号INT
中断类型号形成线路
INTR
判 优 线 路
中断查询信号何时发出？
中断响应过程中，也称为中断回答信号
CPU发出中断查询请求信号
屏蔽字寄存器
中断请求寄存器
来自CPU，通过OUT指令设置
IRQ0、…、IRQi、…
中断控制器
来自不同外设，如IRQ0为键盘中断

中断优先权编码器

BACK

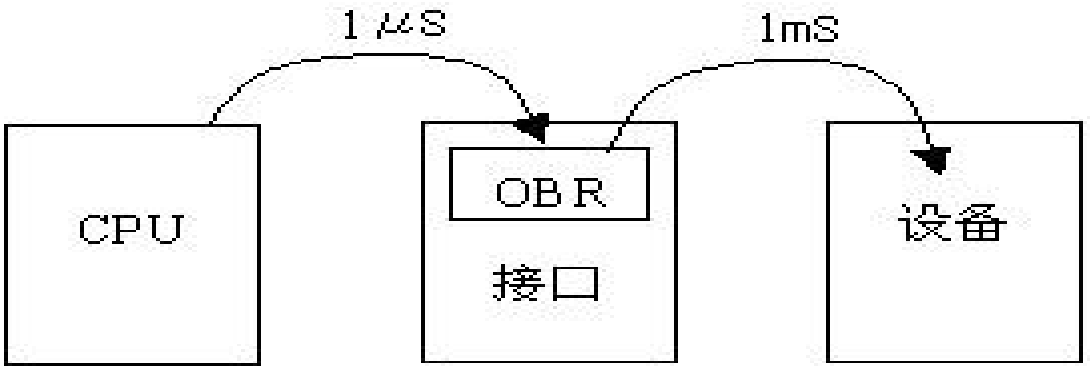


轮询方式和中断方式的比较

- 举例：假定某机控制一台设备输出一批数据。数据由主机输出到接口的数据缓冲器OBR，需要1μs。再由OBR输出到设备，需要1ms。设一条指令的执行时间为1μs(包括隐指令)。试分别计算采用轮询方式和中断方式的数据传输速度和对主机的占用率。

问题：CPU如何把数据送到OBR，I/O接口如何把OBR中的数据送到设备?

CPU执行I/O指令来将数据送OBR；而I/O接口则是自动把数据送到设备。



对主机占用率：

在进行I/O操作过程中，处理器有多少时间花费在输入/出操作上。

数据传送速度（吞吐量、I/O带宽）：

单位时间内传送的数据量。

假定每个数据的传送都要重新启动！也即，是字符型设备

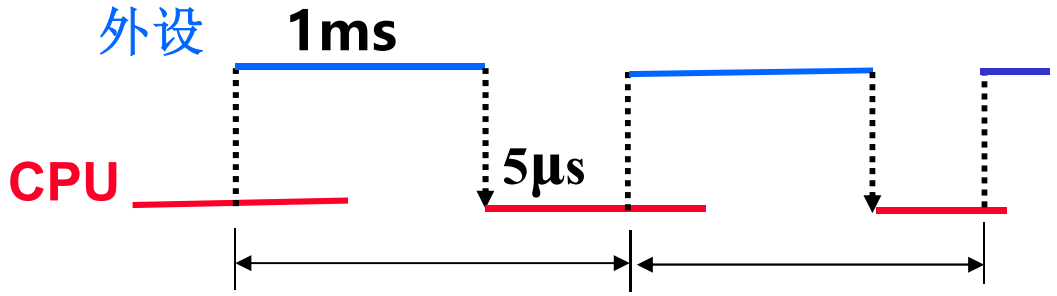
轮询方式和中断方式的比较

（1）程序直接控制传送方式

若查询程序有10条，第5条为启动设备的指令，则：

数据传输率为：1/(1000+5) μs，约为每秒995个数据。

主机占用率=100%



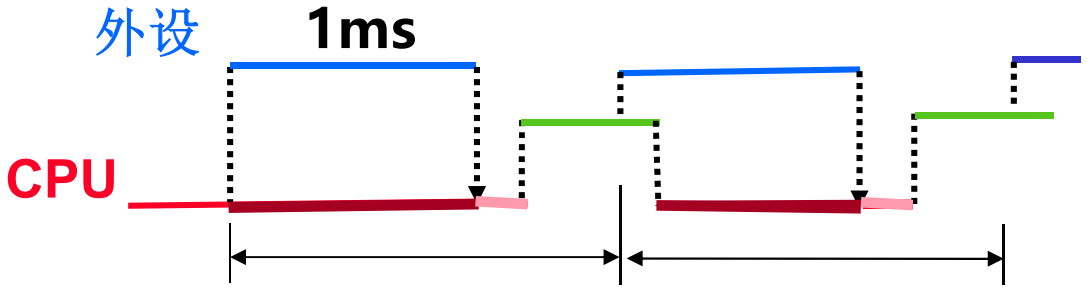
轮询方式

（2）中断传送方式

若中断服务程序有30条，在第20条启动设备，则：

数据传输率为：1/(1000+1+20)μs，约为每秒979个数据。

主机占用率为：(1+30)/(1000+1+20)=3%



中断方式

为什么中断服务程序比查询程序长?

因为中断服务程序有额外开销，如：保存现场、保存旧屏蔽字、设置新屏蔽字、开中断、查询中断源等

若是磁盘等高速设备与主机交换数据，那么，采用中断方式会怎么样?

DMA方式的基本要点

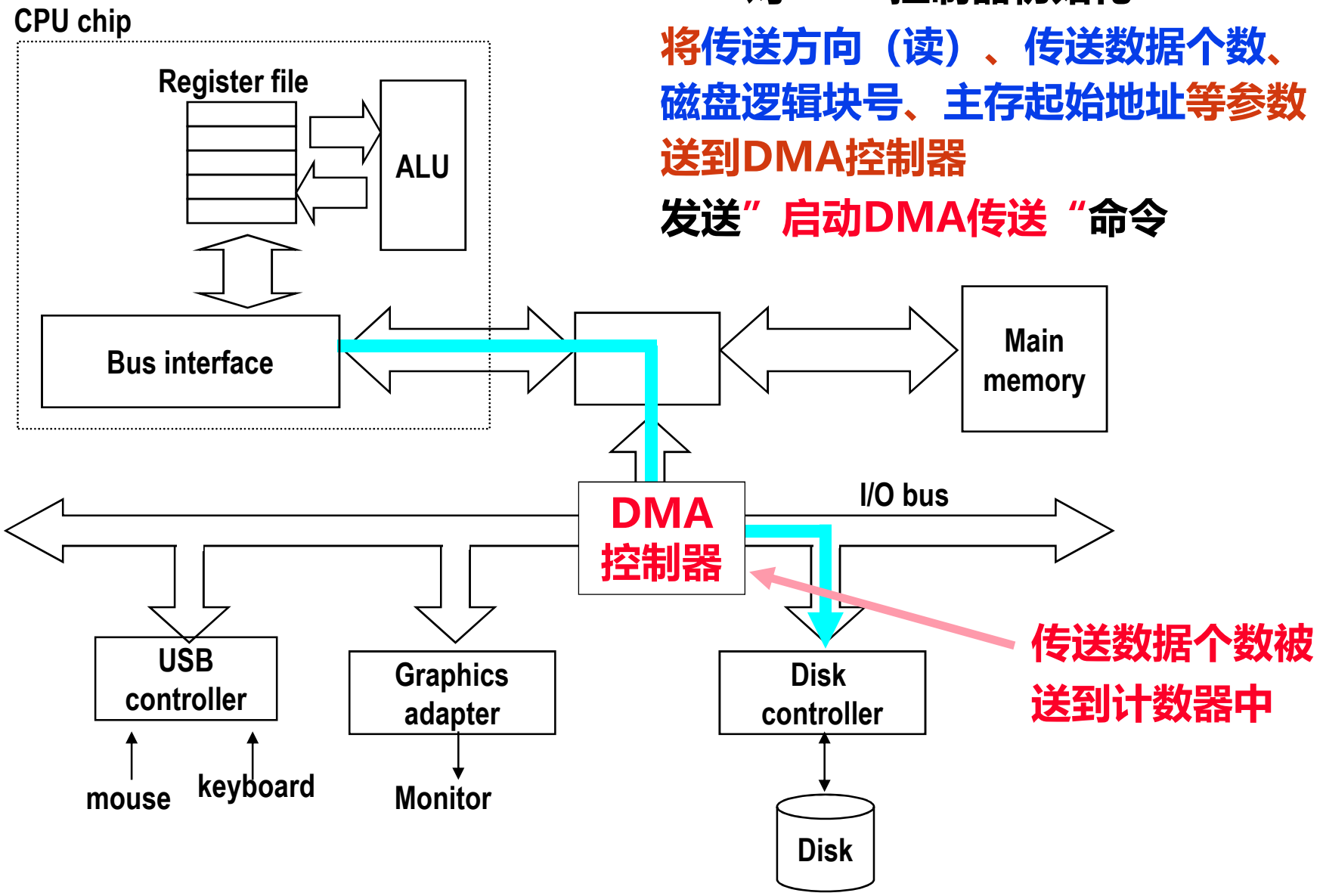
- DMA方式的基本思想
  - 在高速外设和主存间直接传送数据
  - 由专门硬件（即：DMA控制器）控制总线进行传输
- DMA方式适用场合
  - 高速设备（如：磁盘、光盘等）
  - 成批数据交换，且数据间间隔时间短，一旦启动，数据连续读写
- 采用“请求-响应”方式
  - 每当高速设备准备好数据就进行一次“DMA请求”，DMA控制器接受到DMA请求后，申请总线使用权
  - DMA控制器的总线使用优先级比CPU高，为什么?
- 与中断控制方式结合使用
  - 在DMA控制器控制总线进行数据传送时，CPU执行其他程序
  - DMA传送结束时，要通过“DMA结束中断”告知CPU

读一个磁盘扇区 – 第一步

CPU对DMA控制器初始化：

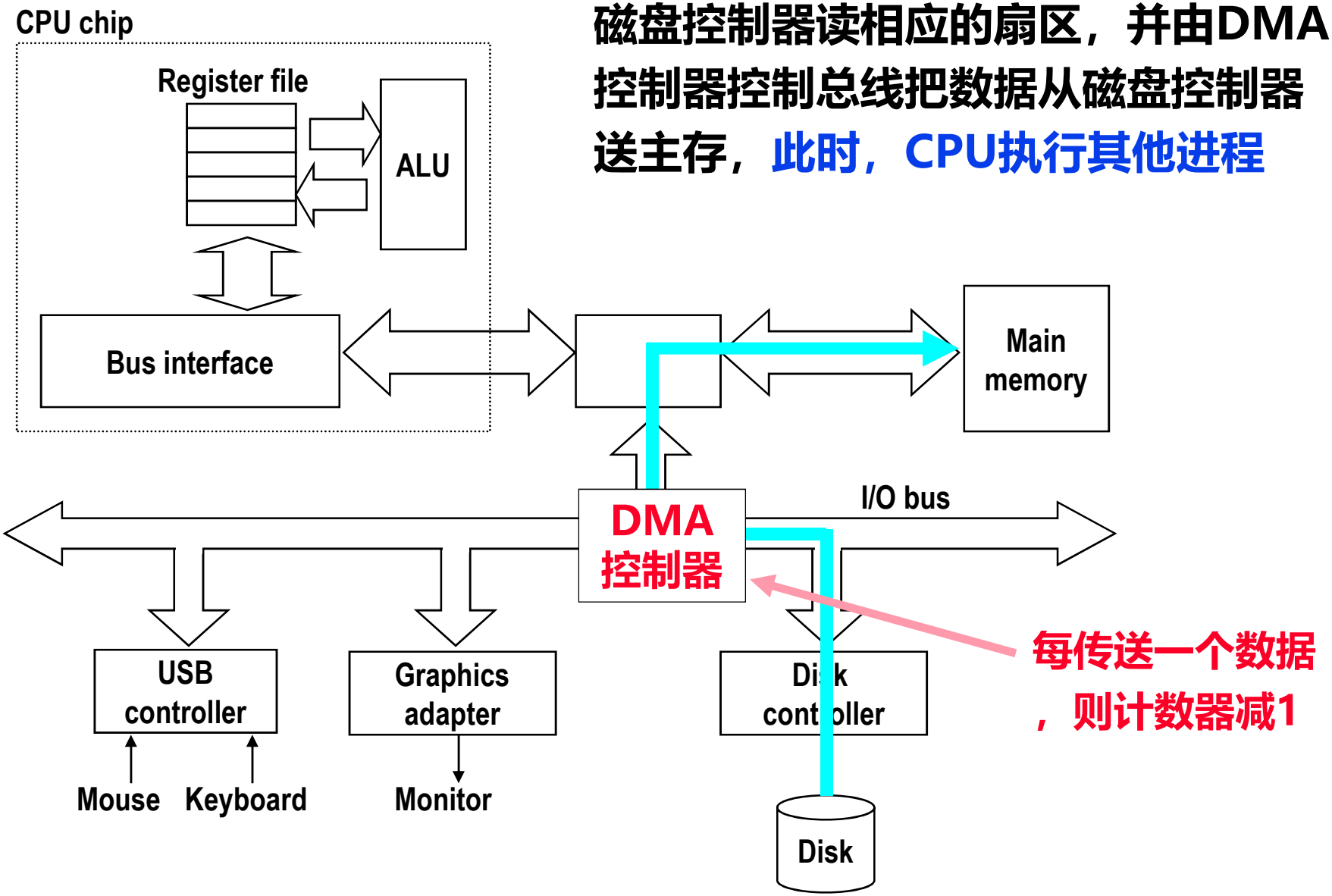
将传送方向（读）、传送数据个数、磁盘逻辑块号、主存起始地址等参数送到DMA控制器

发送”启动DMA传送“命令



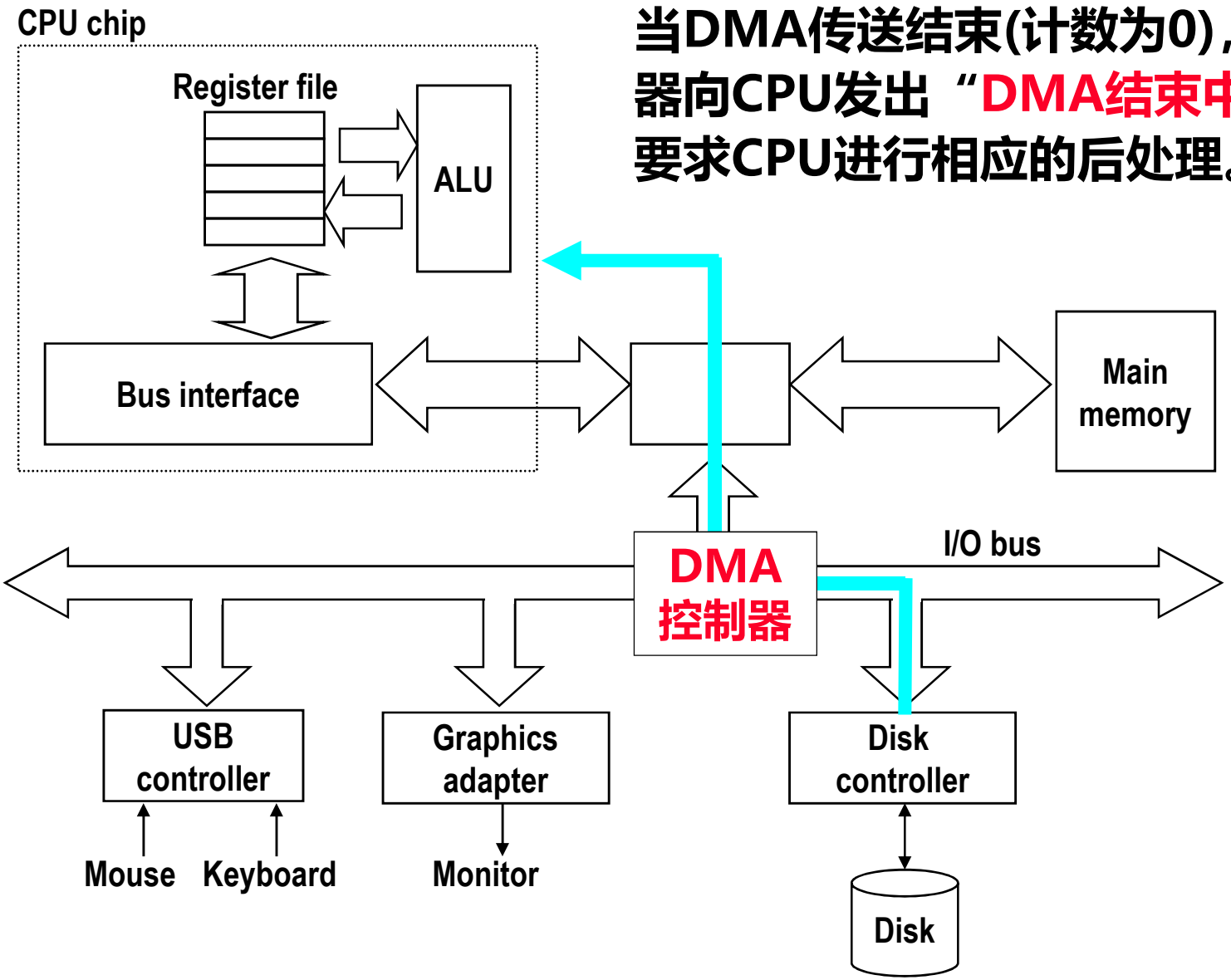
读一个磁盘扇区 - 第二步

磁盘控制器读相应的扇区，并由DMA控制器控制总线把数据从磁盘控制器送主存，此时，CPU执行其他进程



读一个磁盘扇区 - 第三步

当DMA传送结束(计数为0)，DMA控制器向CPU发出“DMA结束中断请求”，要求CPU进行相应的后处理。



DMA方式下CPU的工作

例子：采用DMA方式进行字符串输出

sys_write进行字符串输出的程序段:

```
copy_string_to_kernel(strbuf, kernelbuf, n);  // 将字符串复制到内核缓冲区
initialize_DMA ( );                           // 初始化DMA控制器（准备传送参数）
*DMA_control_port=START;                      // 发送“启动DMA传送”命令
scheduler ( );                                // 阻塞用户进程P，调度其他进程执行
```

DMA控制器接受到“启动”命令后，控制总线进行DMA传送。通常用”周期挪用法“：设备每准备好一个数据，挪用一次”存储周期“，使用一次总线事务进行数据传送，计数器减1。计数器为0时，发送DMA结束中断请求

”DMA结束“中断服务程序：

```
acknowledge_interrupt();      // 中断回答（清除中断请求）
unblock_user ( );             // 用户进程P解除阻塞，进入就绪队列
return_from_interrupt();      // 中断返回
```

CPU仅在DMA控制器初始化和处理“DMA结束中断“时介入，在DMA传送过程中不参与，因而CPU用于I/O的开销非常小。

例：中断、DMA方式下CPU的开销

设处理器按500MHz的速度执行，磁盘控制器中有一个16B的数据缓存器，磁盘传输速率为4MB/Sec，在磁盘传输数据过程中，要求没有任何数据被错过，并假定CPU访存和DMA访存没有冲突。

（1）若用中断方式，每次传送的开销（包括用于中断响应和处理的时间）是500个时钟周期。如果硬盘仅用5%的时间进行传送，那么处理器用在硬盘I/O操作上所花的时间百分比（主机占用率）为多少?

（2）若用DMA方式，处理器用1000个时钟进行DMA传送初始化，在DMA完成后的中断处理需要500个时钟。如果每次DMA传送8000B的数据块，那么当硬盘进行传送的时间占100%（即：硬盘一直进行读写，并传输数据）时，处理器用在硬盘I/O操作上的时间百分比（主机占用率）为多少?

- **中断传送：**
  - 硬盘每次中断，可以以16字节为单位进行传送，为保证没有任何数据被错过，应达到每秒4MB /16B=250k次中断的速度；
  - 每秒钟用于中断的时钟周期数为250k x 500=$125x10^6$；
  - 在一次数据传输中，处理器花费在I/O上的时间的百分比为：

    $125x10^6/(500x10^6)=25\%$
  - 假定硬盘仅用其中5%的时间来传送数据，则处理器花费在I/O方面的百分比为25%x5%=1.25% 。

例：中断、DMA方式下CPU的开销

设处理器按500MHz的速度执行，硬盘控制器中有一个16B的数据缓存器，磁盘传输速率为4MB/Sec，在磁盘传输数据过程中，要求没有任何数据被错过，并假定CPU访存和DMA访存没有冲突。

（1）若用中断方式，每次传送的开销（包括用于中断响应和处理的时间）是500个时钟周期。如果硬盘仅用5%的时间进行传送，那么处理器用在硬盘I/O操作上所花的时间百分比（主机占用率）为多少?

（2）若用DMA方式，处理器用1000个时钟进行DMA传送初始化，在DMA完成后的中断处理需要500个时钟。如果每次DMA传送8000B的数据块，那么当硬盘进行传送的时间占100%（即：硬盘一直进行读写，并传输数据）时，处理器用在硬盘I/O操作上的时间百分比（主机占用率）为多少?

- DMA传送：一秒钟内有4MB/8000B=500次DMA传送
  - 每次DMA传送将花费8000B/(4MB/Sec)≈2×10^{-3}秒；
  - 一秒钟内有$1/(2\times10^{-3})=500$次DMA传送；
  - 如果硬盘一直在传送数据的话，处理器必须每秒钟花$(1000+500)\times500=750\times10^{3}$个时钟周期来为硬盘I/O操作服务；
  - 在硬盘I/O操作上处理器花费的时间占：

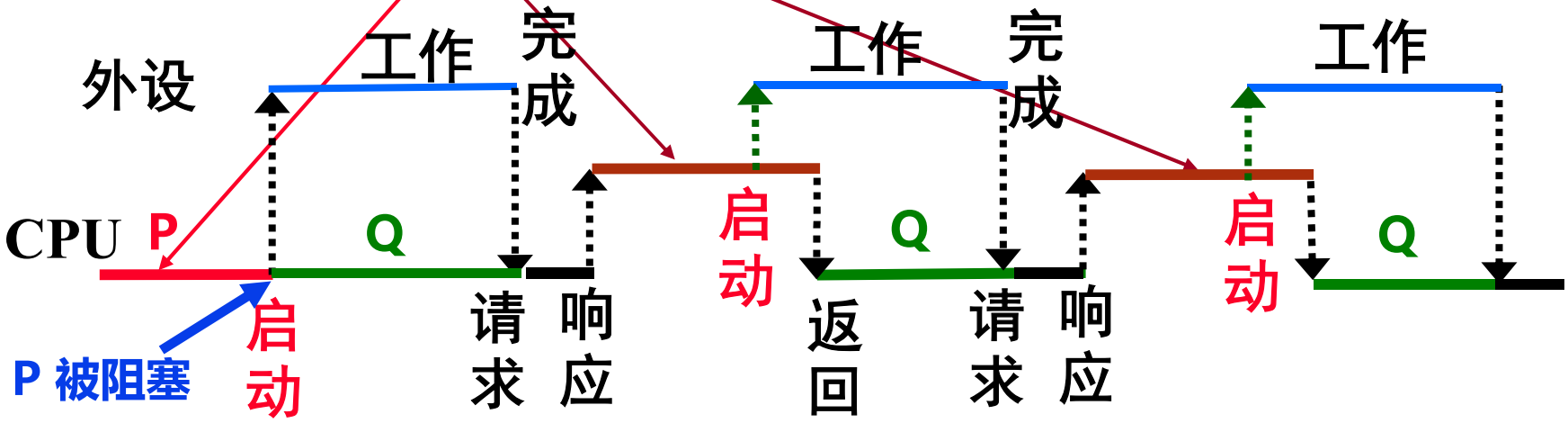
$$750\times10^{3}/(500\times10^{6})=1.5\times10^{-3}=0.15\%$$

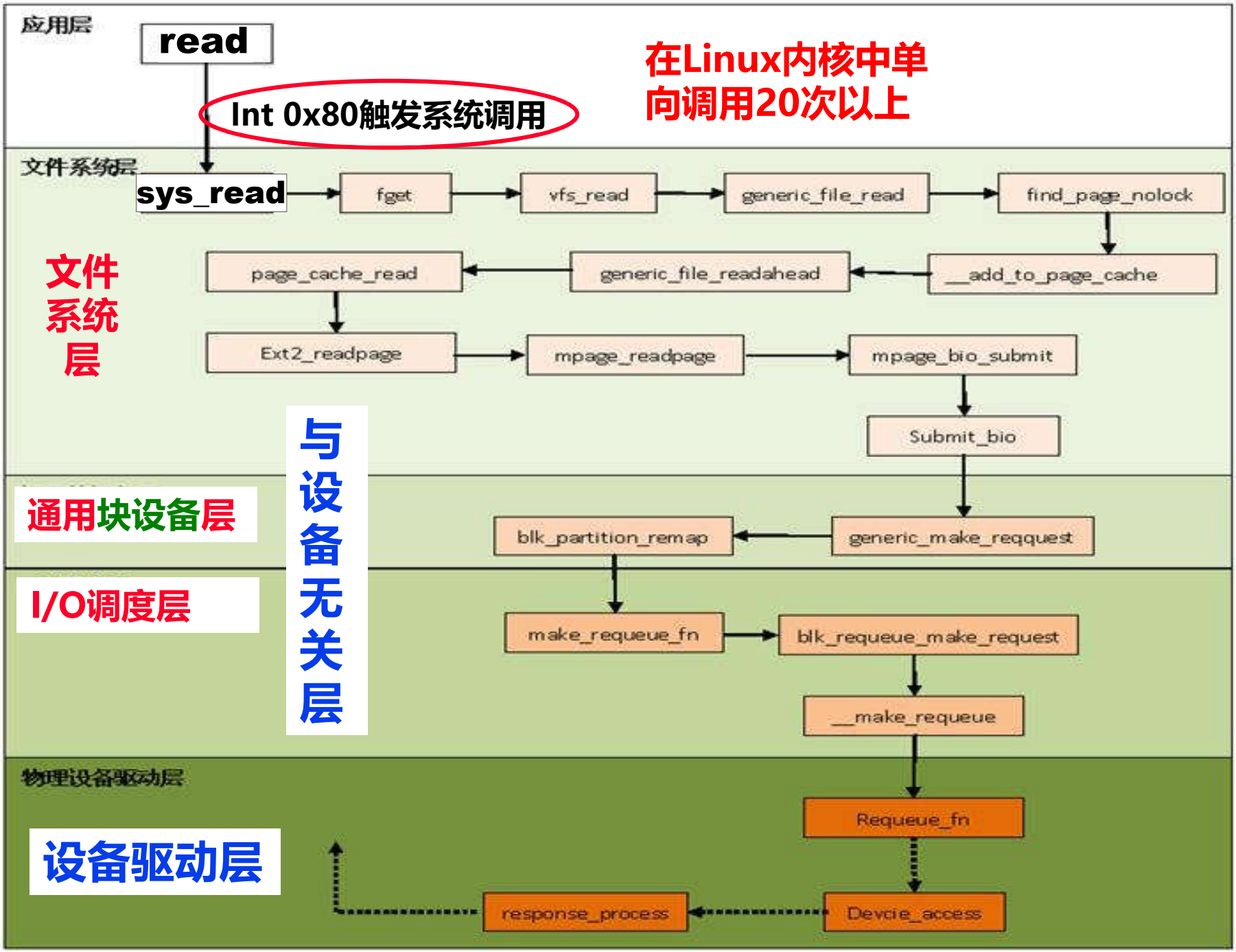
内核空间I/O软件

- 所有用户程序提出的I/O请求，最终都通过系统调用实现
- 通过系统调用封装函数中的陷阱指令转入内核I/O软件执行
- 内核空间I/O软件实现相应系统调用的服务功能
- 内核空间的I/O软件分三个层次
  - 设备无关软件层
  - 设备驱动程序层
  - 中断服务程序层

  } 系统调用服务例程，被陷阱指令调出执行，一旦发送“启动”命令，则所代表的进程被送等待队列（即被阻塞）

- 设备驱动程序层、中断服务程序层与I/O硬件密切相关




应用层
read
Int 0x80触发系统调用
在Linux内核中单向调用20次以上
文件系统层
sys_read
fget
vfs_read
generic_file_read
find_page_nolock
文件系统层
page_cache_read
generic_file_readahead
__add_to_page_cache
Ext2_readpage
mpage_readpage
mpage_bio_submit
Submit_bio
通用块设备层
与设备无关层
blk_partition_remap
generic_make_reqquest
I/O调度层
make_requeue_fn
blk_requeue_make_request
__make_requeue
物理设备驱动层
Requeue_fn
设备驱动层
response_process
Devcie_access

设备无关I/O软件层

- 设备驱动程序统一接口
  - 操作系统为所有外设的设备驱动程序规定一个统一接口，这样，新设备的驱动程序只要按统一接口规范来编制，就可在不修改操作系统的情况下，添加新设备驱动程序并使用新的外设进行I/O。
  - 所有设备都抽象成文件，设备名和文件名在形式上没有差别，设备和文件具有统一的接口，不同设备名和文件名被映射到对应设备驱动程序。
- 缓冲处理
  - 每个设备的I/O都需使用内核缓冲区，因而缓冲区的申请和管理等处理是所有设备公共的，可包含在与设备无关的I/O软件部分
- 错误报告
  - I/O操作在内核态执行时所发生的错误信息，都通过与设备无关的I/O软件返回给用户进程，也即：错误处理框架与设备无关。
  - 直接返回编程等错误，无需设备驱动程序处理，如，请求了不可能的I/O操作；写信息到一个输入设备或从一个输出设备读信息；指定了一个无效缓冲区地址或者参数；指定了不存在的设备等。
  - 有些错误由设备驱动程序检测出来并处理，若驱动程序无法处理，则将错误信息返回给设备无关I/O软件，再由设备无关I/O软件返回给用户进程，如写一个已被破坏的磁盘扇区；打印机缺纸；读一个已关闭的设备等。

设备无关I/O软件层

- 打开与关闭文件
  - 对设备或文件进行打开或关闭等I/O函数所对应的系统调用，并不涉及具体的I/O操作，只要直接对主存中的一些数据结构进行修改即可，这部分工作也由设备无关软件来处理。
- 逻辑块大小处理
  - 为了为所有的块设备和所有的字符设备分别提供一个统一的抽象视图，以隐藏不同块设备或不同字符设备之间的差异，与设备无关的I/O软件为所有块设备或所有字符设备设置统一的逻辑块大小。
  - 对于块设备，不管磁盘扇区和光盘扇区有多大，所有逻辑数据块的大小相同，这样，高层I/O软件就只需处理简化的抽象设备，从而在高层软件中简化了数据定位等处理 。

设备驱动程序

- 每个外设具体的I/O操作需通过执行设备驱动程序来完成
- 外设种类繁多、其控制接口不一，导致不同外设的设备驱动程序千差万别，因而设备驱动程序与设备相关
- 每个外设或每类外设都有一个设备控制器，其中包含各种I/O端口。 CPU通过执行设备驱动程序中的I/O指令访问各种I/O端口
- 设备所采用的I/O控制方式不同，驱动程序的实现方式也不同
  - 程序直接控制：驱动程序完成用户程序的I/O请求后才结束。这种情况下，用户进程在I/O过程中不会被阻塞，内核空间的I/O软件一直代表用户进程在内核态进行I/O处理 。（干等！）
  - 中断控制：驱动程序启动第一次I/O操作后，将调出其他进程执行，而当前用户进程被阻塞。在CPU执行其他进程的同时，外设进行I/O操作，此时，CPU和外设并行工作。外设完成I/O时，向CPU发中断请求，然后CPU调出相应中断服务程序执行。在中断服务程序中再次启动I/O操作。
  - DMA控制：驱动程序对DMA控制器初始化后，便发送“启动DMA传送”命令，外设开始进行I/O操作并在外设和主存间传送数据。同时CPU执行处理器调度程序，转其他进程执行，当前用户进程被阻塞。DMA控制器完成所有I/O任务后，向CPU发送一个“DMA完成”中断请求信号。

中断服务程序

- 中断控制和DMA控制两种方式下都需进行中断处理
- **中断控制方式**：中断服务程序主要进行从数缓器取数或写数据到数缓器，然后启动外设工作
- **DMA控制方式**：中断服务程序进行数据校验等后处理工作

在内核I/O软件中用到的I/O指令、“开中断”和“关中断”等指令都是特权指令，只能在操作系统内核程序中使用