操作系统实验三
I/O系统

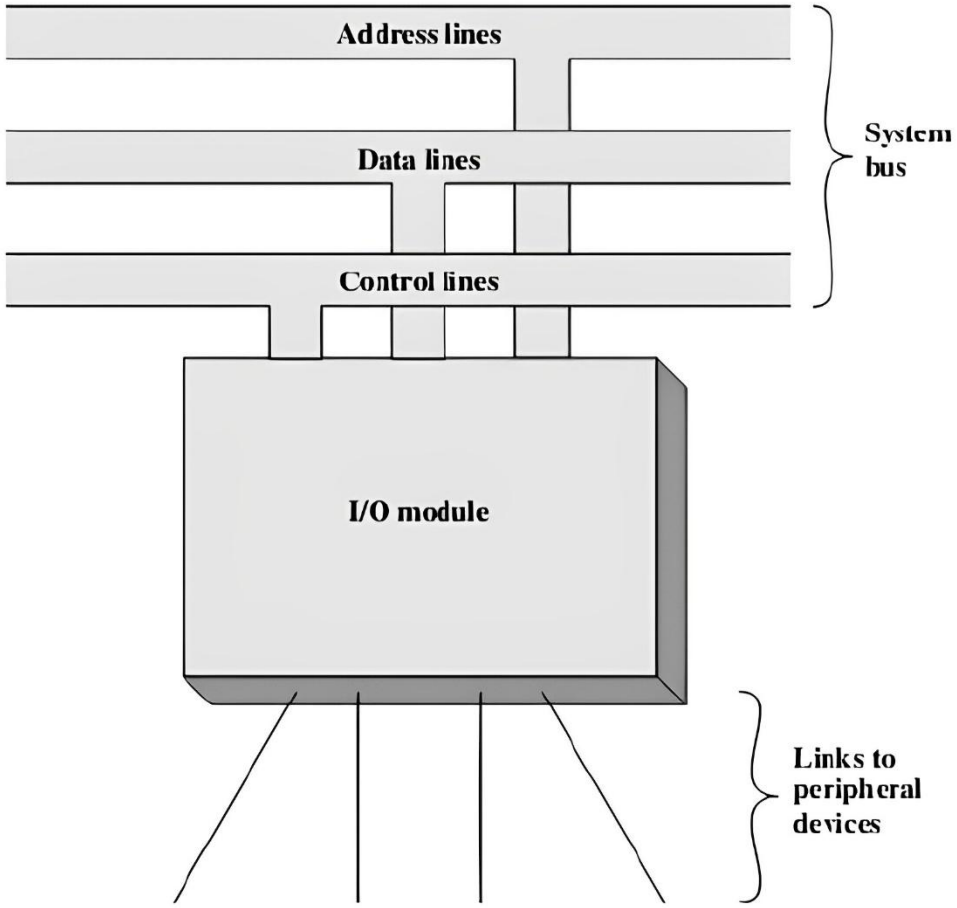
南京大学软件学院

I/O

- 输入和输出
- 最简单也是最原始的和计算机交互的方式
- 输入
- 用户向计算机发送的请求和数据
- 输出
- 计算机给予用户的反馈
- 依赖
- 依靠外接设备
  - 输入依靠键盘(和鼠标…)
  - 输出依靠显示屏(和打印机…)
- 计算机需要为这些外接设备提供接口

控制外接设备

- CPU接口对外设进行控制的方式：
- 程序查询方式(轮询)
- 中断方式
- DMA方式
  - 直接存储器存取，内存和设备由数据通路成块地传送数据，传输过程无须CPU干预.

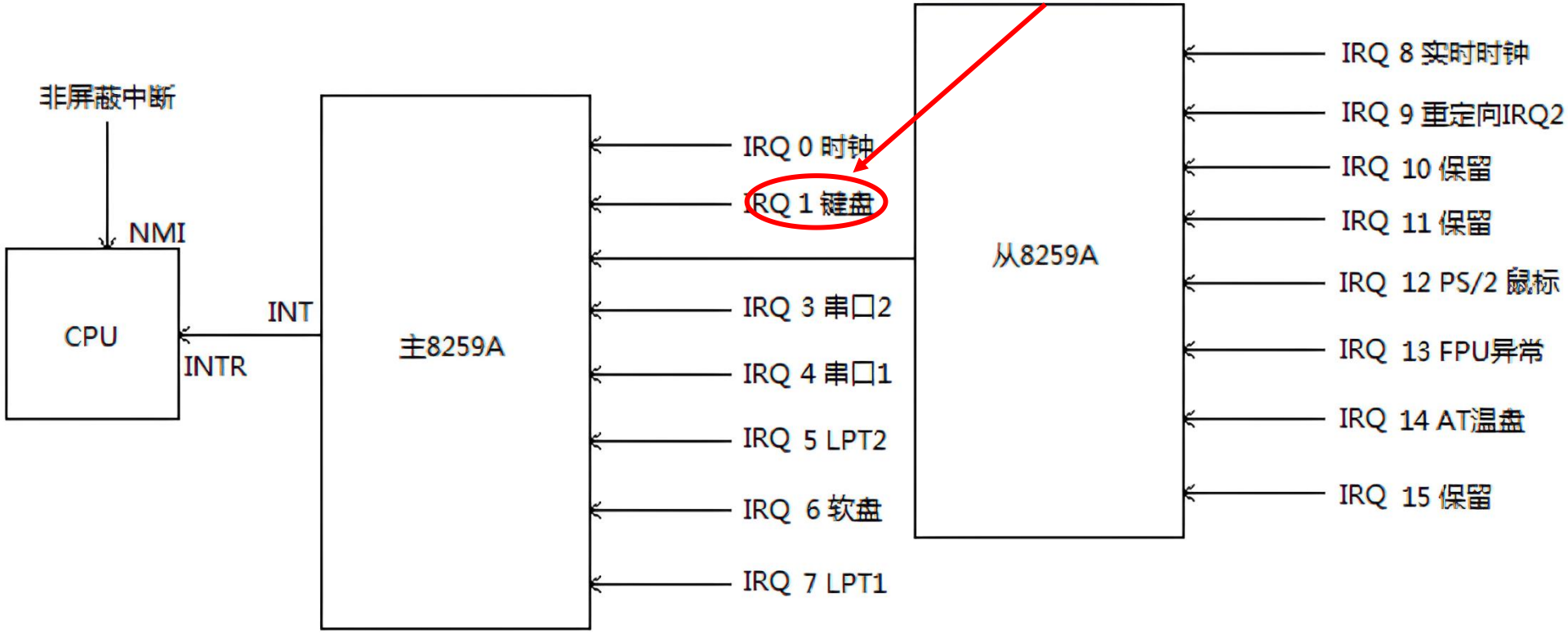


敲击键盘的含义

- 敲击键盘有两方面的含义：三类动作& 两种编码
- 三类动作：按下，保持按住以及放开
- 两种编码——扫描码(Scan Code)
  - Make Code——当一个键被按下或者保持住按下时会产生Make Code
  - Break Code——当一个键弹起时，产生Break Code
- 键盘上不同的键，无论字母键还是数字键，回车键还是箭头键， 每个键按下与弹起都对应不同的扫描码

从中断开始

- 8259A为进行中断控制而设计的芯片，它是可以用程序控制的中断控制器
- 在8259A中 中断请求1 对应的就是键盘



键盘和主机的连接

- 在键盘中存在一枚叫做键盘编码器(Keyboard Encoder)的芯片8048，用于监视键盘的输入把适当的数据传送给计算机；

- 在计算机主板上还有一个键盘控制器(Keyboard Controller)的芯片8042，用于接收和解码来自键盘的数据，并与8259A以及软件等进行通信

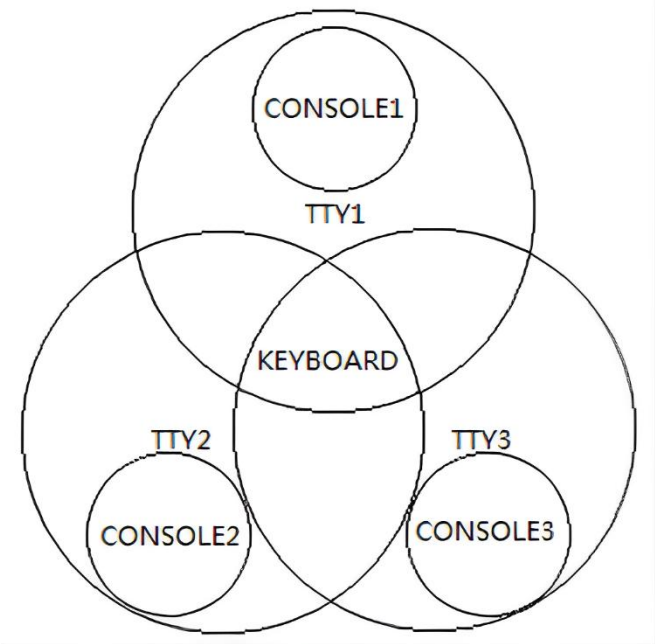
敲击键盘后的变化

- 当8048(键盘编码器，Intel 8048及兼容芯片)检测到一个键的动作后，会把相应的扫描码发给8042(键盘控制器)，8042会把它转换成相应的某套扫描码将其放置在缓冲区
- 然后8042告诉8259A产生中断
- 响应中断，键盘中断程序被执行，对缓冲区数据进行处理，这样8042才能继续响应新的按键

寄存器名称	寄存器大小	端口	R/W	用法
输出缓冲区	1BYTE	0x60	Read	读输出缓冲区
输入缓冲区	1BYTE	0x60	Write	写输入缓冲区
状态寄存器	1BYTE	0x64	Read	读状态寄存器
控制寄存器	1BYTE	0x64	Write	发送指令

来看看输出—— 终端

- 对于Linux或者UNIX的使用者，终端(TTY)一定不陌生,我们可以开启多个命令行窗口，在不同的窗口屏幕中，分别有不同的输入和输出,相互不受影响。屏幕上显示的是一些数据，存放这些显示出来的数据的内存段，我们就称其为显存。



显示器(视频)

- 在最初我们是通过BIOS中断来实现的，但是到了保护模式BIOS不能再使用
  - 我们就在GDT中建立一个段,它的开始地址是0xB8000，通过段寄存器gs对它进行写操作，从而实现数据的显示，这个段就是显存；
- Orange's中的系统开机看到的默认模式是80*25文本模式；
  - 在这种模式下，显存大小为32KB
  - 每两个字节代表屏幕上的一个字符，其中低字节表示字符的ASCII码，高字节表示字符的属性(背景，前景的RGB值以及是否高亮)
- 一个屏幕在显存中占80×25×2 = 4000字节，32KB可以存放8个屏幕的数据
  - 所以我们可以设置三个终端TTY, 每个终端使用10KB的显存

I/O控制权限

- 对I/O的控制权限是很重要的一项内容，保护模式对此也做了限制，用户进程如果不被许可是无法进行I/O操作的。
- 这种限制通过两个方面来实现，IOPL和I/O许可位图

IOPL (I/O Privilege Level)

- 它位于寄存器eflags的第12，13位

保留(设为0)	ID	VIP	VTF	AC	VM	RF	0	NT	IOPL	OF	DF	IF	TF	SF	ZF	0	AF	0	PF	1	CF

- 指令in、ins、out、outs、cli、sti只有在CPL<=IOPL时才能执行
- 这些指令被称作I/O敏感指令，如果低特权级的指令试图访问这些I/O敏感指令将会导致常规保护错误(#GP)

I/O许可位图(I/O Permission Bitmap)

- 之所以叫位图，是因为它的每一位表示一个字节的端口地址是否可用。
- 如果一位是0，表示此位对应的端口可用，1则不可用

- 由于每一个任务都可以有单独的TSS,所以每一个任务可以有它单独的I/O许可位图,这样就能对每一个任务制定不同的I/O权限
- 如果I/O位图基址大于或等于TSS段界限，就表示没有I/O许可位图，如果CPL<=IOPL，则所有I/O指令都会引起异常。

- 可以看出，I/O许可位图的使用使得即便在同一特权级下不同的任务也可以有不同的I/O访问权限

Thanks !