



# I/O设备

---

中国科学院大学计算机与控制学院

中国科学院计算技术研究所

2021-11-15





# 输入和输出

- 计算机的工作是处理数据
  - 计算：CPU、高速缓存、内存
  - 将数据传入及传出计算机系统
    - 数据在I/O设备与内存之间来回传输
- I/O设备的挑战
  - 多种类型：输入设备、输出设备、存储设备、网络设备、...
  - 大量的厂商、丰富的产品
  - 需要大量的设备驱动
  - 设备驱动运行于内核态，其bug常常引发宕机
- OS的目标
  - 提供一种通用的、一致的、方便的、可靠的方法来访问各种I/O设备
  - 充分发挥I/O设备的性能



# 内容提要

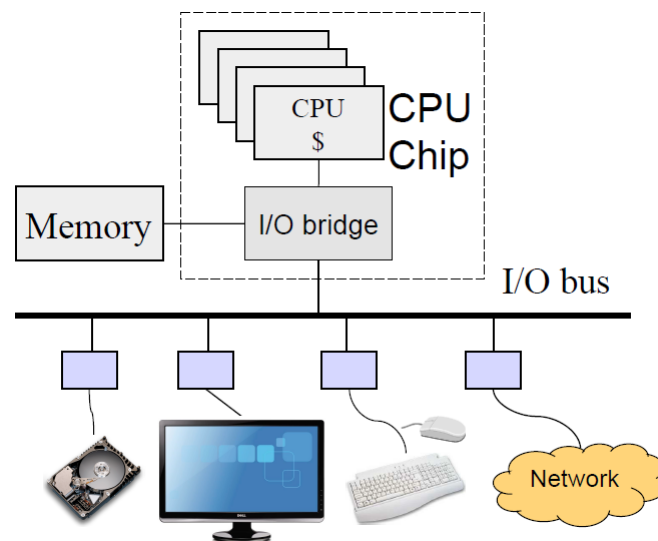
---

- I/O 设备
  - 设备控制器与缓冲区
  - 寻址
- 数据传输
  - PIO, 中断, DMA
- 设备驱动
- 同步I/O与异步I/O



# 硬件回顾

- 计算硬件
  - CPU核和高速缓存
  - 内存控制器和内存
  - I/O总线
  - I/O设备
- I/O硬件
  - I/O总线或互连
  - I/O设备控制器或适配器
  - I/O设备





# 设备控制器 ( Controller/ Adaptor )

- 控制设备的部件
  - 解析主机发来的命令，控制设备进行操作
- 组成
  - 与主机的接口：主机与设备间传递命令、状态、数据
    - 硬件接口：PCIe、SATA、USB
  - 控制寄存器：1个或多个，控制设备操作
    - 写控制寄存器，命令设备执行指定的事情
    - 读控制寄存器，获得设备的状态
  - 数据缓冲区
    - 数据缓冲或缓存
    - 主机端和设备内DRAM



# 寻址：控制寄存器和设备数据缓冲区

- I/O端口：I/O地址空间（Port Mapped IO, PMIO）
  - 端口号：8位或16位的数值
  - I/O专用指令in/out（例如x86）：特权指令
  - 控制线：指示CPU发出的地址是内存空间还是I/O空间
  - 内存地址空间和IO设备地址空间隔离
- 内存映射I/O（Memory Mapped IO, MMIO）
  - 统一地址空间，内存地址空间预留一部分给IO设备内存和寄存器
  - 使用常规的访存指令（例如RISC指令集）
  - 内存地址与I/O地址无重叠
  - CPU发出的地址，所有内存模块和所有设备都要解析



# 寻址：控制寄存器和设备数据缓冲区

- 两者结合
  - 控制寄存器采用I/O端口寻址
  - 数据缓冲区采用内存映射I/O
- 好处
  - 编程方便
  - 保护方便&灵活：利用虚存的保护机制
  - 高效：访问数据缓冲区不用专用指令，减少指令



# 内核缓冲区

---

- 生产者与消费者之间速度不匹配
  - 字符设备和块设备等
  - 适配不同的数据传输大小，例如网络包与字节流
- 缓存
  - 可以服务对同一数据的请求
  - 减少实际访问设备的I/O操作





# 内容提要

---

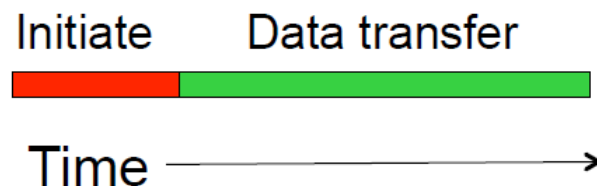
- I/O 设备
  - 设备控制器与缓冲区
  - 寻址
- 数据传输
  - PIO, 中断, DMA
- 设备驱动
- 同步I/O与异步I/O



# 设备数据传输

- 数据传输

- 启动设备 + 数据传输
- 启动时间（开销）
  - CPU用于启动设备进行操作的时间
- 带宽
  - 启动设备后数据传输的速率
  - Bytes/sec
- 延迟
  - 传输1字节的时间
  - 启动时间+ 将1字节传输到目的地的时间



Device	Transfer rate
Keyboard	10Bytes/sec
Mouse	100Bytes/sec
...	...
10GE NIC	1.2GBytes/sec

- 通用方法：不同的传输速率

- 字符设备：对字节流传输的抽象，键盘、串口、打印机等  
字节粒度访问，以若干字节为传输粒度，顺序读写
- 块设备：以块为存储粒度和传输粒度，按块寻址，整块读写，随机读写



# 数据传输方式

---

- PIO ( Programmed I/O )
- 中断 ( Interrupt )
- DMA ( Direct Memory Access )



# PIO

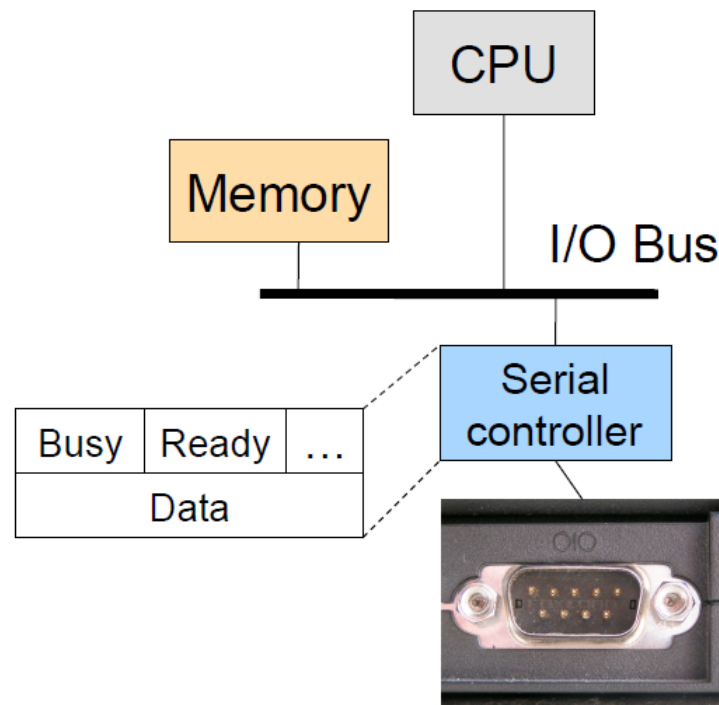
- 例子：RS-232串口
- 简单的串行控制器
  - 状态寄存器：就绪、忙、...
  - 数据寄存器
- 输出数据时

CPU：

- 等待设备状态变为非“忙”
- 写数据到数据寄存器
- 通知设备“就绪”

设备：

- 等待直到状态变为“就绪”
- 清除“就绪”标志，设置“忙”标志
- 从数据寄存器中拿走数据
- 清除“忙”标志





# PIO的轮询 (Polling)

- CPU等待直到设备状态变为非“忙”
  - 轮询：不停地检查设备状态，“忙等”
- 好处
  - 简单
- 坏处
  - 慢（如果设备慢的话）
  - 浪费CPU
- 例子
  - 如果一个设备的速度是100 ops/s，CPU需要等待10ms
  - 对于1GHz的CPU，意味着 1千万个CPU 时钟周期
- 改进：中断机制可避免CPU轮询



# 支持中断的设备

- 例子：鼠标
- 简单的鼠标控制器
  - 状态寄存器（完成、中断、...）
  - 数据寄存器（ $\Delta X$ ,  $\Delta Y$ , 按键）

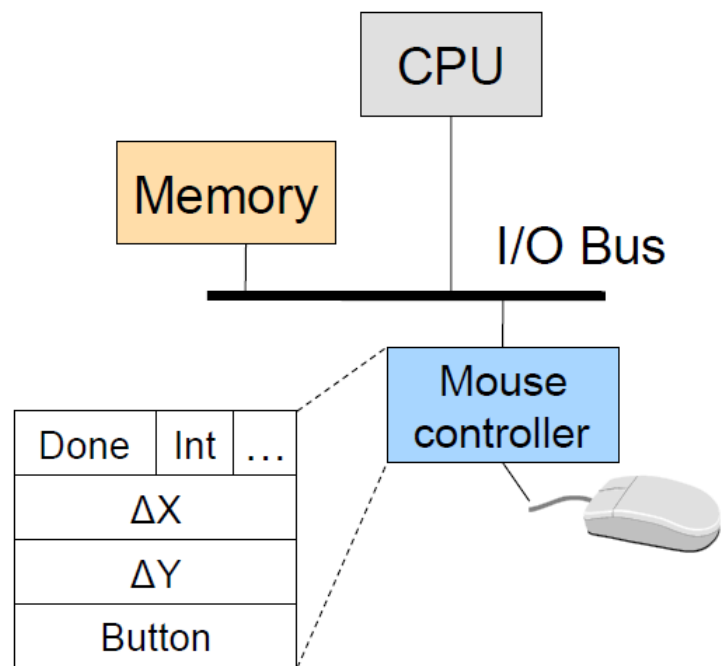
- 输入数据时

## 鼠标

- 等待直到设备状态变为“完成”
- 将 $\Delta X$ ,  $\Delta Y$ 和按键的值保存到数据寄存器
- 发中断

## CPU（中断处理）

- 清除“完成”标志
- 将 $\Delta X$ ,  $\Delta Y$ 和按键的值读到内核缓冲区（内存）中
- 置“完成”标志
- 调用调度器





# DMA (Direct Memory Access)

- 基于数据寄存器的不足
  - 数据寄存器满后，发送中断请求，CPU进行中断处理
  - 传输大量数据时，中断频繁，需要CPU频繁处理中断
- DMA
  - 以数据块为单位进行传输，由DMA控制器控制完成外设与主机间的数据传输
  - DMA需要地址连续的内核缓冲区
  - CPU在数据开始时设置DMA控制器
  - 数据传输结束后，DMA控制器发送中断给CPU，CPU进行处理
  - 减少占用CPU资源



# DMA (Direct Memory Access)

- 例子：磁盘
- 一个简单的磁盘控制器
  - 状态寄存器（完成、中断、...）
  - DMA内存地址和字节数
  - DMA控制寄存器：命令、设备、传输模式及粒度
  - DMA数据缓冲区

- DMA写

CPU：

- 等待DMA设备状态为“就绪”
- 清除“就绪”
- 设置DMA命令为write，地址和大小
- 设置“开始”
- 阻塞当前的线程/进程

磁盘控制器：

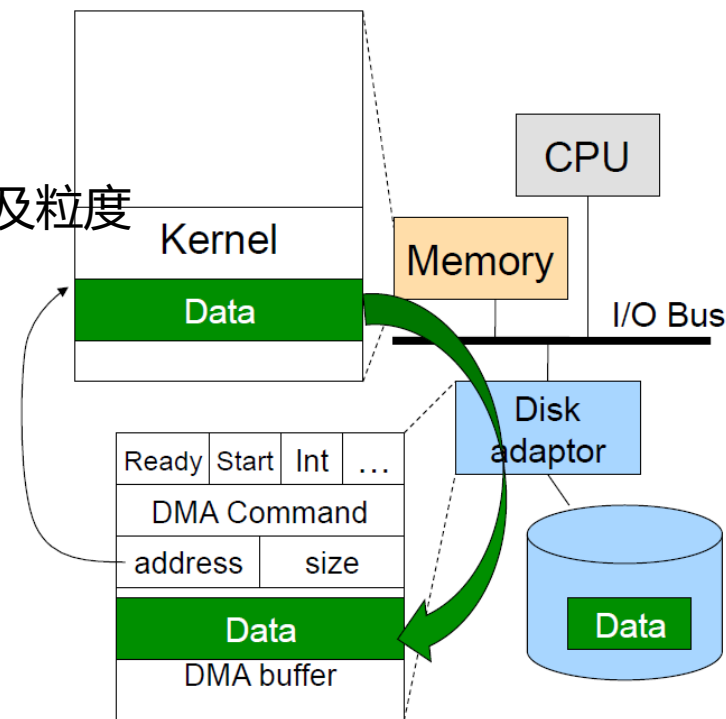
- DMA方式将数据传输到缓冲区，（count--；addr++）
- 当count==0，发中断

CPU（中断处理）：

- 将被该DMA阻塞的线程/进程加到就绪队列

磁盘：

- 将数据从缓冲区写入磁盘







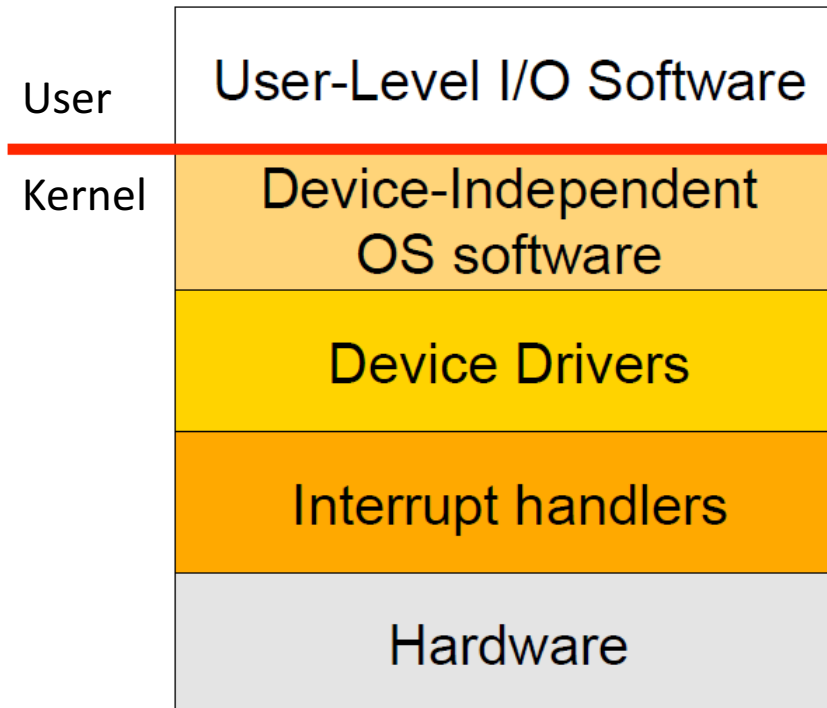
# 内容提要

---

- I/O 设备
  - 设备控制器与缓冲区
  - 寻址
- 数据传输
  - PIO, 中断, DMA
- 设备驱动
- 同步I/O与异步I/O

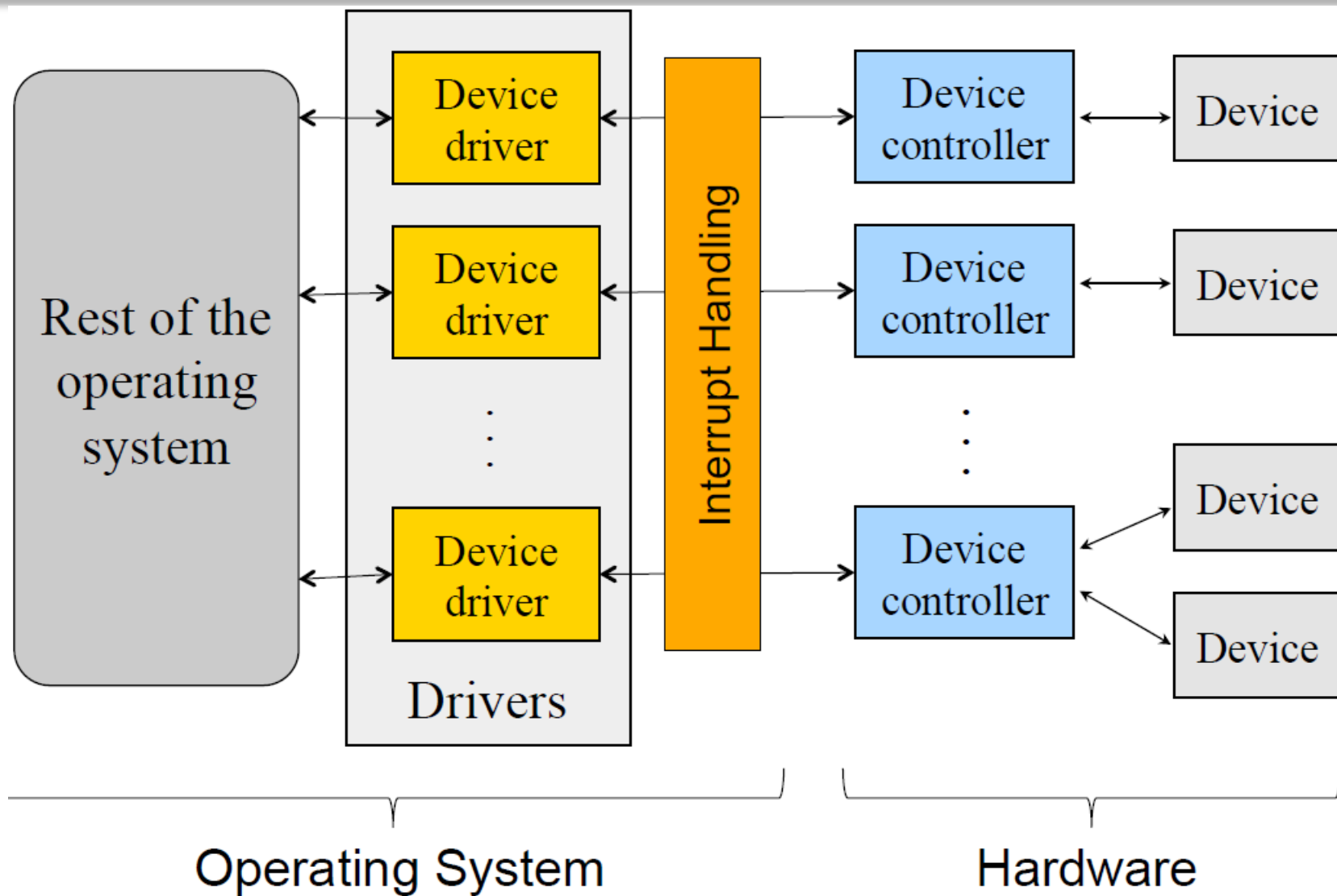


# I/O软件栈





# 设备驱动





# 设备驱动

- 给操作系统的其它模块提供操作设备的API
  - Init, Open, Close, Read, Write, ...
- 与设备控制器交互
  - 与设备控制器交互以进行数据传输：命令、参数、数据
- 主要功能
  - 初始化设备
  - 解析OS发来的命令
  - 多个请求的调度
  - 管理数据传输
  - 接收和处理中断
  - 维护驱动与内核数据结构的完整性



# 设备驱动的主要流程

- 准备工作
  - 参数检查、请求格式转换
  - 设备状态检查：忙→请求入队列
  - 开设备或上电时执行
- 操纵设备
  - 将控制命令写入设备的控制寄存器
  - 检查设备状态：就绪→写下一命令
  - 直到设备完成所有命令
- 阻塞等待
  - 等待设备完成工作
  - 被中断唤醒
  - 有的设备不需要等待：如显示器
- 错误处理
  - 检查设备返回结果，如果返回错误，可能retry
- 返回调用者



# 设备驱动操作接口

- Init (deviceNumber)
  - 初始化硬件
- Open (deviceNumber)
  - 初始化驱动，并在内核中分配资源
- Close (deviceNumber)
  - 回收资源，关闭设备
- 设备驱动的类型
  - 字符设备：可变长度的数据传输
    - 几个字节，e.g. 鼠标，串口
  - 块设备：以固定大小的块为粒度的数据传输



# 设备驱动操作接口

- 字符设备接口
  - read (deviceNumber, bufferAddr, size)
    - 从字节流设备上读 “size” 字节数据
  - write (deviceNumber, bufferAddr, size)
    - 将 “bufferAddr” 中size字节数据写入字节流设备
- 块设备接口
  - read (deviceNumber, deviceAddr, bufferAddr)
    - 从设备传输1个块的数据到内存
  - write (deviceNumber, deviceAddr, bufferAddr)
    - 从内存传输1个块的数据到设备
  - seek (deviceNumber, deviceAddr)
    - 将磁头移动到指定块
    - 现在的设备磁头移动是控制器来控制，对设备驱动透明



# 示例：UNIX设备驱动接口

- `init()`
  - 初始化硬件
- `start()`
  - 开机时初始化，需要系统服务
- `halt()`
  - 在系统关机前要调用
- `open(dev, flag, id)` 和 `close(dev, flag, id)`
  - 初始化资源和释放资源
- `intr(vector)`
  - 在发生硬件中断时由内核调用
- `read(...)` 和 `write(...)`
  - 数据传输
- `poll(pri)`
  - 内核每秒调用25次~100次
- `ioctl(dev, cmd, arg, mode)`
  - 特殊请求处理





# 设备驱动安装

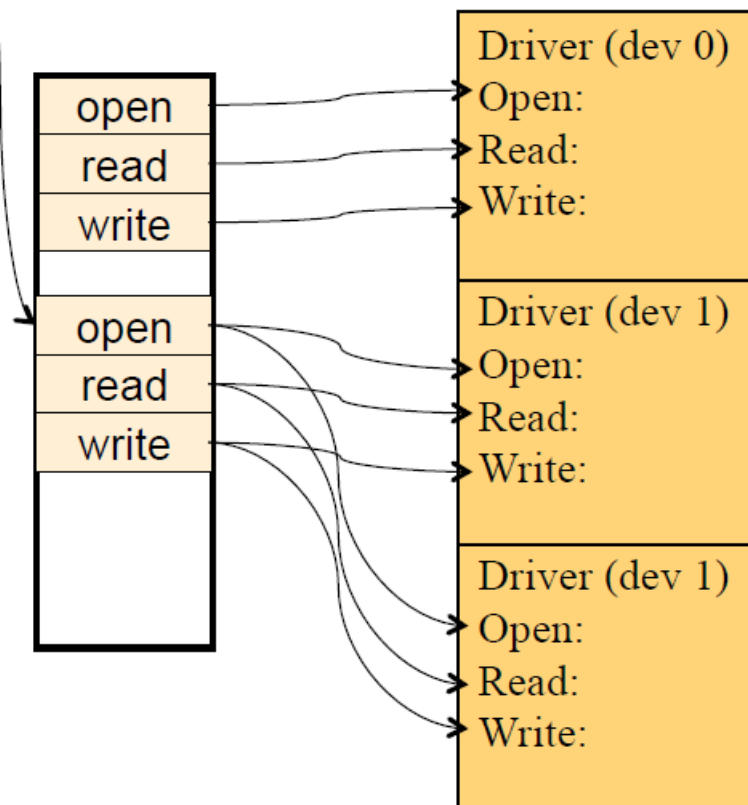
- 静态安装设备驱动
  - 将设备驱动直接编译进内核，系统启动后可以直接调用
  - 新设备的使用需要重启OS
  - 设备驱动修改效率不高，需要重新编译内核
- 动态挂载设备驱动
  - 将驱动动态加载进内核空间
  - 不需要重启，而是采用间接指针
  - 安装入口点，维护相关的数据结构
  - 初始化设备驱动



# 动态绑定设备驱动

- 间接指针
  - 设备入口点表：  
所有设备的入口点
- 加载设备驱动
  - 分配内核空间
  - 存储驱动代码
  - 与入口点关联
- 删除设备驱动
  - 删除入口点
  - 释放内核空间

Open(1,...)





# 设备驱动的利与弊

- 灵活性：
  - 用户可以下载和安装设备驱动
  - 灵活接入不同硬件设备
- 安全隐患
  - 设备驱动运行于内核态
  - 有bug的设备驱动会导致内核崩溃，或者引入安全漏洞
- 如何让设备驱动更安全
  - 检查设备驱动的代码
  - 为设备驱动构建状态机模型
- 用户态驱动：面向高速设备，提升性能



# 内容提要

---

- I/O 设备
  - 设备控制器与缓冲区
  - 寻址
- 数据传输
  - PIO, 中断, DMA
- 设备驱动
- 同步I/O与异步I/O



# 同步I/O与异步I/O

---

- 同步I/O
  - `read( )` 和 `write( )`将阻塞用户进程，直到读写完成
  - 在一个进程做同步I/O时，OS调度另一个进程执行
- 异步I/O
  - `aio_read( )` 和 `aio_write( )`不阻塞用户进程
  - 在I/O完成以前，用户进程可以做别的事
  - I/O完成将通知用户进程



# 异步I/O接口

POSIX P1003.4 异步I/O接口函数  
( Solaris, AIX, Linux 2.6, ...都支持 )

- aio\_read: 异步读
- aio\_write: 异步写
- aio\_fsync: 异步地将缓存脏块写回磁盘，并将errno设置为 ENOSYS
- lio\_listio: 提交一组I/O请求
- aio\_return: 获取异步I/O操作的状态
- aio\_error: 获取异步I/O错误状态
- aio\_cancel: 取消异步读写请求
- aio\_suspend: 挂起直到异步I/O操作完成

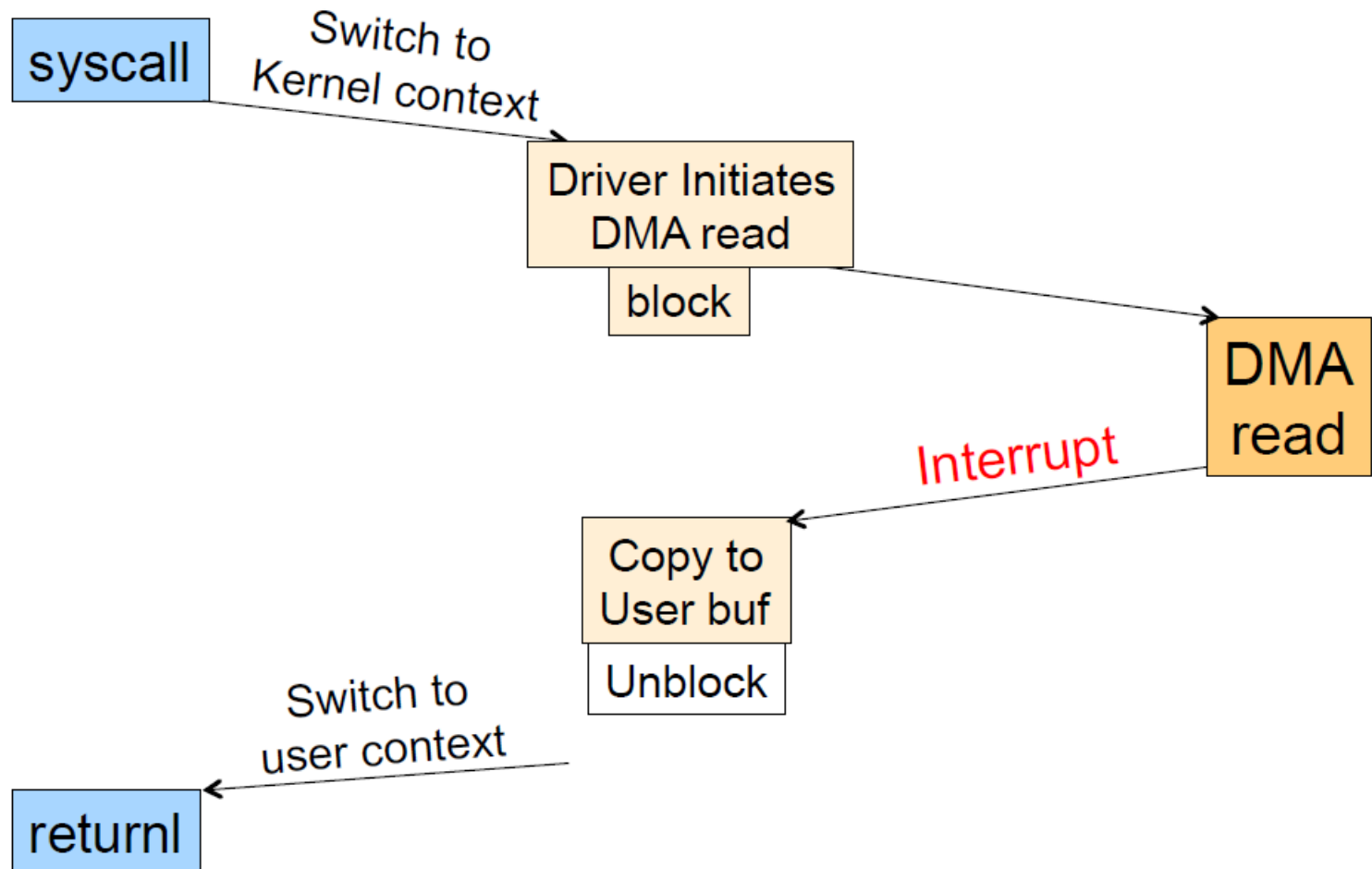


# 同步读

Application

Kernel

HW Device



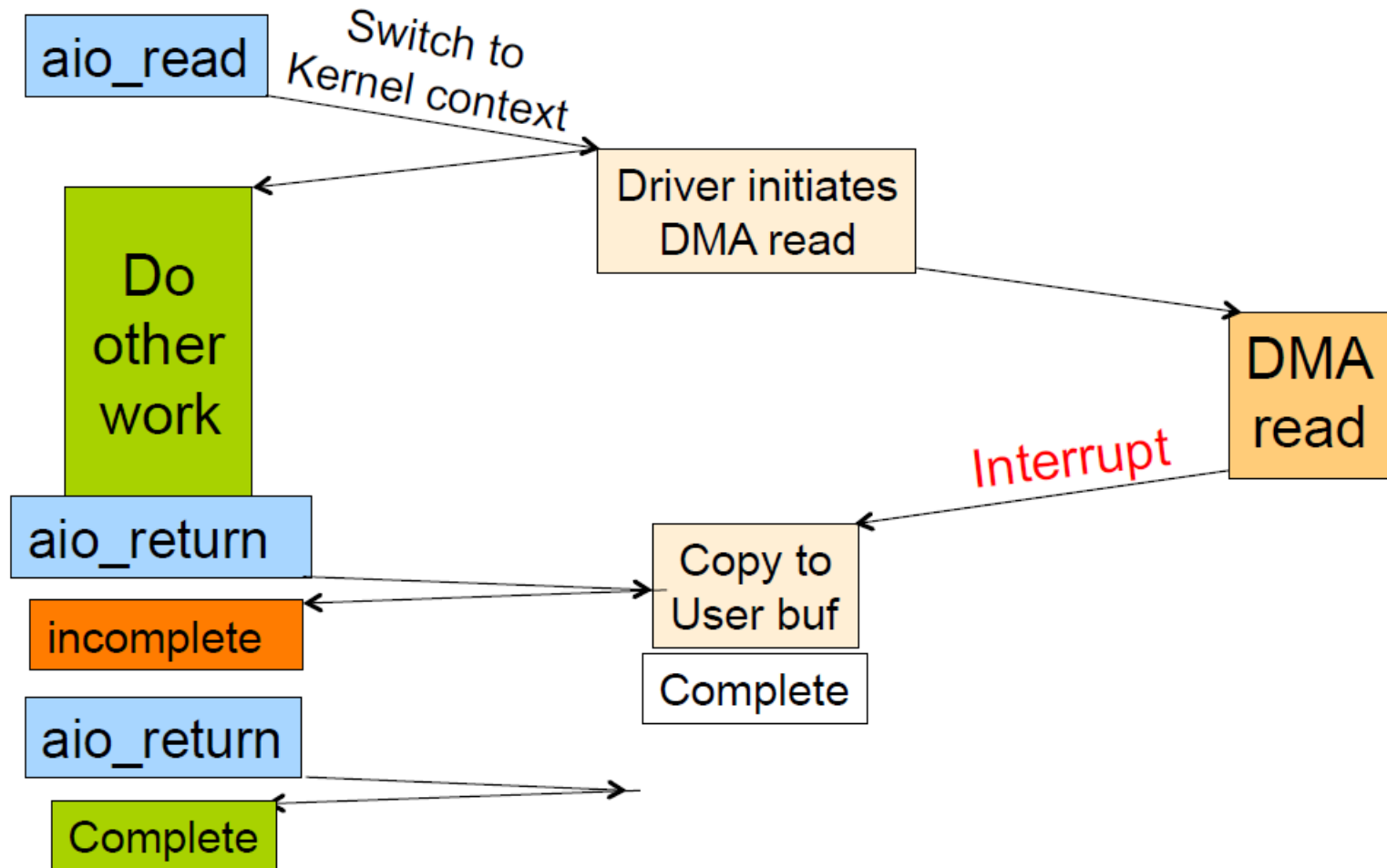


# 异步读

Application

Kernel

HW Device







# 同步读

- 用户进程P1调用read( )系统调用
- 系统调用代码检查正确性和缓存
- 如果需要进行I/O，会调用设备驱动程序
- 设备驱动程序为读数据分配一个buffer，并调度I/O请求
- 启动DMA进行读传输
- **阻塞**当前进程P1，**调度**另一个就绪的进程P2
- 设备控制器进行DMA读传输
- 传输完时，设备发送一个**中断**请求
- 中断处理程序**唤醒**被阻塞的用户进程P1（将P1加入就绪队列）
- 设备驱动检查结果（是否有错误），返回
- 将数据从内核buffer拷贝到用户buffer
- read系统调用返回到用户程序
- 用户进程继续执行



# 总结

- I/O设备
  - PIO简单、但不高效
  - 中断机制支持CPU与I/O重叠
  - DMA比基于数据寄存器的中断更加高效，减少CPU在传输过程的介入
- 设备驱动
  - 直接操纵设备的程序代码
  - OS代码中大量设备驱动代码
  - 设备驱动引入安全漏洞，解决设备驱动的安全隐患是一个open problem
  - 用户态驱动的发展
- 异步I/O
  - 异步I/O允许用户程序的计算与I/O重叠



# 附录：表示大小的单位

单位名称	缩写	$10^n$	缩写	$2^n$	英文 Short scale
yotta	Y	$10^{24}$	Yi	$2^{80}$	septillion
zetta	Z	$10^{21}$	Zi	$2^{70}$	sextillion
exa	E	$10^{18}$	Ei	$2^{60}$	quintillion
peta	P	$10^{15}$	Pi	$2^{50}$	quadrillion
tera	T	$10^{12}$	Ti	$2^{40}$	trillion
giga	G	$10^9$	Gi	$2^{30}$	billion
mega	M	$10^6$	Mi	$2^{20}$	million
kilo	K	$10^3$	Ki	$2^{10}$	hundred
hecto	h	$10^2$			ten
deca	da	$10^1$			one

单位名称	缩写	$10^n$	英文 Short scale
deci	d (分)	$10^{-1}$	tenth
centi	c (厘)	$10^{-2}$	hundredth
milli	m (毫)	$10^{-3}$	thousandth
micro	$\mu$ (微)	$10^{-6}$	millionth
nano	n (纳)	$10^{-9}$	billionth
pico	p (皮)	$10^{-12}$	trillionth
femto	f	$10^{-15}$	quadrillionth
atto	a	$10^{-18}$	quintillionth
zepto	z	$10^{-21}$	sextillionth
yocto	y	$10^{-24}$	septillionth

GB= $10^9$  Bytes vs. GiB= $2^{30}$  Bytes, 类似地, KB vs. KiB, MB vs. MiB, ...