第二章　进程管理

翁楚良

https://chuliangweng.github.io

2023 春　ECNU

第二章进程管理 提纲

- 2.1 进程
- 2.2 进程间通信
- 2.3 经典并发问题
- 2.4 进程调度
- 2.5 MINIX3进程概述
- 2.6 MINIX3进程实现
- 2.7 MINIX3系统任务
- 2.8 MINIX3时钟任务

MINIX3时钟任务

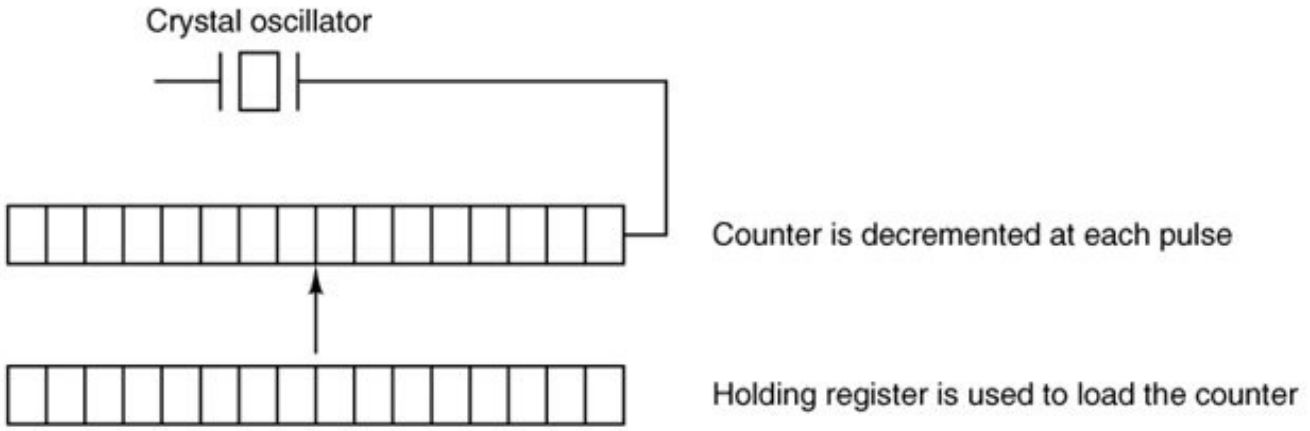
- 时钟（也称为定时器）对于任何分时操作的系统都是很重要的
- 维护日常时间，并且防止某一进程独占CPU而不让其他实体使用CPU
- 时钟既不是象磁盘一样的块设备，也不是象终端一样的字符设备
- 时钟任务类似于设备驱动程序由硬件设备的中断驱动

MINIX3时钟任务

- 时钟硬件
- 计时程序
- 时钟驱动程序概述
- 时钟驱动程序实现

可编程时钟

- 由三个元件组成：晶振、计数器和一个保持寄存器。
- 当把石英晶体进行合适的切削并安装于一定的压力之下时，它会产生非常精确的周期信号。根据所选晶体的不同，其典型范围在5至200M之间。
- 在任何计算机系统中，都至少可以发现一个这样的电路向计算机的各种电路提供同步信号。把这个信号送入计数器并使其递减计数至零，当计数至零时，产生一个中断。



可编程模式

- 单触发模式（One-shot Mode），
  - 当一个时钟启动时，它把保持寄存器的值拷贝到计数器中，然后每从晶振来一个脉冲，对计数器值减一，当计数器为零时，产生一个中断，并停止工作直到再次被程序启动为止。
- 方波模式（Square-wave Mode）
  - 每次计数至零并引起中断以后，保持寄存器自动拷贝到计数器，整个过程不断重复重复执行。这些周期性的中断称为时钟滴答（Clock Tick）。

MINIX3时钟任务

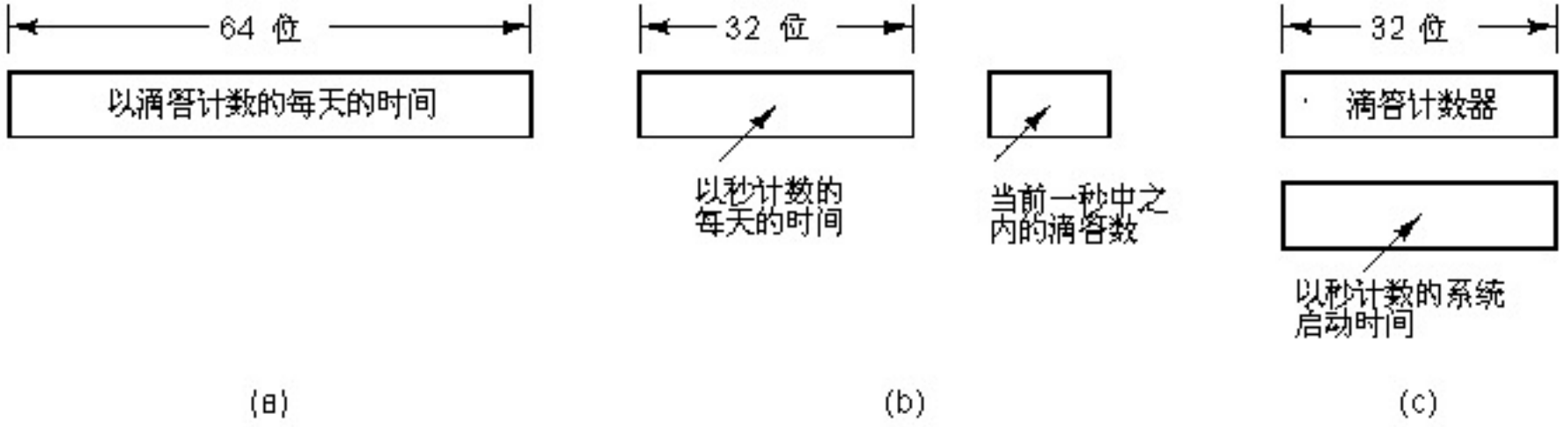
- 时钟硬件
- 计时程序
- 时钟驱动程序概述
- 时钟驱动程序实现

时钟任务的作用

- 维护日期时间。
- 防止进程的运行时间超出其允许的时间。
- 审计CPU使用时间。
- 处理用户进程提出的alarm系统调用。
- 对系统某些部分提供看门狗时钟。
- 执行测试统计、监视获取统计数据。

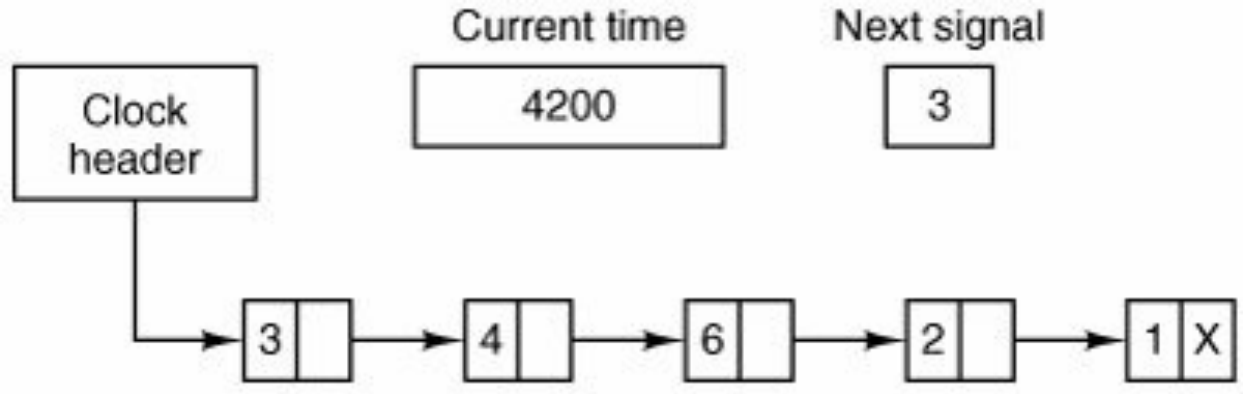
日常时间维护方法

- 是使用64位计数器。因为1秒以内需执行多次维护计数器的工作，因此提高了维护时钟的代价。
- 用秒作为单位来维护日常时间，而不是节拍数，在每秒内使用一个辅助计数器来计节拍数
- 按滴答计数，但是相对系统启动时间，而不是相对一个确定的外部时刻。当读后备时钟或用户输入实际时间时，系统启动时间就可以得到，并存储于系统中。之后，存储的时间值加上计数器的时间值，就可以得到当前的时间。



使用单个时钟模拟多个定时器

- MINIX3中，进程可以要求操作系统在间隔一定时间后给它一个alarm
- 可以用一个物理时钟模拟多个虚拟时钟
- 把所有的时钟请求在一个链表中连接起来并按时间排序，以此模拟多个时钟将更加有效。链表中的每个入口指出前一信号发生以后需要等待多少个滴答才引发下一个信号。在这个例子中，信号发生在 4203、4207、4213、4215、4216。



看门狗时钟

- 操作系统的有些部分也需要定时器，这些是所谓的看门狗时钟（watch-dog Timer）。
  - 每次向硬盘控制器发命令时，都要安排唤醒调用，以便当命令执行完全失败时，可以试图进行恢复。
- 时钟驱动程序处理看门狗的机制和对待用户信号的机制是一样的。唯一的不同是当定时器时间到时，时钟驱动程序将调用一个调用者提供的过程而不是产生一个信号，这个过程是调用者代码的一部分。

MINIX3时钟任务

- 时钟硬件
- 计时程序
- 时钟驱动程序概述
- 时钟驱动程序实现

时钟驱动程序概述

- 包含在kernel/clock.c中
  - 主循环，等待消息然后指定相应的子程序执行
  - 中断处理程序，维护基本的日常时间，更新记录系统启动以来的时钟节拍数的变量，与下次定时器到时的时间比较等。若检测到进程完成它的时间片等，发送返回主循环的消息
  - 一系列起支持作用的子程序

时钟驱动程序提供的服务

Service	Access	Response	Clients
get_uptime	Function call	Ticks	Kernel or system task
set_timer	Function call	None	Kernel or system task
reset_timer	Function call	None	Kernel or system task
read_clock	Function call	Count	Kernel or system task
clock_stop	Function call	None	Kernel or system task
Synchronous alarm	System call	Notification	Server or driver, via system task
POSIX alarm	System call	Signal	User process, via PM
Time	System call	Message	Any process, via PM

MINIX3时钟任务

- 时钟硬件
- 计时程序
- 时钟驱动程序概述
- 时钟驱动程序实现

```
/*===========================================================================*
 *                                clock_task                                 *
 *===========================================================================*/
PUBLIC void clock_task()
{
/* Main program of clock task. If the call is not HARD_INT it is an error.
 */
  message m;                    /* message buffer for both input and output */
  int result;                   /* result returned by the handler */

  init_clock();                 /* initialize clock task */

  /* Main loop of the clock task.  Get work, process it. Never reply. */
  while (TRUE) {

      /* Go get a message. */
      receive(ANY, &m);

      /* Handle the request. Only clock ticks are expected. */
      switch (m.m_type) {
      case HARD_INT:
          result = do_clocktick(&m);    /* handle clock tick */
          break;
      default:                          /* illegal request type */
          kprintf("CLOCK: illegal request %d from %d.\n", m.m_type,m.m_source);
      }
```

```
/*===========================================================================*
 *                                init_clock                                 *
 *===========================================================================*/
PRIVATE void init_clock()
{
  /* Initialize the CLOCK's interrupt hook. */
  clock_hook.proc_nr = CLOCK;

  /* Initialize channel 0 of the 8253A timer to, e.g., 60 Hz. */
  outb(TIMER_MODE, SQUARE_WAVE);        /* set timer to run continuously */
  outb(TIMER0, TIMER_COUNT);            /* load timer low byte */
  outb(TIMER0, TIMER_COUNT >> 8);       /* load timer high byte */
  put_irq_handler(&clock_hook, CLOCK_IRQ, clock_handler);/* register handler */
  enable_irq(&clock_hook);              /* ready for clock interrupts */
}
```

第二章进程管理 提纲

- 2.1 进程
- 2.2 进程间通信
- 2.3 经典并发问题
- 2.4 进程调度
- 2.5 MINIX3进程概述
- 2.6 MINIX3进程实现
- 2.7 MINIX3系统任务
- 2.8 MINIX3时钟任务

作业 (操作系统设计与实现)

- 2, 4, 7, 8, 9, 17, 23, 26, 27, 28, 33, 38

第三章　I/O系统

翁楚良

https://chuliangweng.github.io

2023 春　ECNU

第三章 I/O系统 提纲

- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

I/O硬件分类

- 按交互对象分类
  - 人机交互设备：视频显示设备、键盘、鼠标、打印机
  - 与计算机或其他电子设备交互的设备：磁盘、磁带、传感器、控制器
  - 计算机间的通信设备：网卡、调制解调器
- 按交互方向分类
  - 输入（可读）：键盘、扫描仪
  - 输出（可写）：显示设备、打印机
  - 输入/输出（可读写）：磁盘、网卡
- 按外设特性分类
  - 使用特征：存储、输入/输出、终端
  - 数据传输率：低速(如键盘)、中速(如打印机)、高速(如网卡、磁盘)
  - 信息组织特征：单个字符或数据块
    - 字符设备(如打印机)
    - 块设备(如磁盘)

I/O设备的特点

- 种类多
- 差异大(控制和速度)
  - 在速率相差多个数量级的不同设备上保持良好的性能，这对操作系统提出了很高的要求

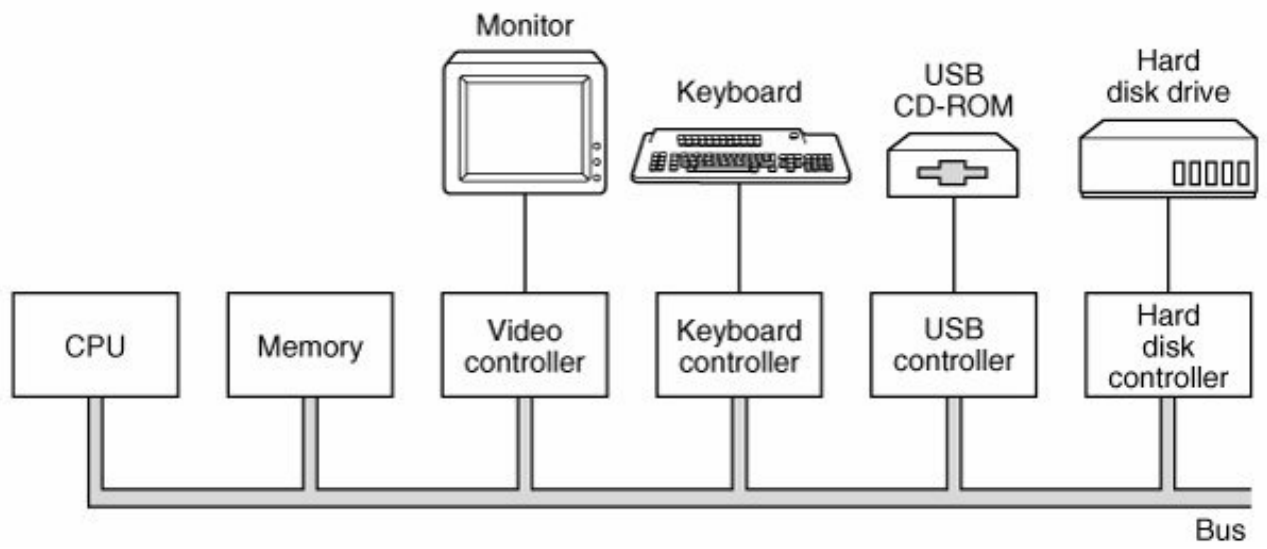
Device	Data rate
Keyboard	10 bytes/sec
Mouse	100 bytes/sec
56K modem	7 KB/sec
Scanner	400 KB/sec
Digital camcorder	4 MB/sec
52x CD-ROM	8 MB/sec
FireWire (IEEE 1394)	50 MB/sec
USB 2.0	60 MB/sec
XGA Monitor	60 MB/sec
SONET OC-12 network	78 MB/sec
Gigabit Ethernet	125 MB/sec
Serial ATA disk	200 MB/sec
SCSI Ultrawide 4 disk	320 MB/sec
PCI bus	528 MB/sec

块设备和字符设备

- 块设备将信息存储在可寻址的固定大小的数据块中
  - 通常数据块大小的范围从512字节到32768字节不等
  - 块设备的主要特征是能够独立地读写单个的数据块
  - 磁盘是最常见的块设备
- 字符设备发送或接收的是字符流，而不是块结构
  - 字符设备无法编址，也不存在任何寻址操作
  - 如：打印机、网络接口、鼠标等

设备控制器

- I/O设备通常包含一个机械部件和一个电子部件。为了达到设计的模块性和通用性，一般将其分开。电子部分称为设备控制器或适配器。在个人计算机中，它常常是一块可以插入主板扩展槽的印刷电路板。机械部分则是设备本身。



设备控制器

- 控制器负责将驱动器读出来的比特流转换成字节块并在需要时进行纠错。
- 通常该字节块是在控制器中的一个缓冲区中逐个比特汇集而成。在对检查和进行校验证实数据正确之后，该块数据随后被拷贝到主存中。

I/O控制技术

- 程序控制I/O
- 中断驱动方式
- 直接存储访问方式
- 通道控制方式

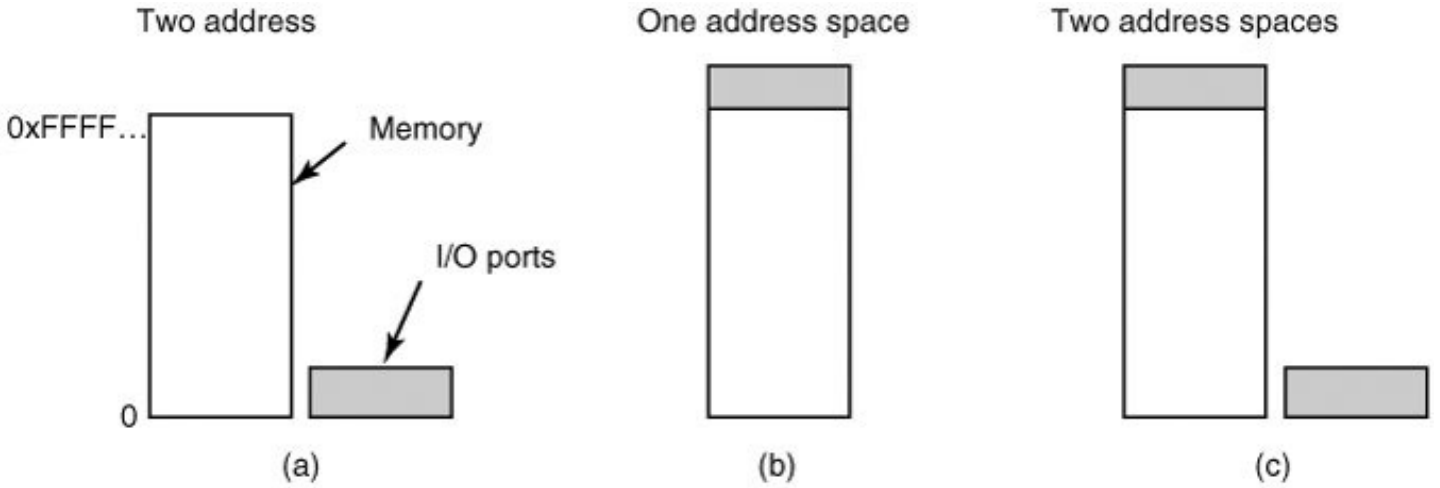
程序控制I/O

- I/O操作由程序发起，并等待操作完成。数据的每次读写通过CPU。
- 每个控制器都有一些用来与CPU通信的寄存器及数据缓冲区。
  - 通过写入寄存器，操作系统可以命令设备发送数据、接收数据、开启或关闭、或其它操作
  - 通常设备有一个数据缓冲区，以供操作系统读写数据

程序控制I/O

- 设备编址方式
  - I/O端口
    - 设备寄存器单独编址，即I/O端口号
  - 内存映射I/O
    - 设备数据缓冲区按内存地址空间进行统一编址

IN REG, PORT
OUT PORT, REG



IBM PC兼容机：0-64KB作为I/O端口；640KB-1MB保留给数据缓冲区

I/O控制技术

- 程序控制I/O
- 中断驱动方式
- 直接存储访问方式
- 通道控制方式

中断驱动方式

- I/O操作是否可以开始或完成
  - 检测设备控制寄存器中的状态标志位
  - 使用中断方式通知CPU
- 中断驱动方式
  - I/O操作由程序发起，在操作完成时（如数据可读或已经写入）由外设向CPU发出中断，通知该程序。数据的每次读写通过CPU。
  - 优点
    - 在外设进行数据处理时，CPU不必等待，可以继续执行该程序或其他程序。
  - 缺点
    - CPU每次处理的数据量少（通常不超过几个字节），只适于数据传输率较低的设备。

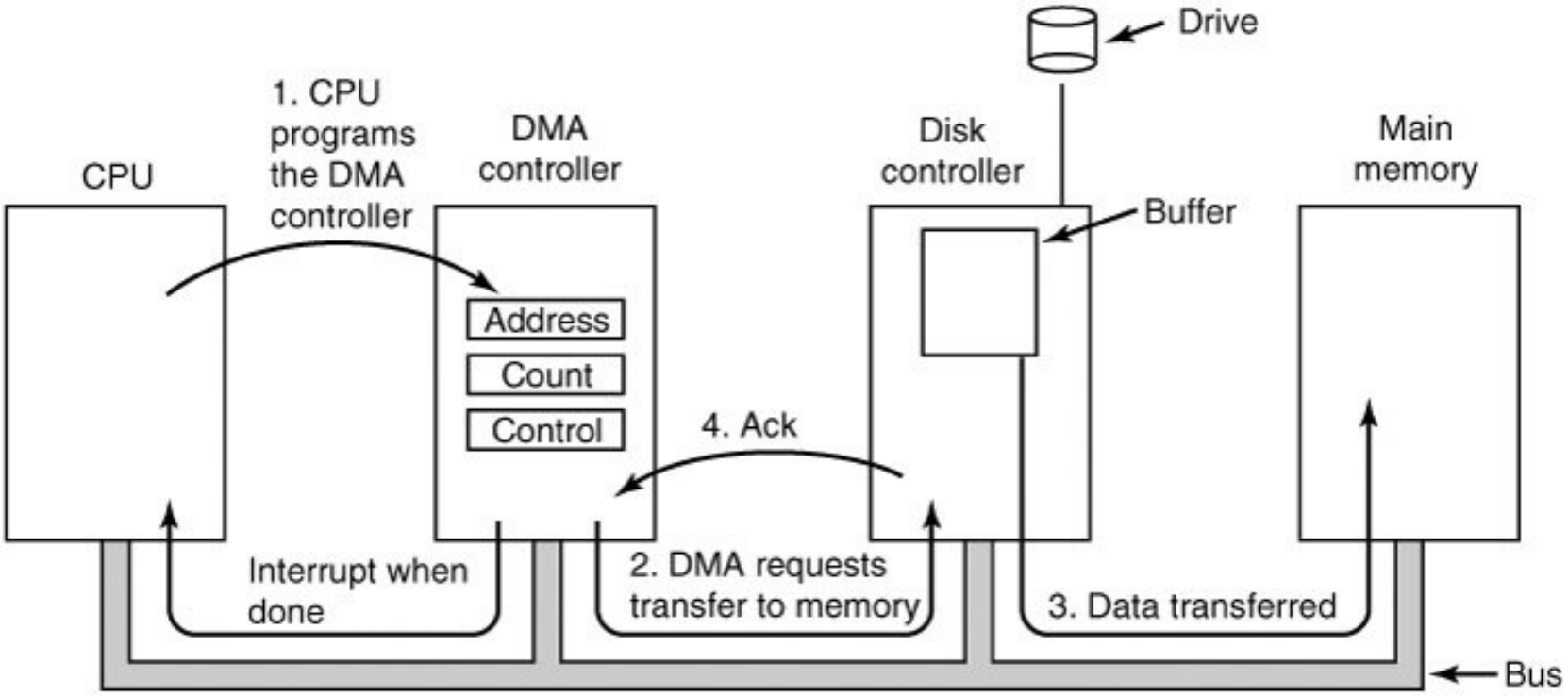
I/O控制技术

- 程序控制I/O
- 中断驱动方式
- 直接存储访问方式
- 通道控制方式

直接存储访问方式

- 由程序设置DMA(Direct Memory Access)控制器中的若干寄存器值（如内存始址，传送字节数），然后发起I/O操作，而后者完成内存与外设的成批数据交换，在操作完成时由DMA控制器向CPU发出中断。
- 优点
  - CPU只需干预I/O操作的开始和结束，而其中的一批数据读写无需CPU控制，适于高速设备。

DMA传送操作

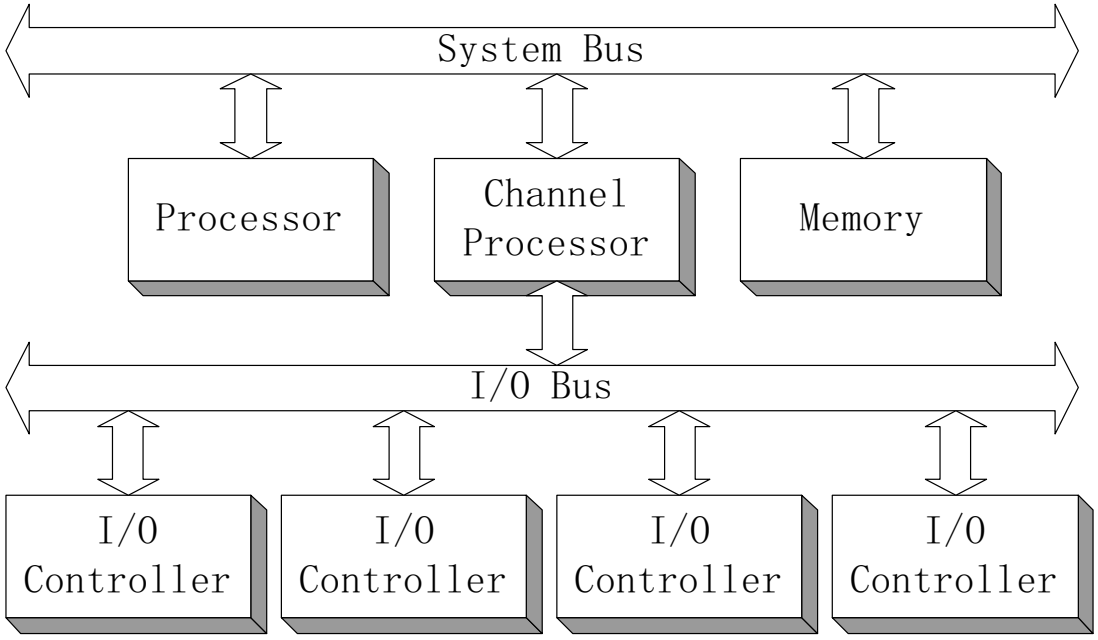

Drive
1. CPU programs the DMA controller
DMA controller
Disk controller
Main memory
CPU
Buffer
Address
Count
Control
4. Ack
Interrupt when done
2. DMA requests transfer to memory
3. Data transferred
Bus

I/O控制技术

- 程序控制I/O
- 中断驱动方式
- 直接存储访问方式
- 通道控制方式

通道控制方式

- 通道控制器(Channel Processor)有自己的专用存储器，可以执行由通道指令组成的通道程序，因此可以进行较为复杂的I/O控制。
- 通道程序通常由操作系统所构造，放在内存里。
- 优点：执行一个通道程序可以完成几批I/O操作



选择通道(selector channel)：可以连接多个外设，而一次只能访问其中一个外设

多路通道(multiplexor channel)：可以并发访问多个外设。分为字节多路(byte)和数组多路(block)通道。