第9章 设备管理

- 本章内容
- 1. 设备管理的基本概念
- 2. 大容量存储器（机械硬盘与固态硬盘）
- 3. 时钟
- 4. I/O系统的层次架构
- 5. 设备驱动程序
- 6. 中断处理程序
- 7. 设备独立内核软件

思考：程序如何与硬件交互？

用户视角：

Printf(“Hello”);

硬件视角：

- UART芯片
- 显卡
- 磁盘控制器

- 核心问题：
  - CPU如何控制设备？
  - 设备如何通知CPU？
  - 如何统一管理不同设备？

I/O设备分类

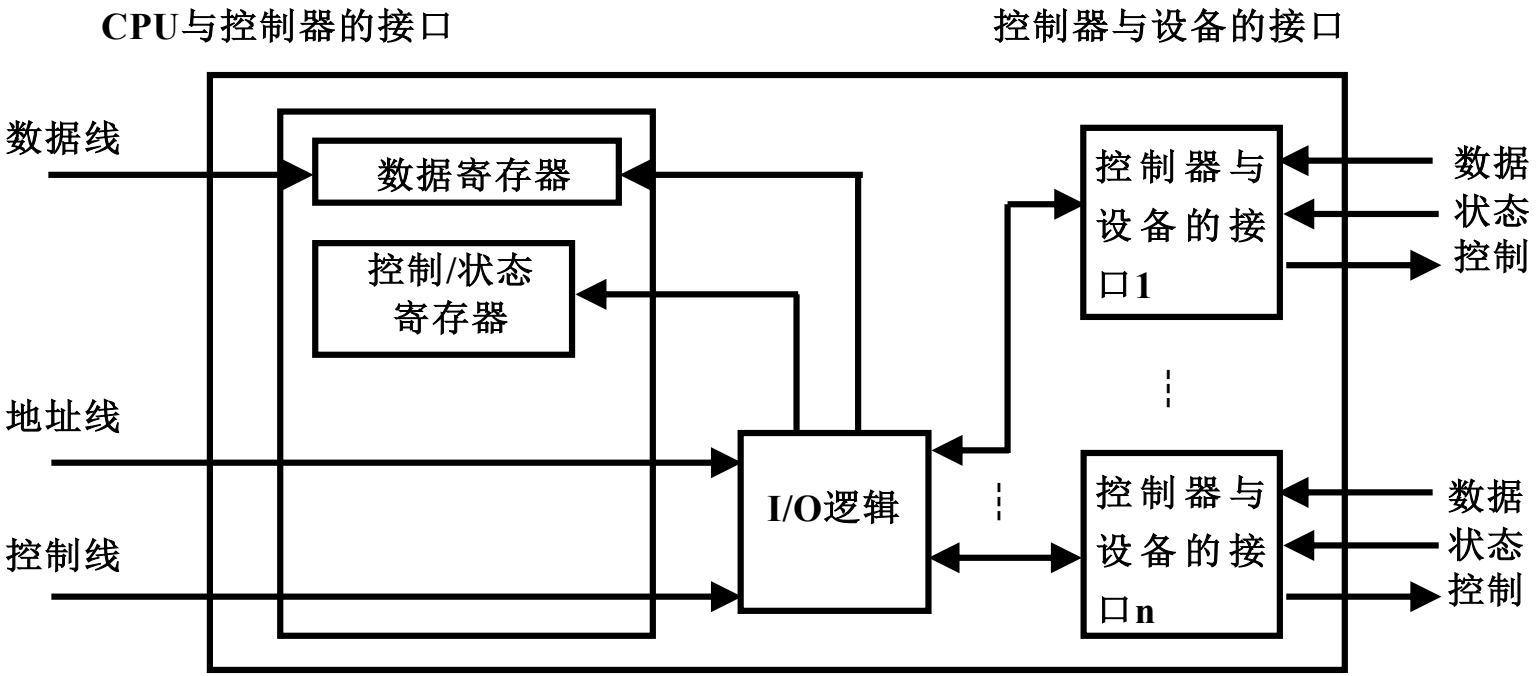
- **按使用特性：**
  - 人机交互设备：键盘、鼠标、显示器
  - 存储设备：硬盘、U盘、光驱
  - 网络通信设备：网卡、调制解调器
- **按传输速率：**
  - 低速：键盘（10B/s）、鼠标（100B/s）
  - 中速：打印机（100KB/s）
  - 高速：磁盘（100MB/s）、网卡（1GB/s）
- **按信息交换单位：**
  - 块设备：硬盘、U盘（可随机访问）
  - 字符设备：键盘、串口（流式访问）
  - 网络设备：网卡（数据包）

设备控制器

- I/O设备通常由**机械部件**（物理设备）和**电子部件**（控制器）组成。
- **"双面"接口模型：**设备控制器是CPU与物理设备之间的桥梁。
  - 1. **面向CPU的接口**（即I/O接口）：
    - 通过系统总线连接
    - 包含**I/O端口**（寄存器）：数据、状态、控制
    - 作用：接收CPU的高级指令，提供数据缓冲
  - 2. **面向设备的接口**（即设备接口）：
    - 通过专用电缆连接硬件
    - 发送底层电子信号（如驱动电机、控制磁头）
    - 作用：屏蔽物理细节，执行具体动作
- **I/O端口的三种类型：**

数据端口 (Data)	存放输入/输出数据 (缓冲)
状态端口 (Status)	指示设备忙/闲、错误 (握手)
控制端口 (Ctrl)	接收启动、复位等命令

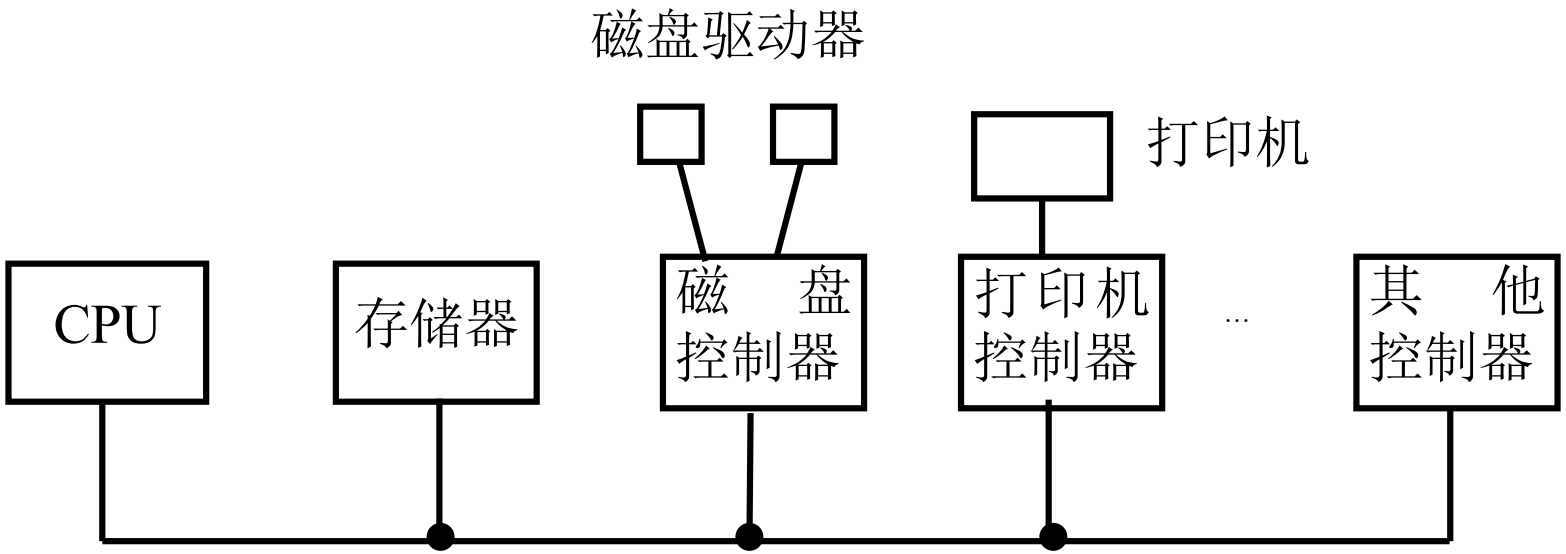
设备控制器组成图



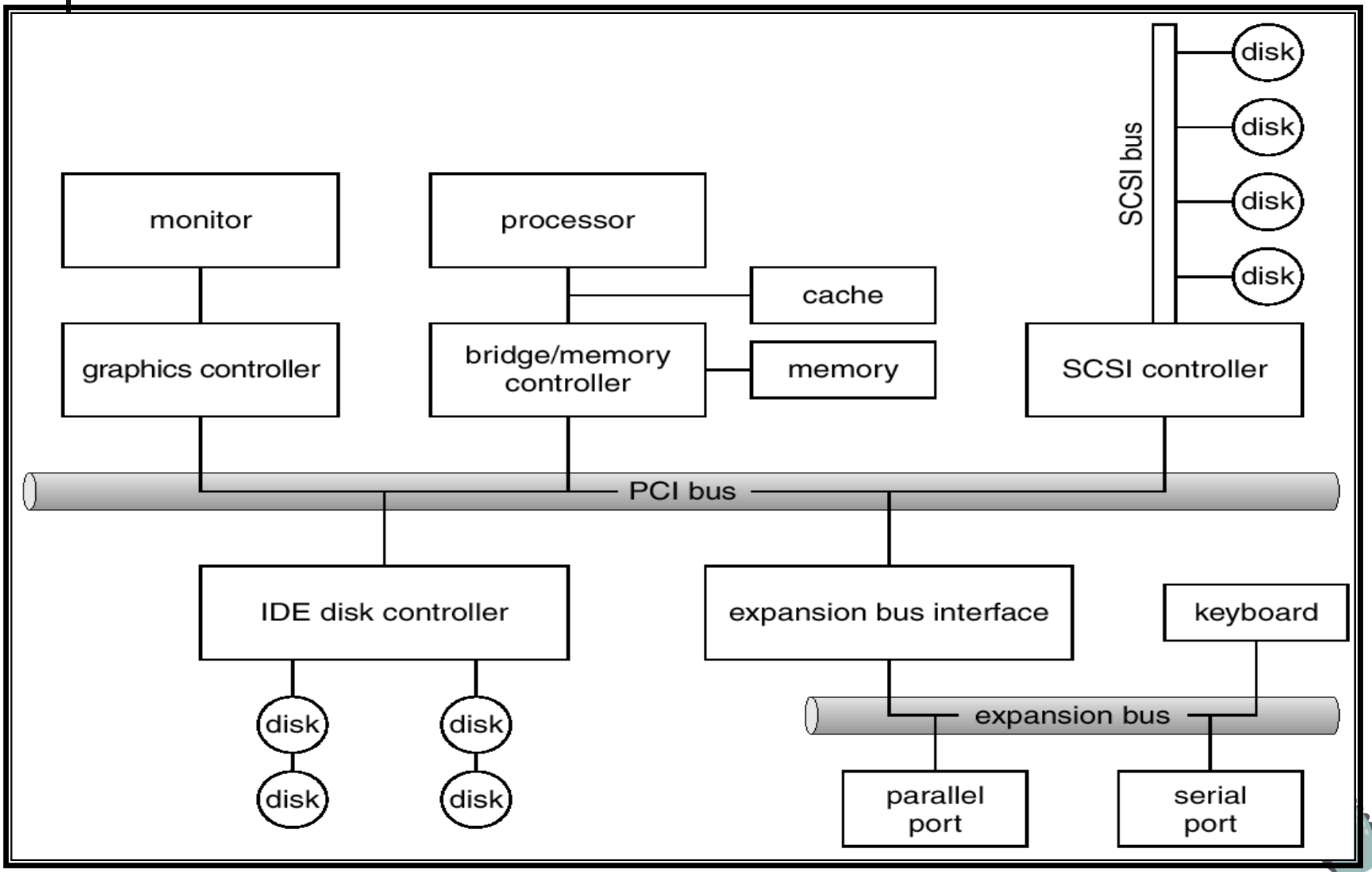
- **CPU与控制器的接口**：实现CPU与设备控制器之间的通信。
- **控制器与设备的接口**：实现设备与设备控制器之间的通信。
- **I/O逻辑**：实现对设备的控制，它负责接收命令、对命令进行译码、再根据译出的命令控制设备。

总线I/O系统结构

- CPU和内存直接连接到总线上，I/O设备通过设备控制器连接到总线上。



典型的PC总线结构

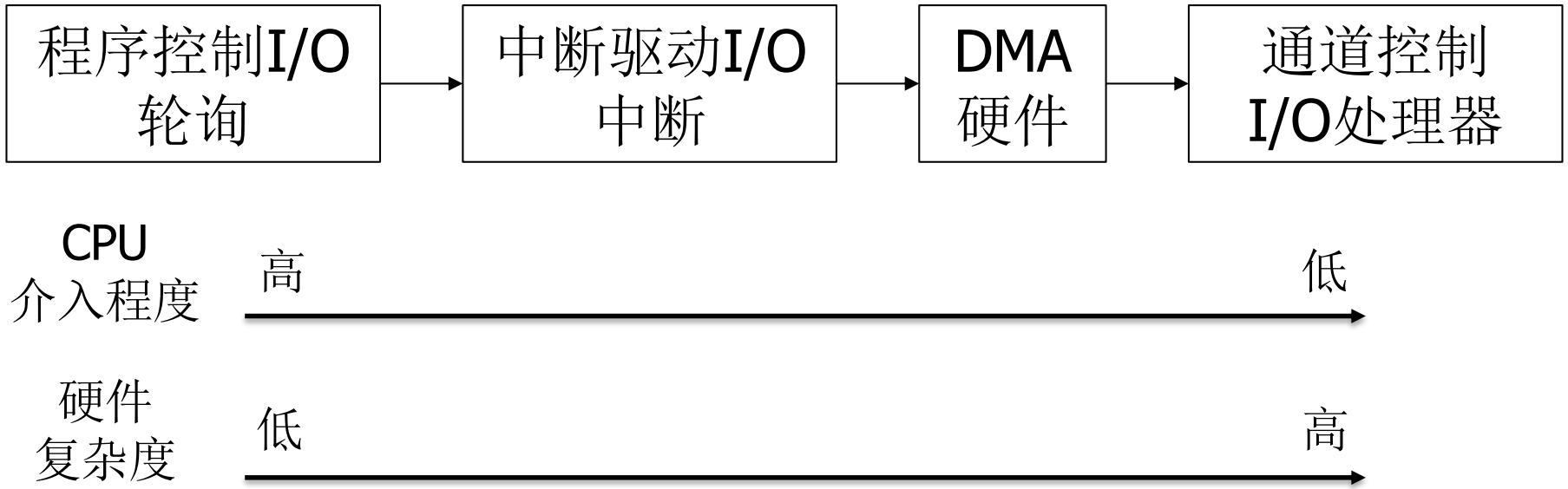

monitor
processor
cache
graphics controller
bridge/memory controller
memory
SCSI bus
disk
disk
disk
disk
SCSI controller
PCI bus
IDE disk controller
expansion bus interface
keyboard
disk
disk
disk
disk
expansion bus
parallel port
serial port

PC中的设备I/O端口位置（部分）

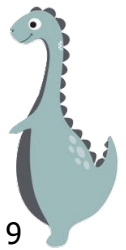
I/O address range (hexadecimal)	device
000-00F	DMA controller
020-021	interrupt controller
040-043	timer
200-20F	game controller
2F8-2FF	serial port (secondary)
320-32F	hard-disk controller
378-37F	parallel port
3D0-3DF	graphics controller
3F0-3F7	diskette-drive controller
3F8-3FF	serial port (primary)

I/O控制方式概览

四种I/O控制方式对比



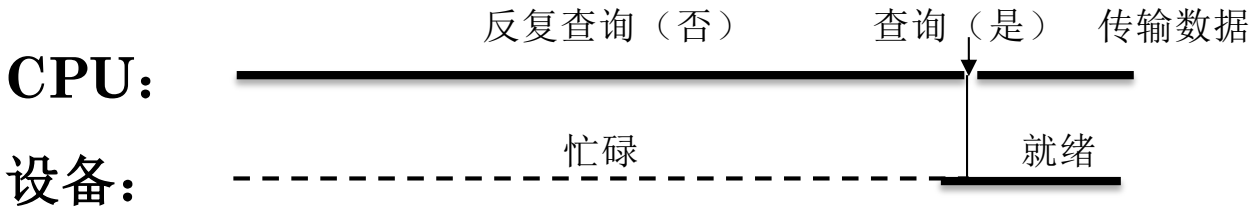
逐步解放CPU



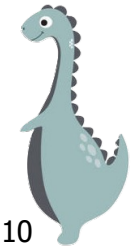
程序控制I/O（轮询）

- 工作原理

```
// CPU不断查询设备状态
while (!(inb(STATUS_PORT) & READY))
    ; // 等待设备就绪
outb(DATA_PORT, data); // 写入数据
```

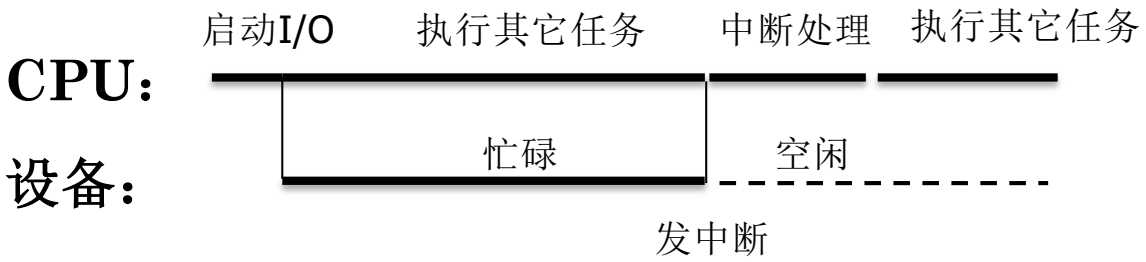


- 优缺点：
  - 简单易实现
  - CPU利用率极低
  - 适用于低速、简单设备

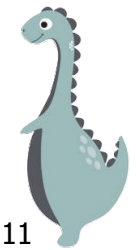


中断驱动I/O

- 工作原理
  - 1. CPU发起I/O请求后继续执行其他任务
  - 2. 设备完成操作后发送中断信号
  - 3. CPU响应中断，处理I/O结果

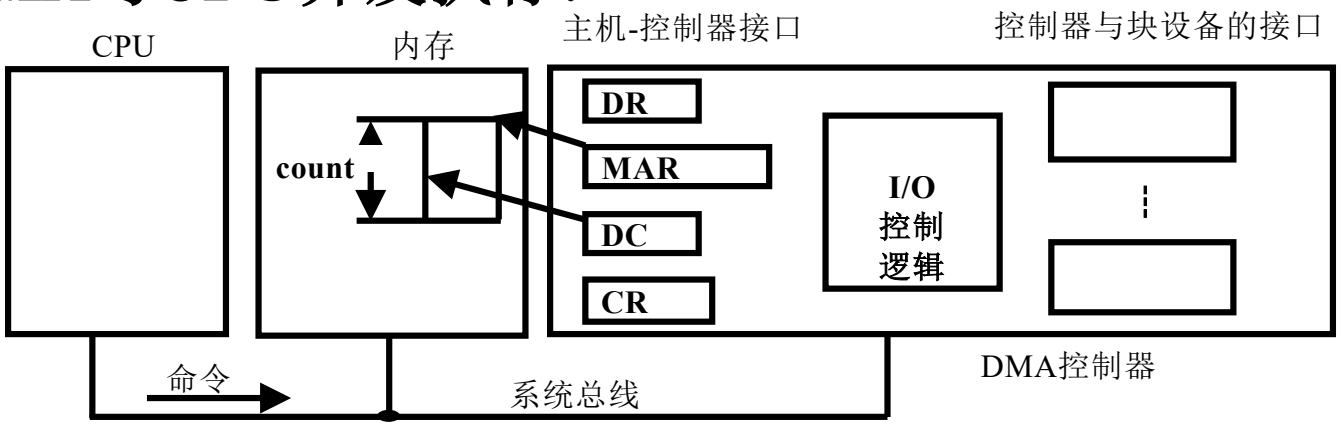


- **关键改进：CPU不再忙等**
- **缺点：每次中断都需要CPU介入**



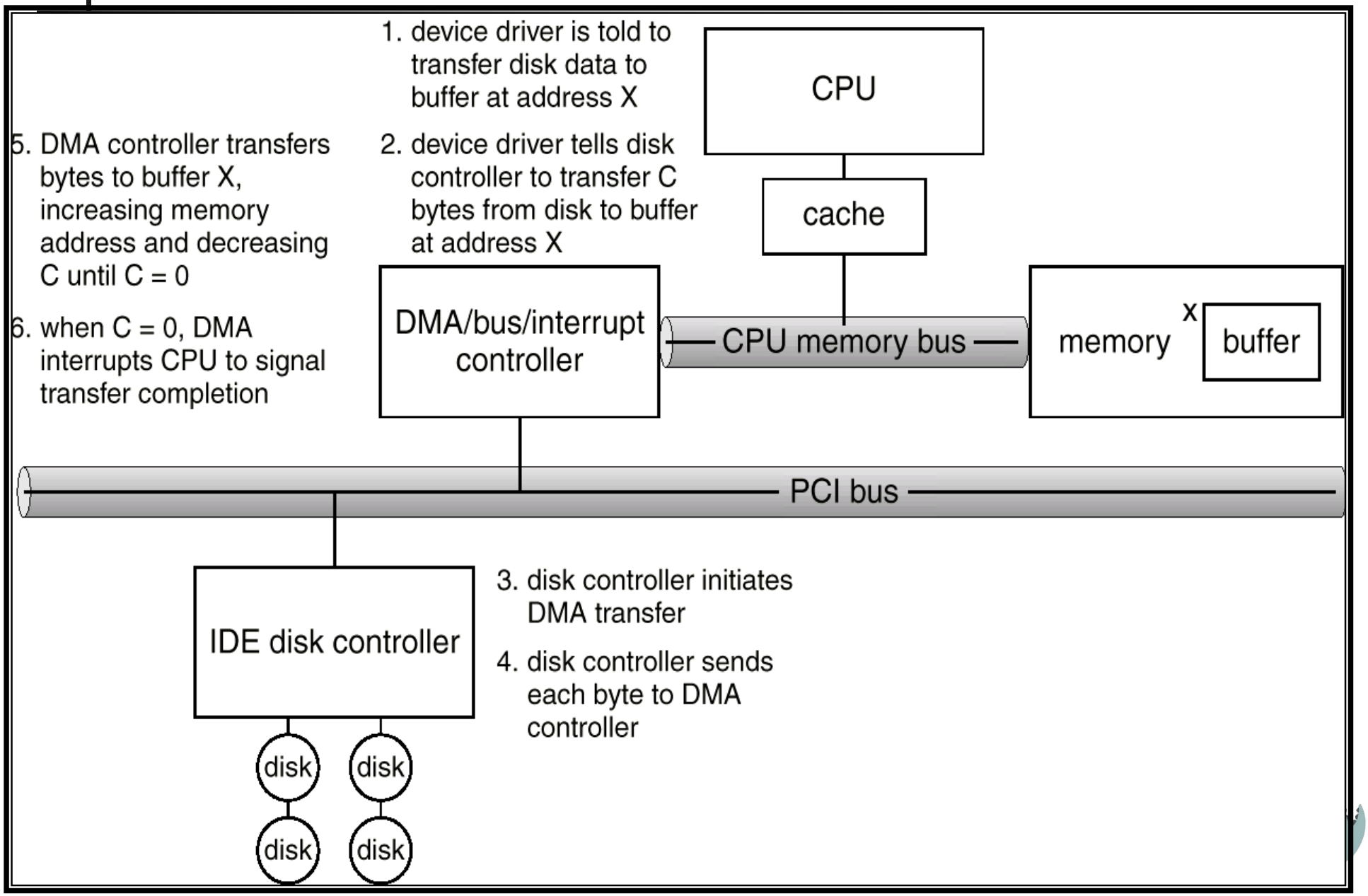
DMA（直接内存访问）

- **核心思想：**让DMA控制器代替CPU传输数据
- **工作流程：**
  - 1. CPU配置DMA控制器（源地址、目标地址、传输长度）
  - 2. DMA控制器直接在内存和设备间传输数据
  - 3. 传输完成后DMA发送中断通知CPU
- **DMA与CPU并发执行：**



- **优点：**大幅减少CPU介入，适合大批量数据传输

DMA传输的六个步骤

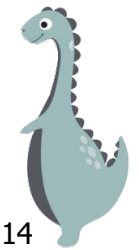

1. device driver is told to transfer disk data to buffer at address X
2. device driver tells disk controller to transfer C bytes from disk to buffer at address X
3. disk controller initiates DMA transfer
4. disk controller sends each byte to DMA controller
5. DMA controller transfers bytes to buffer X, increasing memory address and decreasing C until C = 0
6. when C = 0, DMA interrupts CPU to signal transfer completion
CPU
cache
DMA/bus/interrupt controller
CPU memory bus
memory
x
buffer
PCI bus
IDE disk controller
disk
disk
disk
disk

通道控制方式

- **通道的本质**：独立的I/O处理器
- **工作方式**：
  - CPU发送通道程序（Channel Program）
  - 通道自主执行多个I/O操作
  - 完成后通知CPU
- **应用场景**：大型主机、高端服务器

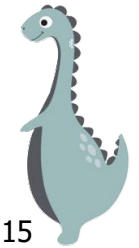
四种控制方式对比

方式	CPU介入	硬件成本	数据传输量	适用场景
程序控制	极高	低	小	简单字符设备
中断驱动	高	中	中	键盘、鼠标
DMA	低	中	大	磁盘、网卡
通道	极低	高	极大	大型机



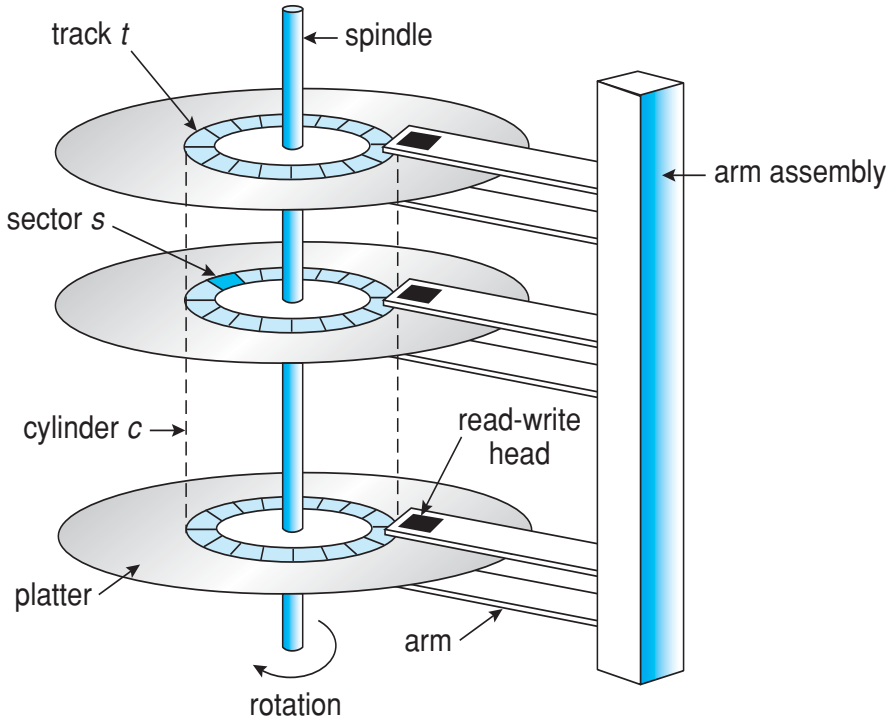
硬盘

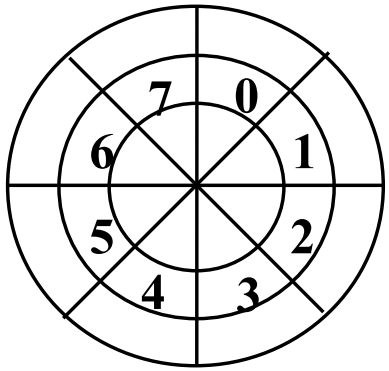
- 核心问题
  - 机械硬盘和固态硬盘有什么本质区别？
  - 为什么需要磁盘调度？
  - 分区和格式化到底做了什么？



机械硬盘物理结构

- 核心部件
  - 盘片 (Platter)
    - 磁道 (Track)
    - 扇区 (Sector)
    - 柱面 (Cylinder)
  - 磁头 (Head)
  - 磁头臂 (Arm)
  - 主轴 (Spindle)
- 存储原理
  - 磁性记录



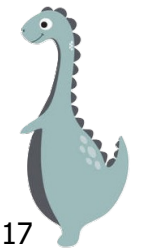


机械硬盘访问时间

- 总访问时间 = 寻道时间 + 旋转延迟 + 传输时间
- 示例计算：

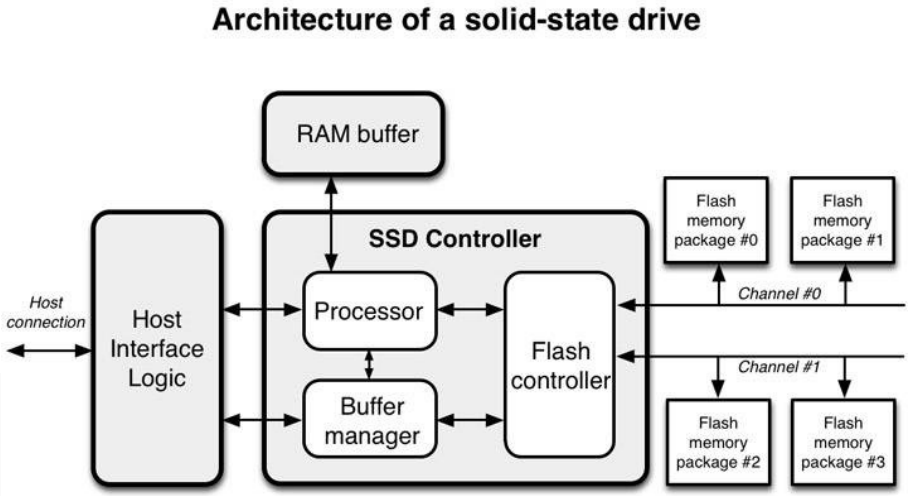
```
假设: 7200RPM硬盘, 平均寻道8ms
- 寻道时间: 8ms
- 旋转延迟: 60s/7200/2 = 4.17ms
- 传输时间: 4KB / 100MB/s ≈ 0.04ms
-----------------------------------
总计: ≈ 12ms
```

- **性能瓶颈**：机械运动（寻道、旋转）



固态硬盘（SSD）结构

- **核心组件：**
  - NAND Flash芯片（存储介质）
  - 控制器（FTL固件）
  - DRAM缓存
- **存储层次：**

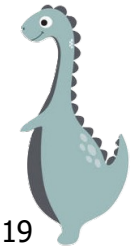



```
芯片 (Chip)
  └ 平面 (Plane)
      └ 块 (Block, 256KB-4MB)
          └ 页 (Page, 4KB-16KB)
```

- **特性：**
  - 读写单位：页（Page）
  - 擦除单位：块（Block）
  - 写前必须擦除

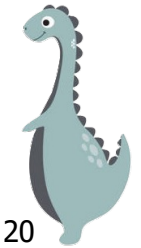
SSD的FTL层

- **FTL（Flash Translation Layer）的三大功能：**
  - **1. 地址映射：**
    - 逻辑块地址（LBA）→ 物理页地址（PPA）
    - 页级映射表
  - **2. 垃圾回收：**
    - 回收无效页，整理碎片
  - **3. 磨损均衡：**
    - 均匀分配写入，延长寿命
- **写放大问题：**
  - 写入4KB → 触发垃圾回收 → 实际写入256KB
  - 写放大因子 = 256/4 = 64



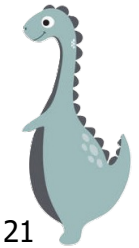
HDD vs SSD对比

特性	机械硬盘（HDD）	固态硬盘（SSD）
存储介质	磁性盘片	NAND Flash
随机访问	慢（10ms）	快（0.1ms）
顺序访问	100-200MB/s	500-3500MB/s
功耗	高（5-10W）	低（2-4W）
抗震性	差	优秀
寿命	长（物理损耗）	有限（P/E次数）
价格	低（$0.02/GB）	高（$0.1/GB）



磁盘调度算法引入

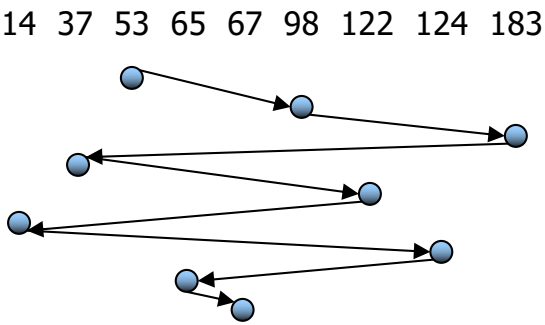
- **问题场景：**
  - 当前磁头位置：53
  - 请求队列：98, 183, 37, 122, 14, 124, 65, 67
- 如何安排访问顺序以减少寻道时间？

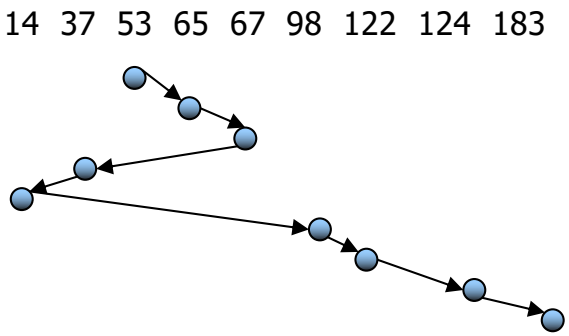


FCFS与SSTF算法

- **FCFS（先来先服务）：**
  - 访问顺序：53 → 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67
  - 总移动距离：= 45+85+146+85+108+110+59+2 = 640
- **SSTF（最短寻道时间优先）：**
  - 访问顺序：53 → 65 → 67 → 37 → 14 → 98 → 122 → 124 → 183
  - 总移动距离：= 12+2+30+23+84+24+2+59 = 236

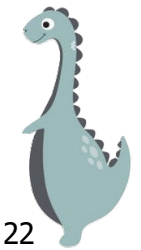
FCFS





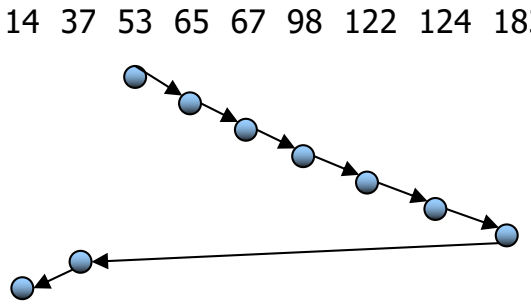
SSTF

- **SSTF问题：**远端请求可能饥饿

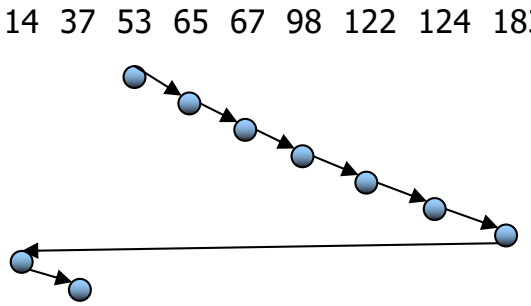


LOOK与C-LOOK算法

- SCAN（电梯算法）vs LOOK:
  - 当前向右移动:
  - 53 → 65 → 67 → 98 → 122 → 124 → 183 → (到边界199)→ 37 → 14
  - 53 → 65 → 67 → 98 → 122 → 124 → 183 → 37 → 14
  - 总移动距离：= (199-53) + (199-14) = 331
  - 总移动距离：= (183-53) + (183-14) = 299
- C-SCAN（循环扫描）vs C-LOOK:
  - 单向扫描:
  - 53 → 65 → 67 → 98 → 122 → 124 → 183 →(到边界199)→(返回0)→ 14 → 37
  - 53 → 65 → 67 → 98 → 122 → 124 → 183 → 14 → 37
  - 总移动距离：= (199-53) + 199+37 = 382
  - 总移动距离：= (183-53) + (183-14)+(37-14) = 322
  - 优点：更公平的等待时间



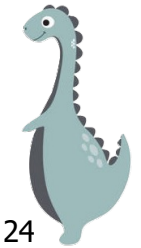
LOOK



C-LOOK

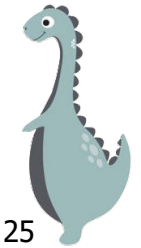
N-Step-SCAN

- 若多个进程反复请求对某一磁道的访问，则磁臂可能停留在某处不动，这一现象称为磁臂粘着。
- N-Step-SCAN算法：将磁盘请求队列分成若干个长度为N的子队列，磁盘调度按FCFS算法依次处理这些子队列，而处理每个队列时按SCAN算法进行，一个队列处理完后，再处理其他队列。



FSCAN算法

- FSCAN算法是N-Step-SCAN算法的简化，它只将磁盘请求队列分成两个子队列。一个是当前所有请求磁盘I/O的进程形成的队列，由磁盘调度按SCAN算法进行处理，另一个队列则是在扫描期间新出现的磁盘请求。



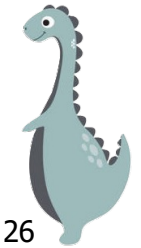
分区与格式化

- 分区：
  - **MBR分区表**：最大支持2TB，4个主分区
  - **GPT分区表**：支持>2TB，128个分区
  - 分区作用：逻辑隔离、多系统、性能优化
- 格式化：
  - **低级格式化**：划分扇区和磁道（出厂完成）
  - **高级格式化**：建立文件系统

磁盘布局

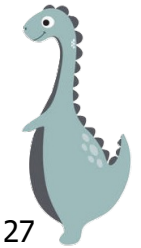
Boot	FAT	Directory	Data

- **Linux示例**：
  - fdisk /dev/sda　　# 分区
  - mkfs.ext4 /dev/sda1　　# 格式化为ext4文件系统



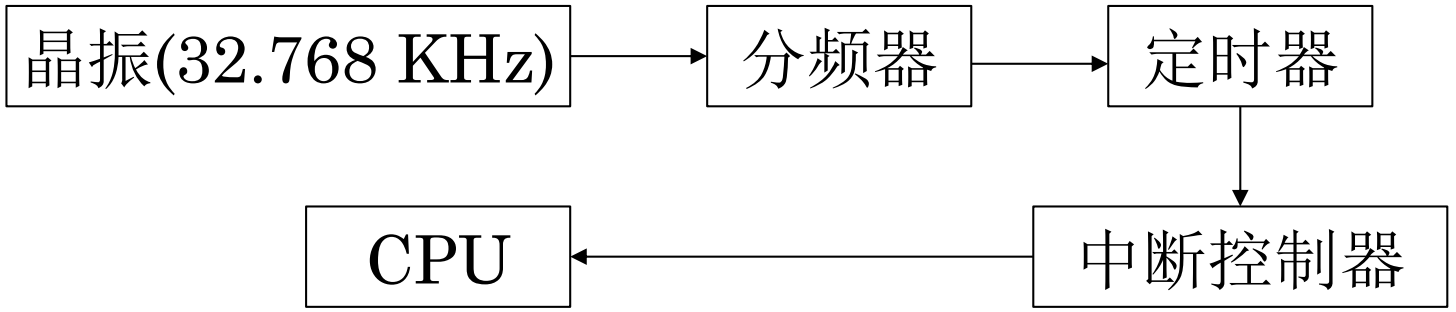
时钟的重要性

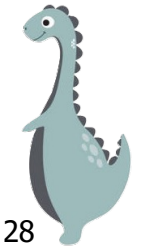
- **核心问题：**操作系统如何感知时间？
- **时钟的四大作用：**
  - 1. 维护系统时间（当前时间）
  - 2. 驱动进程调度（时间片）
  - 3. 实现定时功能（sleep、timeout）
  - 4. 统计CPU使用时间



时钟硬件

- 硬件组成：
  - 晶振（Crystal Oscillator）：提供稳定的时钟信号
  - 可编程定时器：如Intel 8254
  - RTC（实时时钟）：掉电后仍保持时间
- 工作原理：

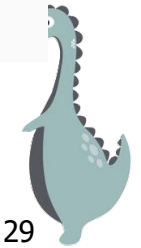




时钟中断

- **时钟中断产生：**
  - 硬件定时器每隔固定时间产生中断
  - Linux中HZ参数：100、250、1000（每秒中断次数）
- **时钟中断处理：**

```
void timer_interrupt_handler() {
    jiffies++;                  // 更新系统滴答数
    update_process_time();      // 更新进程时间
    check_timers();             // 检查定时器队列
    scheduler_tick();           // 调度器时钟滴答
}
```

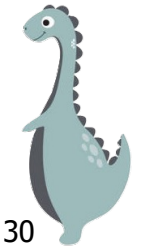


软定时器

- **实现原理：**
  - 定时器队列，按到期时间排序
  - 每次时钟中断检查队列头部
- **Linux定时器API：**

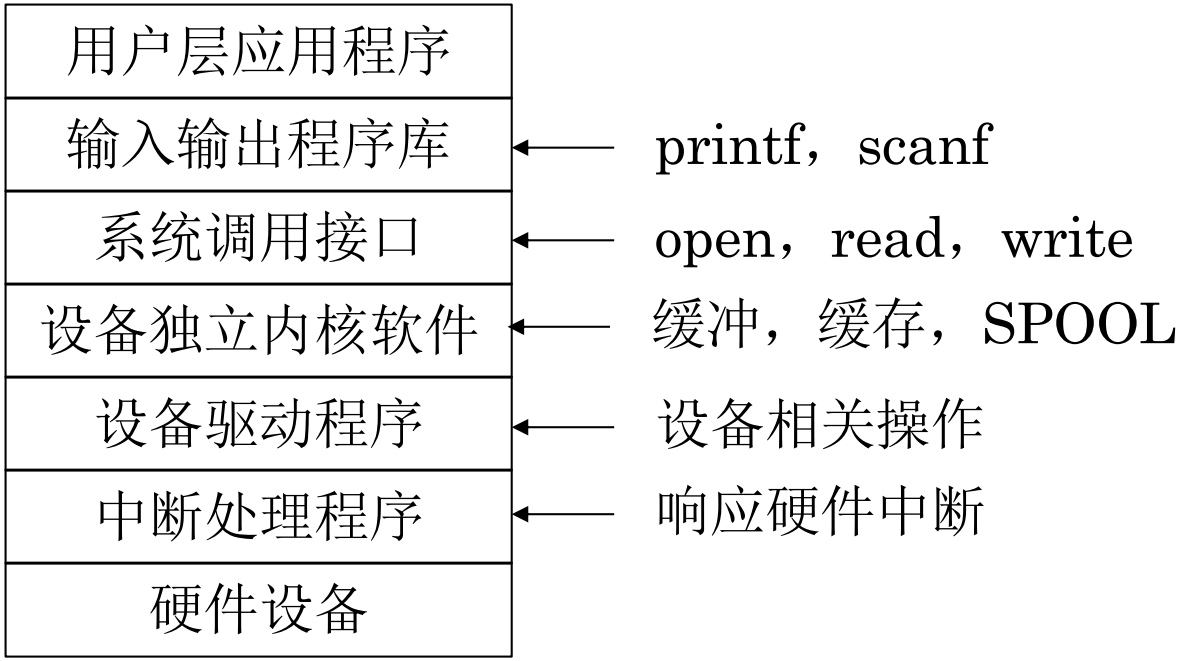
```
struct timer_list timer;
timer_setup(&timer, callback_func, 0);
mod_timer(&timer, jiffies + HZ); // 1秒后触发
```

- **高精度定时器（hrtimer）：**
  - 纳秒级精度
  - 用于实时系统

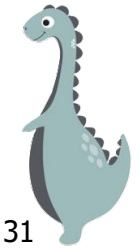


I/O系统层次架构

- 核心问题
  - 应用程序printf("Hello")如何最终输出到屏幕？
  - 中间经历了哪些层次？
  - 为什么需要这么多层？

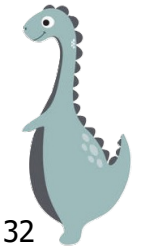


I/O软件层次结构



各层次功能详解（1）

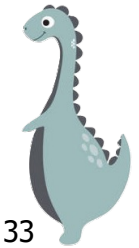
- **硬件设备层：**
  - 实际的物理设备
  - 包括控制器、缓冲区、寄存器
- **中断处理程序层：**
  - 响应设备中断
  - 保存/恢复上下文
  - 唤醒被阻塞的进程
  - 向驱动程序报告I/O完成
  - **关键特点**：时间敏感，执行快速



各层次功能详解（2）

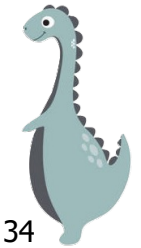
- **设备驱动程序层：**
  - 设备相关的具体操作
  - 将高层请求转换为设备命令
  - 检查设备状态、处理错误
  - 对上层提供统一接口
- **示例功能：**

```
disk_driver.read(block_num) {
    // 1. 将块号转换为柱面/磁头/扇区
    // 2. 发送命令到磁盘控制器
    // 3. 等待中断或轮询状态
    // 4. 返回数据
}
```



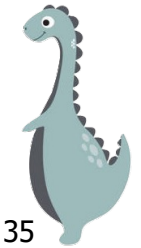
各层次功能详解（3）

- **设备独立内核软件层：**
  - 提供设备无关的抽象接口
  - 统一设备命名
  - 缓冲管理
  - 错误报告
  - 设备分配与释放
- **关键作用：**屏蔽设备差异



各层次功能详解（4）

- 系统调用接口层：
  - POSIX标准接口：
    - open()打开设备/文件
    - close()关闭
    - read()读取
    - write()写入
    - ioctl()设备控制
- 输入输出程序库层：
  - C标准库：`printf`, `scanf`, `fopen`, `fread`
  - 在用户态运行
  - 提供缓冲、格式化等便利功能



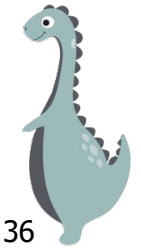
“一切皆文件”设计思想

- **Unix/Linux哲学：**
  - 将设备抽象为文件
  - 统一操作接口
- 设备文件

```
/dev/sda     # 块设备（硬盘）
/dev/tty     # 字符设备（终端）
/dev/null    # 特殊设备（空设备）
```

  - **优势：**
    - 简化编程接口
    - 统一的权限管理
    - 支持管道、重定向
  - **示例：**

```
cat file.txt > /dev/tty   # 输出到终端
dd if=/dev/sda of=disk.img   # 拷贝磁盘
```



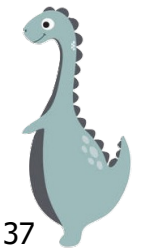
设备号机制

- **设备号组成：**
  - 主设备号 (Major Number)：标识驱动程序
  - 次设备号 (Minor Number)：标识设备实例

```
$ ls -l /dev/sda*
brw-rw---- 1 root disk 8, 0  /dev/sda
brw-rw---- 1 root disk 8, 1  /dev/sda1
brw-rw---- 1 root disk 8, 2  /dev/sda2
                       ↑  ↑
                      主  次
```

  - 内核使用

```
MAJOR(dev)  // 获取主设备号
MINOR(dev)  // 获取次设备号
MKDEV(major, minor)  // 构造设备号
```



驱动程序注册接口

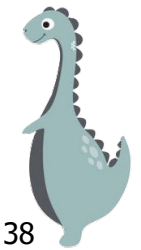
- 字符设备注册：

```
int register_chrdev(unsigned int major,
                    const char *name,
                    struct file_operations *fops);
```

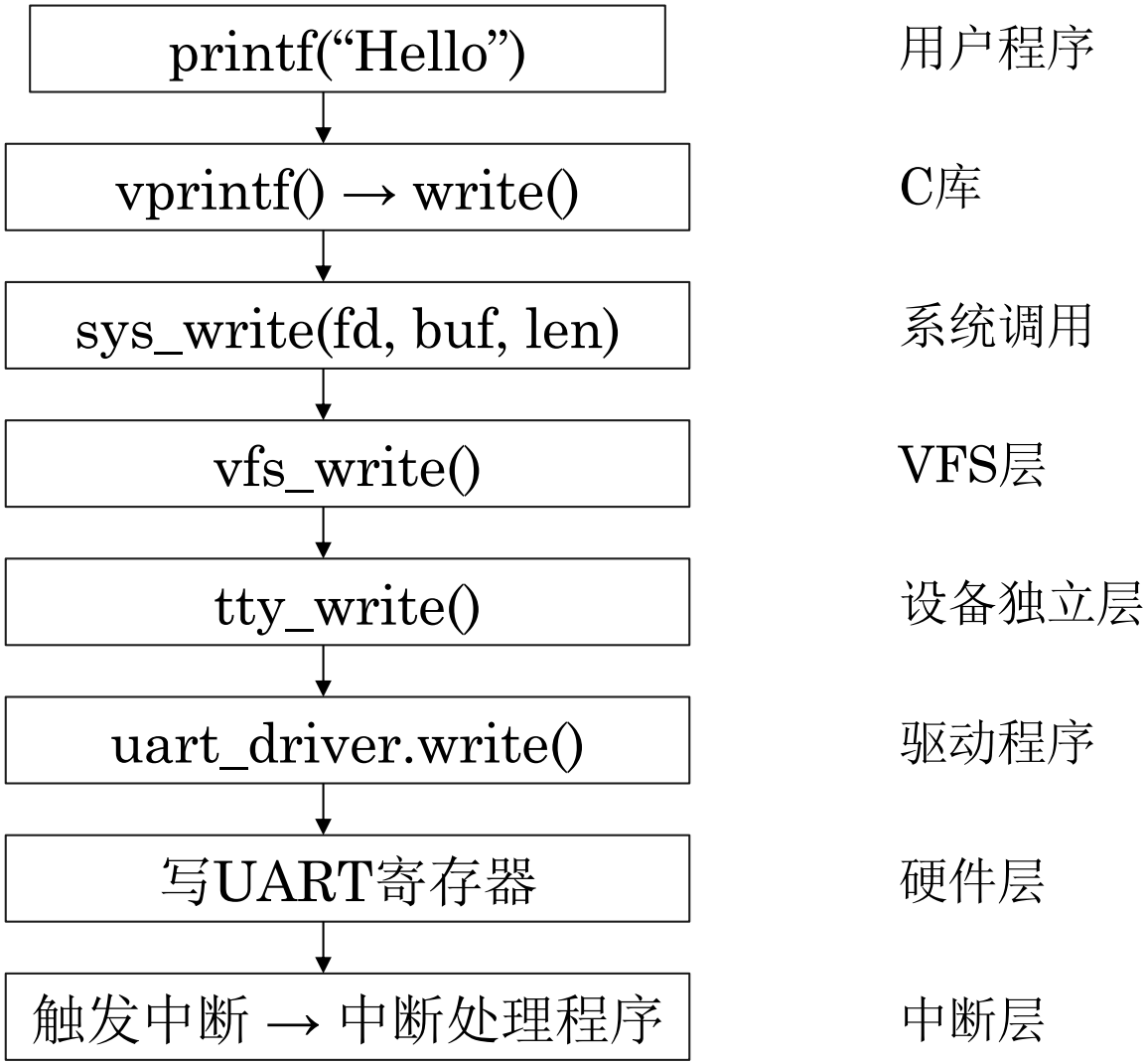
- 块设备注册：

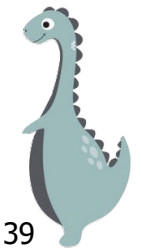
```
int register_blkdev(unsigned int major,
                    const char *name);
```

- 注册过程：
  - 分配或使用指定的主设备号
  - 将驱动名称和操作函数表关联
  - 在内核字符设备表中注册



printf到硬件的完整流程



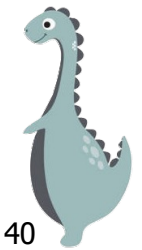


网络接口的特殊性

- 为什么网络设备不用"文件"模型？
  - 网络通信是双向的、异步的
  - 需要处理协议栈
  - 涉及多路复用
- Socket接口：

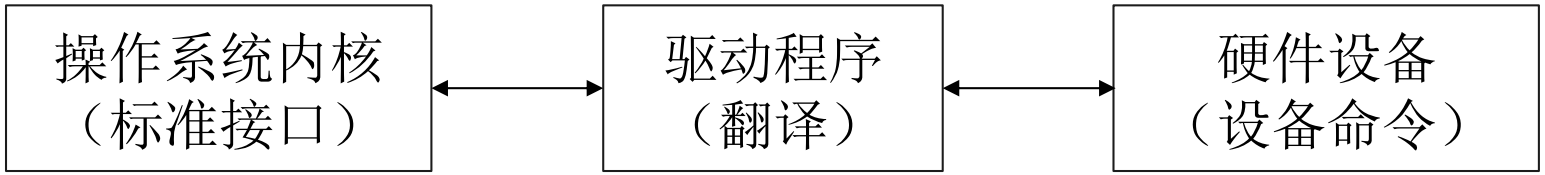
```
int sock = socket(AF_INET, SOCK_STREAM, 0);
connect(sock, ...);
send(sock, data, len, 0);
recv(sock, buf, len, 0);
close(sock);
```

  - 仍保持"一切皆文件描述符"：
    - Socket也是fd
    - 可以用select/poll/epoll统一管理

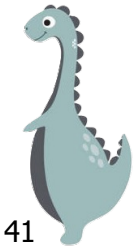


设备驱动程序

- 核心问题
  - 驱动程序的本质是什么？
  - 为什么插入U盘需要"安装驱动"？
- 驱动程序是中间人



- 关键任务
  - 封装硬件细节
  - 提供标准接口
  - 管理设备状态
  - 处理并发访问



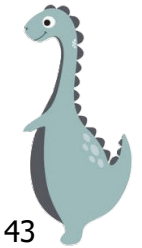
驱动程序安装与卸载

- 静态编译方式
  - 编译进内核
  - 优点：加载快、稳定
  - 缺点：不灵活，需重新编译内核
- 动态加载方式（内核模块）
  - 优点：灵活、无需重

```
# 加载模块
insmod mydriver.ko

# 查看已加载模块
lsmod | grep mydriver

# 卸载模块
rmmod mydriver

# 智能加载（处理依赖）
modprobe mydriver
```

中断处理程序

- 核心问题：
  - 中断处理程序和驱动程序是什么关系？
  - 为什么要分上半部和下半部？
  - 中断嵌套会有什么问题？



中断处理全流程

硬件产生中断信号

↓

CPU保存当前上下文（PC、寄存器、PSW）

↓

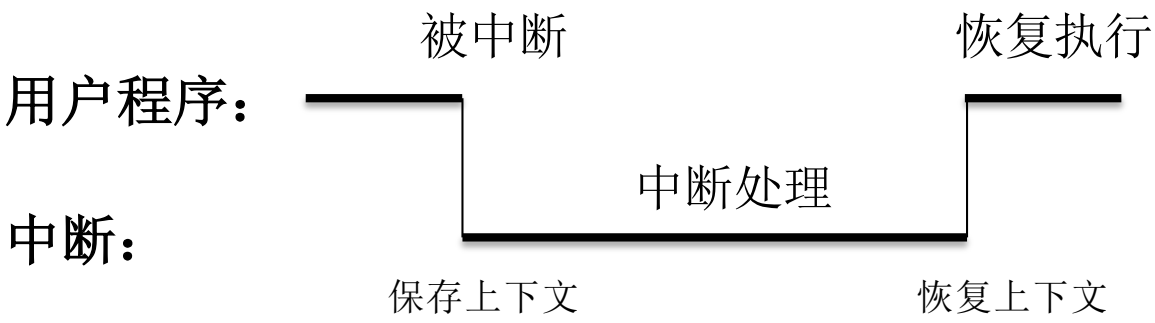
查中断向量表，跳转到中断处理程序

↓

执行中断处理程序

↓

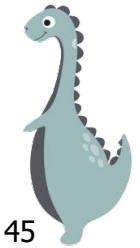
恢复上下文，返回被中断程序



上下文保护与恢复

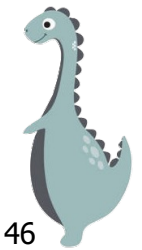
- **需要保存的内容：**
  - 程序计数器（PC/IP）
  - 通用寄存器（EAX, EBX, ECX...）
  - 程序状态字（PSW/EFLAGS）
  - 栈指针（SP）
- **保存位置：**内核栈
- **谁来保存：**
  - **硬件自动保存：**PC、PSW（部分）
  - **软件保存：**其他寄存器
- **恢复时机：**中断返回指令（IRET）
- **示例代码：**

```
; 中断入口
push eax              ; 保存寄存器
push ebx
; ... 中断处理 ...
pop ebx               ; 恢复寄存器
pop eax
iret                  ; 中断返回
```



中断嵌套

- **什么是中断嵌套？**
  - 处理中断A时，又发生中断B
  - 如果B优先级高，可以打断A
- **中断优先级：**
  - 高优先级：硬件故障、电源故障
  - 中优先级：时钟、键盘
  - 低优先级：软件中断
- **嵌套的问题：**
  - **栈溢出风险：**每次嵌套都要保存上下文
  - **实时性影响：**低优先级长时间得不到服务
  - **复杂性增加：**临界区保护更复杂
- **限制嵌套：**
  - 关中断（cli/sti）
  - 中断屏蔽

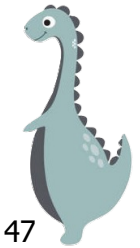


上半部与下半部机制

- **问题：**为什么要分离？
  - 中断处理应尽可能快
  - 某些工作可以延迟
  - 避免在中断上下文做复杂操作

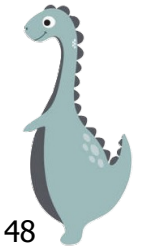
- 上半部（Top Half）
  - 运行在中断上下文
  - 不可阻塞、不可睡眠
  - 快速响应、时间敏感
  - 任务：
    - 确认中断
    - 读取硬件状态
    - 清除中断标志
    - 调度下半部

- 下半部（Bottom Half）
  - 运行在进程上下文或软中断上下文
  - 可以睡眠（工作队列）
  - 处理耗时的I/O操作
  - 任务：
    - 数据处理
    - 协议栈处理
    - 唤醒等待进程



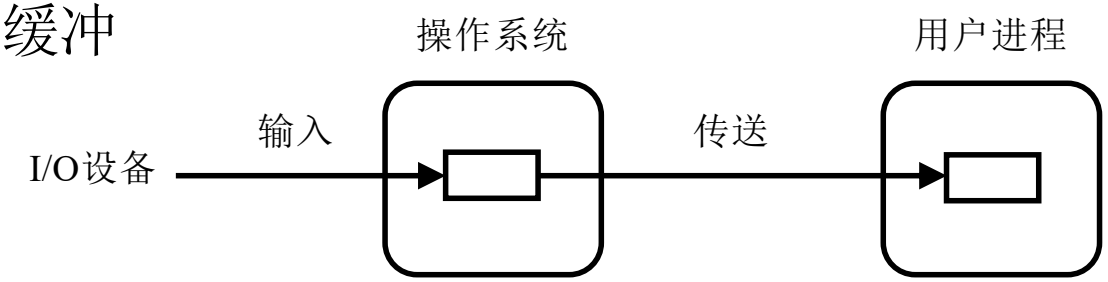
设备独立内核软件

- 核心问题
  - 为什么需要设备独立层？
  - 缓冲和缓存有什么区别？
  - SPOOLing如何让打印机"变快"？
- 缓冲技术的必要性
  - **速度不匹配**：CPU快，设备慢
  - **减少中断**：批量处理
  - **粒度不匹配**：应用1字节，设备1块
  - **提高并发**：CPU和设备同时工作



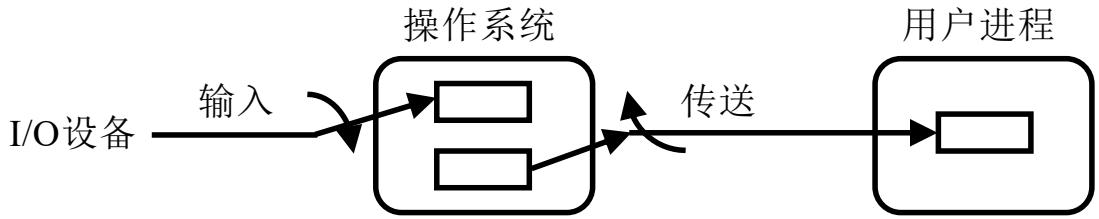
单缓冲与双缓冲

- 单缓冲



设备和CPU不能同时工作

- 双缓冲（乒乓缓冲）

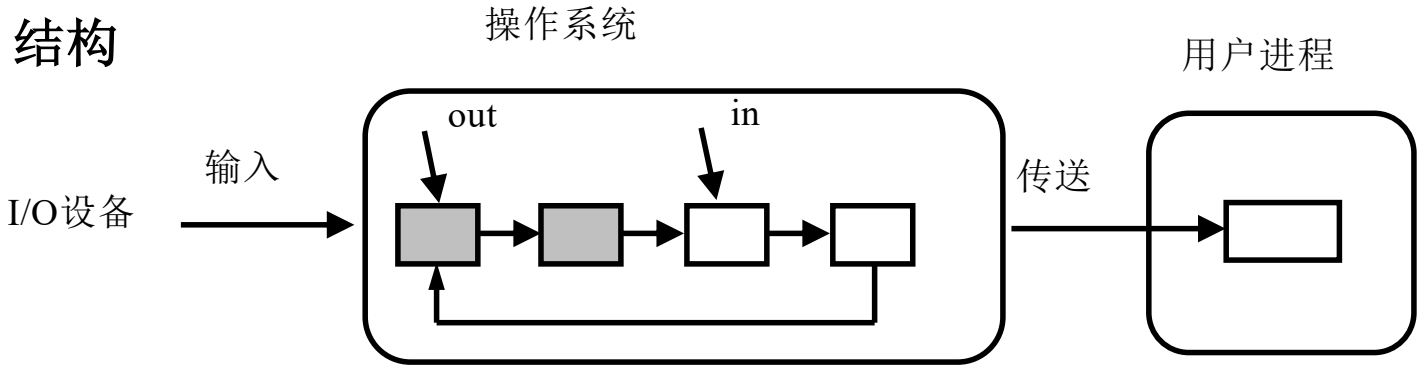


设备和CPU可以并行工作

- 性能对比：
  - 假设：M=设备传输时间，C=CPU处理时间
  - 单缓冲：时间 = M + C
  - 双缓冲：时间 = max(M, C)
- 应用：视频播放、网络传输

环形缓冲

- **结构**



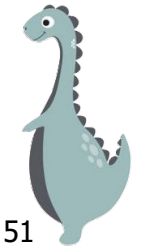
- **关键指针：**
  - **写指针（生产者）in：**设备写入位置
  - **读指针（消费者）out：**CPU读取位置
- **满与空判断：**
  - 空：out == in
  - 满：(in + 1) % size == out
- **优点：**
  - 充分利用缓冲区
  - 适合流式数据（音频、串口）

缓冲池

- **概念：**
  - 系统维护一个缓冲区池
  - 动态分配给需要的进程/设备
- **三个队列：**

空缓冲队列	← 可用缓冲区
装满数据队列	← 待处理数据
正在使用缓冲队列	← 设备正在使用

- **工作流程：**
  - 设备从空队列获取缓冲区
  - 填充数据后放入装满队列
  - CPU从装满队列取出处理
  - 处理完放回空队列
- **优点：**灵活、高效

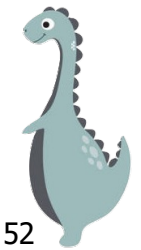


缓存技术

- **缓冲 vs 缓存**

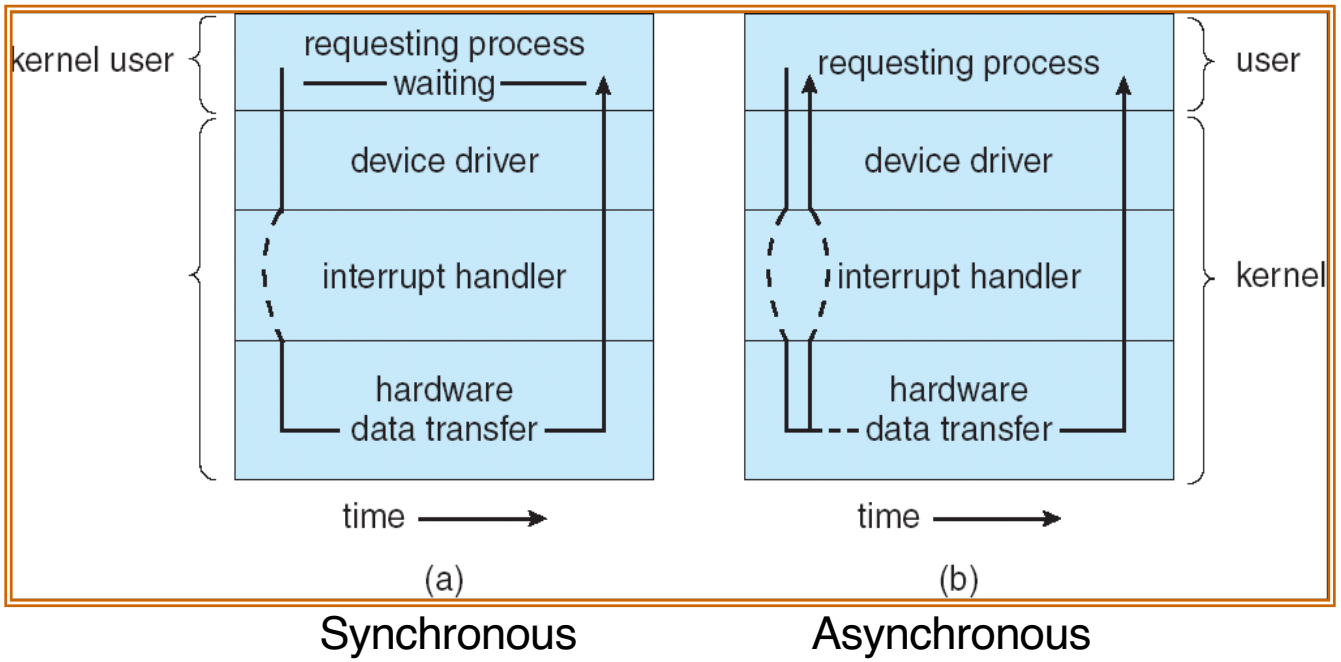
特性	缓冲（Buffer）	缓存（Cache）
目的	解决速度不匹配	提高访问速度
数据使用	通常一次	可能多次
方向	单向（输入或输出）	双向（读写）
示例	键盘缓冲	磁盘缓存

- **磁盘高速缓存：**
  - 缓存最近访问的磁盘块
  - LRU替换算法
  - 命中率通常80-95%
- **Linux页缓存（Page Cache）：**
  - 统一的文件/块缓存
  - 使用空闲内存
  - 自动管理



异步I/O

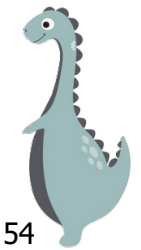
- 同步I/O vs 异步I/O



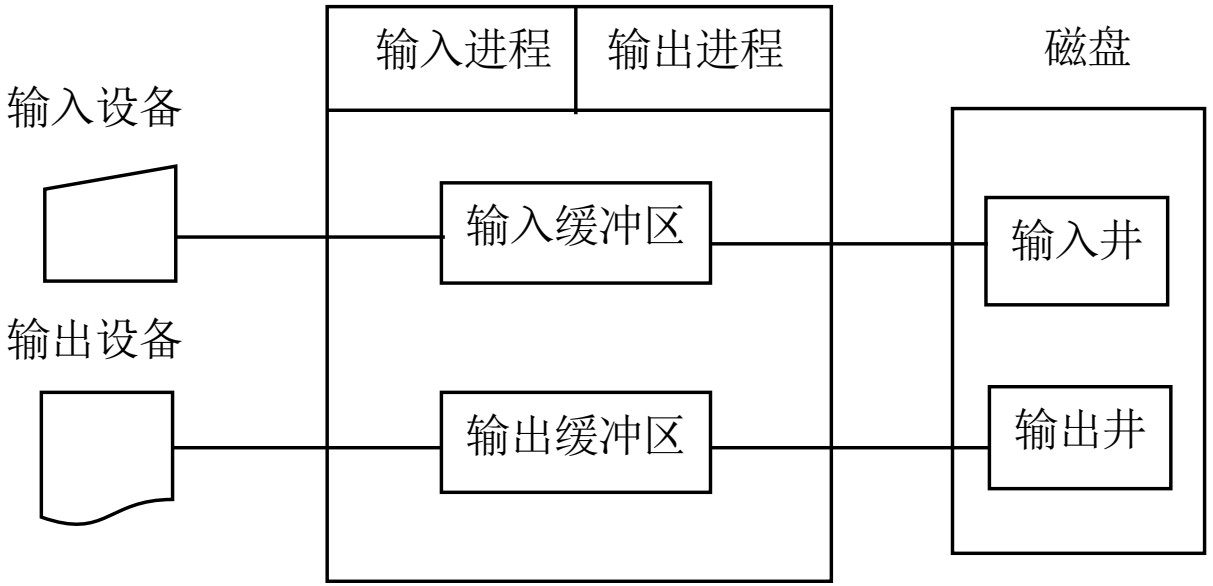
- 异步I/O的优点：
  - 提高CPU利用率
  - 适合I/O密集型应用
- 应用：Web服务器、数据库

SPOOLing技术

- **什么是SPOOLing：**
  - Simultaneous Peripheral Operation On-Line
  - 假脱机技术
  - **核心思想：**用磁盘（快）模拟慢速设备
- **典型应用：打印机共享**
- **问题场景：**
  - 打印机慢（几十页/分钟）
  - 多个进程需要打印
  - 直接独占会导致长时间等待

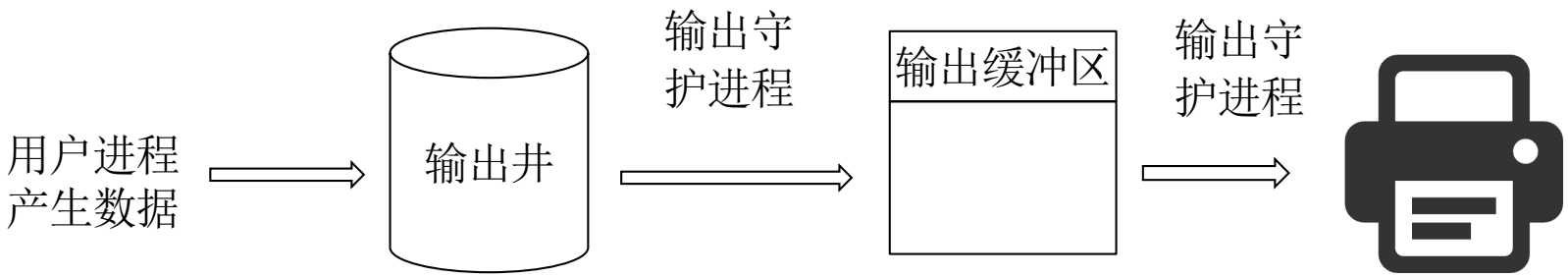


SPOOLing工作原理



- 系统组成
  - 输入井/输出井：磁盘上的文件
  - 请求队列：等待处理的任务
  - 输入进程/输出进程：用于守护数据从慢速设备输入到输入井/从输出井输出到慢速设备上
  - 输入缓冲区/输出缓冲区：内存中用于收容输入/输出数据的区域

共享打印机

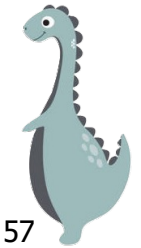


- 工作流程
  - 用户进程：将打印内容写入输入井（磁盘文件），并发送打印请求。打印任务被添加进打印队列。用户进程立即返回（不等待）
  - 输出进程（守护进程）：
    - 从打印队列获取任务，将打印数据从输出井读取数据，放入输出缓冲区
    - 控制打印机打印
    - 完成后删除输出井文件
- 优点：
  - 用户无需等待
  - 提高打印机利用率
  - 实现设备共享
  - 可实现优先级调度

设备分配与回收

- **设备分类：**
  - **独占设备：**一次只能一个进程（打印机）
  - **共享设备：**可同时多个进程（磁盘）
  - **虚拟设备：**通过SPOOLing虚拟化
- **分配策略：**
  - 先请求先服务（FCFS）
  - 优先级调度
  - 考虑死锁避免
- **设备独立性：**
  - 逻辑设备名 → 物理设备名
  - 示例：打印机 → /dev/lp0
- **保护机制：**
  - 权限检查
  - Linux设备文件权限：

```
crw-rw---- 1 root audio 14, 3  /dev/dsp
↑           ↑           ↑
字符设备   所有者       组权限
```



总结

设备管理基本概念 → I/O控制方式进化

↓

典型设备（硬盘、时钟）原理

↓

I/O软件层次架构

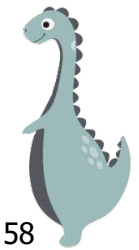
- 中断处理程序（上下半部）
- 设备驱动程序
- 设备独立软件（缓冲、缓存、SPOOL）

核心设计思想：

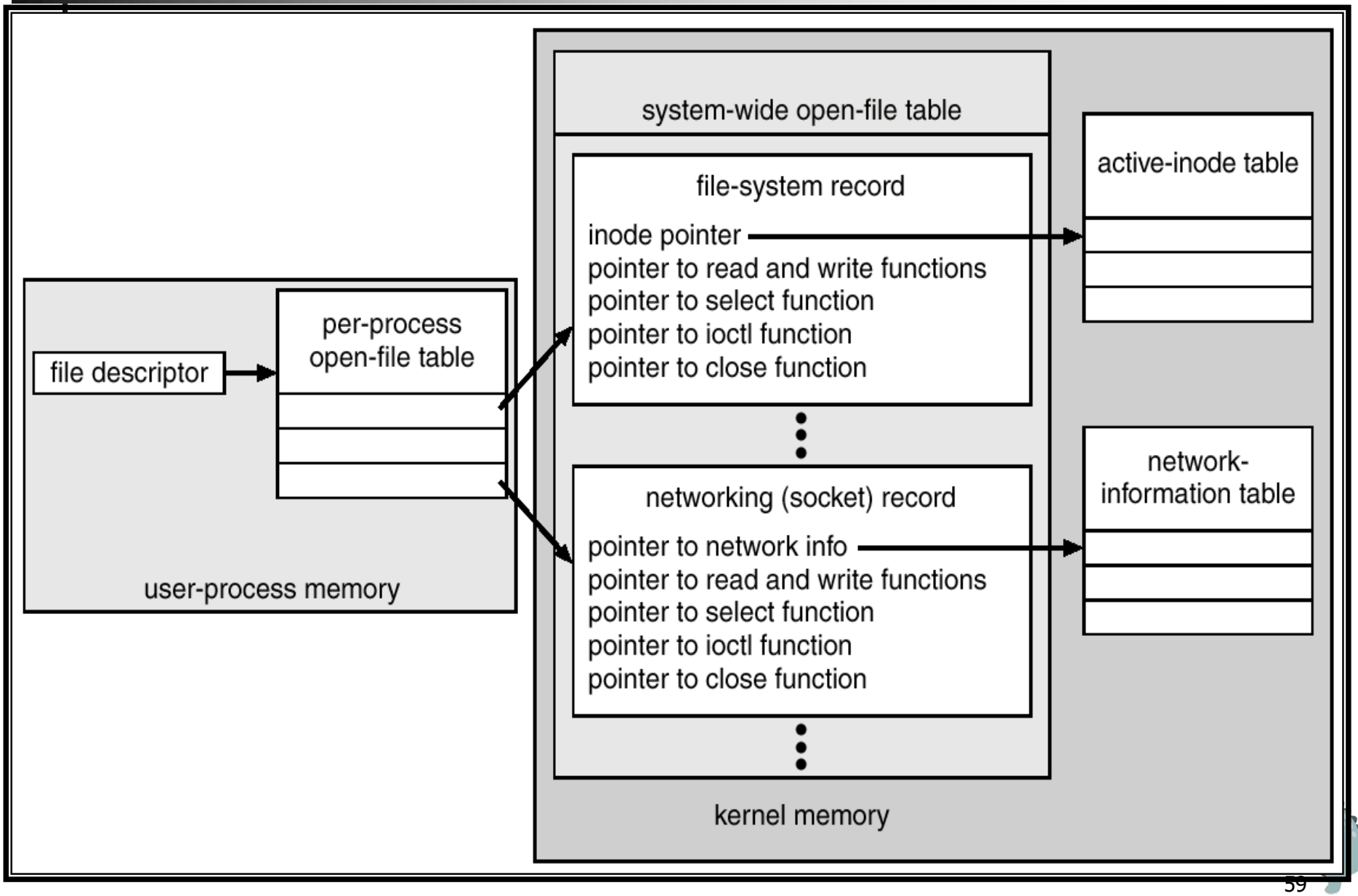
1. **分层抽象**：隐藏硬件细节
2. **一切皆文件**：统一接口
3. **上下半部分离**：提高响应性
4. **缓冲/缓存**：解决速度不匹配
5. **SPOOLing**：虚拟化独占设备

重点掌握：

- 四种I/O控制方式对比
- 磁盘调度算法
- I/O软件层次及各层功能
- 中断上下半部机制
- 缓冲技术和SPOOLing



UNIX I/O Kernel Structure



练习题

1. 在一个现代化的计算机系统中，同时存在多种不同类型的 I/O 设备和数据传输任务。请根据每个任务对**数据传输速率、CPU 占用率、硬件成本**的不同需求，从下列四种 I/O 控制方式中选择**最合适的一种**，并简述理由。

- **备选方式：** A. 程序控制 I/O（轮询） B. 中断驱动 I/O C. DMA（直接内存访问） D. 通道控制

任务场景：

- **场景一：嵌入式温度传感器** 一个简单的嵌入式系统连接了一个温度传感器。该传感器只能存储 1 个字节的数据，且数据变化非常缓慢（每 5 秒才更新一次）。系统没有多余的中断引脚资源，且 CPU 此刻除了监控温度外没有其他紧迫任务。
- **场景二：高性能游戏键盘** 用户正在进行高强度的电竞游戏。每一次按键都需要系统做出毫秒级的极速响应。键盘的输入是随机的、非周期性的，且每次数据量很小（几个字节）。
- **场景三：4K 视频文件拷贝** 用户正在将一个 50GB 的高清电影从外接 NVMe 固态硬盘复制到内存中进行编辑。数据传输量巨大且连续，且此时 CPU 正在运行复杂的视频渲染算法，负载很高，不能被打断。
- **场景四：大型银行的主机系统** 一台负责处理全国交易的大型主机（Mainframe），通过光纤连接了 20 个高速磁盘阵列和 10 个网络终端控制器。系统需要同时从多个磁盘和网络端口并发读写海量数据，CPU 极其昂贵，必须专注于交易逻辑计算，完全不应理会 I/O 细节。

练习题

2. 假设磁盘磁头当前位于 100 号磁道，正在向磁道号增加的方向移动。 磁道请求队列如下（按到达时间顺序）：

55, 58, 39, 18, 90, 160, 150, 38, 184

问题：

（1）若采用LOOK算法，请写出磁头的移动序列和总寻道长度。

（2）假设在磁头向右移动的过程中，源源不断地有访问 100-105 磁道号的新请求到达。LOOK 算法会出现什么现象？这对 18 号磁道的请求意味着什么？

（3）假设系统对 I/O 响应时间有严格要求：任何读请求的等待时间不得超过 500ms。 请你在 LOOK 算法的基础上进行改进，设计一个新的调度策略来满足这一要求。描述你的策略，并说明你的策略是如何解决第（2）问中的问题的？

练习题

3. 操作系统将 I/O 软件划分为四个层次：

- 用户层 I/O 软件
- 设备独立性软件（Device-Independent Software）
- 设备驱动程序（Device Drivers）
- 中断处理程序（Interrupt Handlers）

请分析以下具体操作分别是在哪一层完成的，并说明理由：

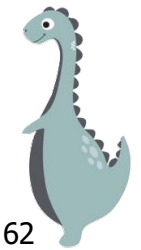
（1）将用户的逻辑块号（Logical Block Number 100）转换为磁盘的物理参数（柱面 C=5, 磁头 H=2, 扇区 S=10）。

（2）检查用户是否有权限访问 /dev/sda1。

（3）将打印机需要的 ASCII 码数据放入设备的硬件寄存器中。

（4）在进程请求读盘时，发现该进程被阻塞，上下文切换到另一个进程。

（5）解析 printf("Hello %d", count) 中的格式字符串。



练习题

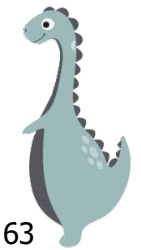
4. 打印机是一个典型的“独占设备”（Exclusive Device）。如果没有 Spooling 系统，当进程 A 正在打印时，进程 B 请求打印必然会被阻塞，甚至可能导致死锁。

问题：

（1）请设计一个基于 **Spooling（假脱机）** 技术的打印系统架构图，说明进程、守护进程（Daemon）、磁盘缓冲区、打印机四者的数据流向。

（2）在这个系统中，进程 A 调用的 write() 函数实际上把数据写到了哪里？函数返回“成功”时，数据真的打印出来了吗？

（3）如果进程 A 正在打印一个 100MB 的文档，打印到一半时，进程 B 也提交了一个打印任务。Spooling 系统如何保证打印出来的纸张不会出现 A 和 B 的内容混杂在一起？



选择题1

- 缓冲技术中的缓冲池在 _____ 中。

  A. 主存　　B. 外存

  C. ROM　　D. 寄存器

- CPU输出数据的速度远远高于打印机的打印速度，为了解决这一矛盾，可采用_____ 。

  A. 并行技术　　B. 通道技术

  C. 缓冲技术　　D. 虚存技术

选择题2

- 通过硬件和软件的功能扩充，把原来独占的设备改造成能为若干用户共享的设备，这种设备称为 _____ 。

  A. 存储设备　　B. 系统设备

  C. 用户设备　　D. 虚拟设备

- 为了使多个进程能有效地同时处理输入/输出，最好使用 _____ 结构的缓冲技术。

  A. 循环缓冲　　B. 缓冲池

  C. 单缓冲　　D. 双缓冲

选择题3

- 如果I/O设备与存储设备进行数据交换不经过CPU来完成，这种数据交换方式是_____。

  A. 程序查询　　B. 中断方式

  C. DMA方式　　D. 无条件存取方式

- 在采用Spooling 技术的系统中．用户的打印结果首先被送到_____。

  A. 磁盘固定区域　　B. 内存固定区域

  C. 终端　　D. 打印机

选择题4

- 按 _____ 分类可将设备分为块设备和字符设备。

  A. 从属关系　　B. 操作特性

  C. 共享属性　　D. 信息交换单位

- _____ 算法是设备分配常用的一种算法。

  A. 短作业优先　　B. 最佳适应

  C. 先来先服务　　D. 首次适应

选择题5

- 在下面关于设备属性的论述中，正确的论述是_____。

A．字符设备的一个基本特征是可寻址的。

B．共享设备必须是可寻址的和可随机访问的设备。

C．共享设备是指在同一时刻，允许多个进程同时访问的设备。

D．在分配共享设备和独占设备时，都可能引起进程死锁。

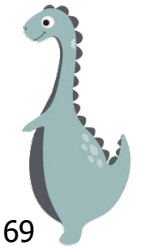
- 通道是一种特殊的___，具有执行I/O指令集的能力。

A．I/O设备　　B．设备控制器

C．处理机　　D．I/O控制器

选择题6

- 共享设备磁盘的物理地址为（柱面号，磁头号，扇区号），磁头从当前位置移动到需访问柱面所用的时间称为 ① ，磁头从访问的柱面移动到指定扇区所用时间称为 ② 。
- A. 寻道时间　　B. 传输时间
- C. 旋转等待时间　　D. 周转时间
- 若进程P1访问199号柱面，磁头是从0号柱面移到199柱面的，且在访问期间依次出现了P2申请读299号柱面，P3申请写209号柱面，P4申请读199号柱面，访问完199号柱面以后，如果采用：先来先服务算法，将依次访问 ① ；最短寻道时间优先算法，将依次访问 ② ；扫描算法，将依次访问 ③ 。
- A. 299，199，209　　B. 299，209，199
- C. 199，209，299　　D. 209，199，299



考研题1

- 程序员利用系统调用打开I/O设备时，通常使用的设备标识是（）。09

  A、逻辑设备名　　B、物理设备名

  C、主设备号　　D、从设备号

- 下列选项中，能引起外部中断的事件是（）。

  A、键盘输入　　B、除数为0

  C、浮点运算下溢　　D、访存缺页

考研题2

- 本地用户通过键盘登陆系统时，首先获得键盘输入信息的程序是____。10

A.命令解释程序　　B.中断处理程序

C.系统调用程序　　D.用户登陆程序

考研题3

- 用户程序发出磁盘I/O请求后，系统的正确处理流程是_____。

A、用户程序→系统调用处理程序→中断处理程序→设备驱动程序

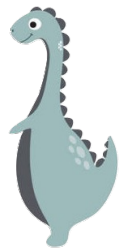
B、用户程序→系统调用处理程序→设备驱动程序→中断处理程序

C、用户程序→设备驱动程序→系统调用处理程序→中断处理程序

D、用户程序→设备驱动程序→中断处理程序→系统调用处理程序

考研题4

- 某文件占10个磁盘块，现要把该文件磁盘块逐个读入主存缓冲区，并送用户区进行分析。假设一个缓冲区与一个磁盘块大小相同，把一个磁盘块读入缓存的时间为100μs，将缓冲区的数据传送到用户区的时间是50μs，CPU对一块数据进行分析的时间是50μs。在单缓冲区及双缓冲区结构下，读入并分析完该文件的时间分别是____。11
- A、1500μs，1000μs　B、1550μs，1100μs
- C、1550μs，1550μs　D、2000μs，2000μs



考研题5

- 中断处理和子程序调用都需要压栈保护现场，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（）。12

A．程序计数器　　B.程序状态寄存器

C．通用数据寄存器　　D.通用地址寄存器

- 操作系统的I/O子系统通常由四个层次组成，每一层明确定义了与邻近层次的接口。其合理的层次组织排列顺序是()。12

A、用户级I/O软件、设备无关软件、设备驱动程序、中断处理程序

B、用户级I/O软件、设备无关软件、中断处理程序、设备驱动程序

C、用户级I/O软件、设备驱动程序、设备无关软件、中断处理程序

D、用户级I/O软件、中断处理程序、设备无关软件、设备驱动程序

考研题6

- 假设磁头当前位于第105道，正在向磁道序号增加的方向移动。现有一个磁道访问序列请求为35、45、12、68、110、180、170、195，采用SCAN算法得到的磁道访问序列为（）。09

  A、110、170、180、195、68、45、35、12

  B、110、68、45、35、12、170、180、195

  C、110、170、180、195、12、35、45、68

  D、12、35、45、68、110、170、180、195

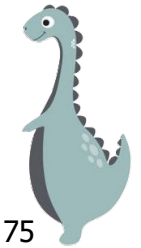
- 下列选项中，不能改善磁盘I/O性能的是（）12

  A. 重排I/O请求次序

  B. 在一个磁盘上设置多个分区

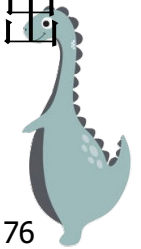
  C. 预读和滞后写

  D. 优化文件物理块的分布



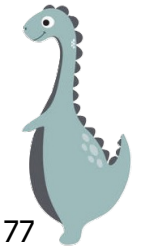
考研题7

- 假设计算机系统采用CSCAN（循环扫描）磁盘调度策略，使用2KB的内存空间记录16384个磁盘块的空闲状态。
- （1）请说明在上述条件下如何进行磁盘块空闲状态管理。
- （2）设某单面磁盘旋转速度为每分钟6000转，每个磁道有100个扇区，相邻磁道间的平均移动时间为1ms。若在某时刻，磁头位于100号磁道处，并沿着磁道号增大的方向移动（如图所示），磁道号请求队列为50，90，30，120，对请求队列中的每个磁道需读取1个随机分布的扇区，则读完这个扇区点共需要多少时间？要求给出计算过程。



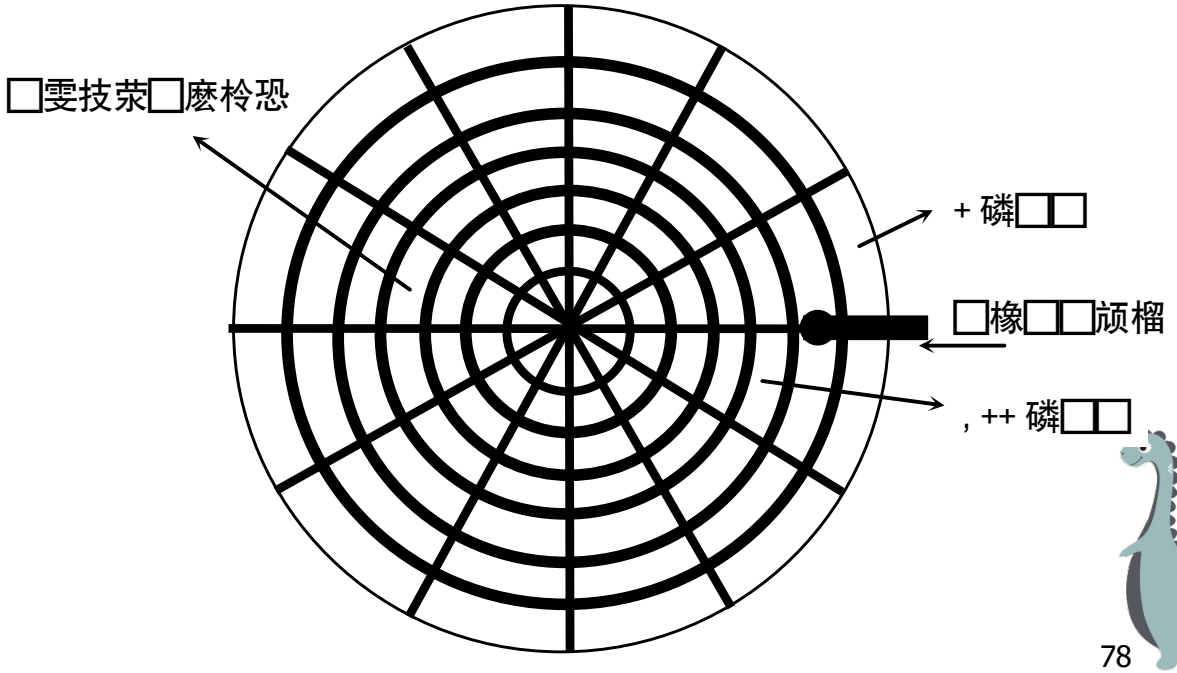
考研题7-2

- （3）如果将磁盘替换为随机访问的Flash半导体存储器（如U盘、SSD等），是否有比CSCAN更高效的磁盘调度策略？若有，给出磁盘调度策略的名称并说明理由；若无，说明理由。



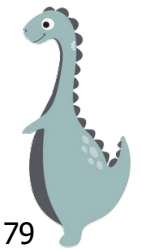
考研题7-3

- （1）使用位示图法表示磁盘的空闲状态（1分），每一位表示一个磁道块是否为空闲，共需要16384/32=512个字=512×4个字节=2KB，正好可放在系统提供的内存中（1分）。



考研题7-4

- （2）采用CSCAN调度算法，访问磁道的顺序为120、30、50、90，则移动磁道长度为20+90+20+40=170，总的移动磁道时间为170×1ms=170ms（1分）。
- 每分钟6000转，则平均旋转廷迟为60/(6000×2)=5ms，总的旋转廷迟时间=5ms×4=20ms（1分）。
- 每分钟6000转，则读取一个磁道上一个扇区的平均读取时间为10ms/100=0.1ms,总的读取扇区的时间=0.1ms×4=0.4ms。
- 读取上述磁道上所有扇区所花的总时间=170ms+20ms+0.4ms=190.4ms（1分）。



考研题7-5

- （3）采用FCFS（先来先服务）调度策略更高效（1分）。因为Flash半导体存储器的物理结构不需要考虑寻道时间和旋转廷迟，可直接按I/O请求的先后顺序服务（1分）。
- 提示：Flash存储器（闪存）属可改写ROM，是一种长寿命的非易失性（在断电情况下仍能保持所存储的数据信息）的存储器，数据删除不是以单个的字节为单位而是以固定的区块为单位，区块大小一般为256KB到20MB。