OPERATING SYSTEMS

I /O Systems

Operating Systems

郑贵锋 博士
中山大学计算机学院
zhenggf@mail.sysu.edu.cn
https://gitee.com/code_sysu

中山大學 SUN YAT-SEN UNIVERSITY

- **目录**
  - 概述
  - I/O硬件
  - 应用程序I/O接口
  - 内核I/O子系统
  - 将I/O请求转换为硬件操作
  - **流
  - 性能

I / O Systems

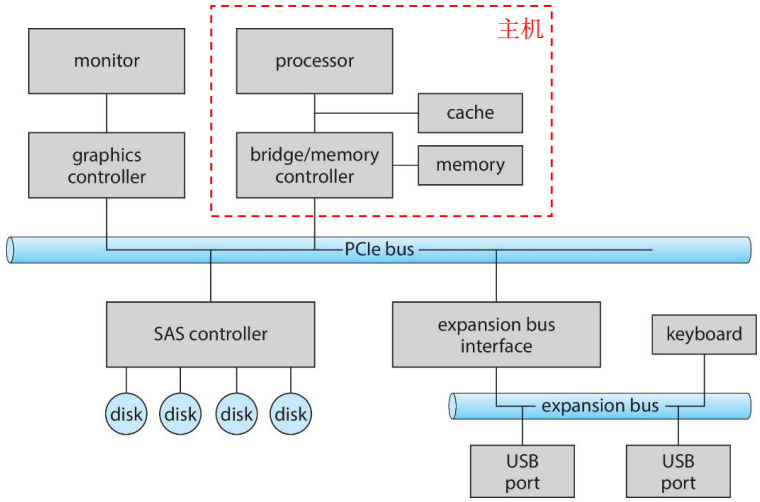
- **概述**
  - 计算机的两个主要工作是I/O和计算。
    - 在许多情况下，主要工作是I/O，而计算或处理只是偶尔的。
  - 操作系统在计算机I/O中的作用是管理和控制I/O设备和I/O操作。对连接到计算机的设备的控制是操作系统设计者主要关心的问题。
    - I/O设备种类繁多，需要多种方法来控制它们。
    - 这些方法构成了内核的I/O子系统，将内核的其余部分与管理I/O设备的复杂性分离。
  - 基本I/O硬件元素，如端口、总线和设备控制器，可适应各种I/O设备。
  - 设备驱动模块用于封装不同设备的细节和反常之处。
    - 它们为I/O子系统提供了一个统一的设备访问接口。

I / O Hardware

- **I/O硬件**
  - 种类繁多的I/O设备可用于：
    - 存储
    - 传输
    - 人机界面
  - 设备通过电缆甚至通过空气发送信号与计算机系统（主机）进行通信。
    - 端口 – 通信的连接点。
    - 总线 – 由设备共享的一组公共电缆，具有严格定义的协议，该协议指定可以在电缆上发送的消息集合。
      - 雏菊花链(daisy chain) 或共享直接访问
      - PCI（外围组件互连）总线常用于PC机和服务器中
      - PCIe (PCI Express)总线
      - 扩展总线连接相对较慢的设备
    - 控制器（主机适配器）– 控制器是可以操作端口、总线或设备的电子设备的集合。
      - 例如，磁盘控制器和FC控制器

I / O Hardware

- **I/O硬件**
  - 一种典型的PC总线结构。



I / O Hardware

- **I/O设备控制寄存器**
  - I/O设备由I/O指令控制。设备通常有寄存器，设备驱动程序在其中放置命令、地址和要写出的数据，或命令执行后从寄存器读取数据
    - 数据输入寄存器：由主机读取以获取输入。
    - 数据输出寄存器：由主机写入以发送输出。
    - 状态寄存器：包含可由主机读取的位。这些位表示状态，如命令结束、数据准备就绪、设备错误等。
    - 控制寄存器：可由主机写入以启动命令或更改设备模式。
      - 例如，全双工/半双工、奇偶校验、速度选择等。
  - 数据寄存器的大小通常为1到4个字节。一些控制器具有FIFO芯片，可容纳若干字节的输入或输出数据，以扩展控制器的容量，超过数据寄存器的大小。FIFO芯片可以保存少量数据，直到设备或主机能够接收这些数据。

I / O Hardware

- **内存映射I/O (MMIO)**
  - 内存映射
    - 设备控制寄存器映射到处理器的地址空间（物理内存）。
    - CPU使用标准数据传输指令执行I/O请求，以在物理内存中的映射位置读取和写入设备控制寄存器。
  - 示例：DriectDraw
    - 图形控制器具有用于基本控制操作的I/O端口，用于保存屏幕内容的大内存映射区域。线程通过将数据写入内存映射区域向屏幕发送输出。控制器根据该内存的内容生成屏幕图像。这项技术使用起来很简单。而且，将数百万字节写入图形内存比发出数百万条I/O指令更快。
  - 如今，大多数I/O是由设备控制器使用内存映射I/O执行的。

I / O Hardware

- **设备I/O端口**
  - PC上的设备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)

轮询

- 在轮询(polling)方案中，主机通过握手与控制器协调，往端口写入输出，过程如下：
  (1) 主机重复读取忙位，直到该位被清除。
  (2) 主机在命令寄存器中设置写入位，并将一个字节写入数据输出寄存器。
  (3) 主机设置命令就绪位。
  (4) 当控制器注意到已设置命令就绪位时，它将设置忙位。
  (5) 控制器读取命令寄存器并查看写入命令。它读取寄存器中的数据以获取字节，并对设备进行I/O操作。
  (6) 控制器清除命令就绪位，清除状态寄存器中的错误位以指示设备I/O成功，清除忙位以指示操作完成。
- 步骤(1)正是忙等待或轮询以等待来自设备的I/O
  - 如果设备速度快，则合理；如果设备速度慢，则效率低。
  - CPU切换到其他任务？
    - 如果错过一个周期，数据将被覆盖或丢失。

中断

- 轮询可以在3个指令周期内发生
  - 读取设备寄存器
  - 逻辑与提取状态位
  - 分支(branch, 跳转)如果非零
  - 如果非零不频繁（总是忙），如何提高轮询效率？
- CPU有一条由I/O设备触发的中断请求线。
  - 处理器在每条指令后检查该行。
- 中断处理程序例程接收中断。
  - 可屏蔽忽略或延迟某些中断
- 中断向量用于将中断分派给正确的处理程序。
  - 开始和结束时的上下文切换
  - 基于优先级
  - 一些不可屏蔽
  - 如果多个设备处于同一中断编号，则中断链接。

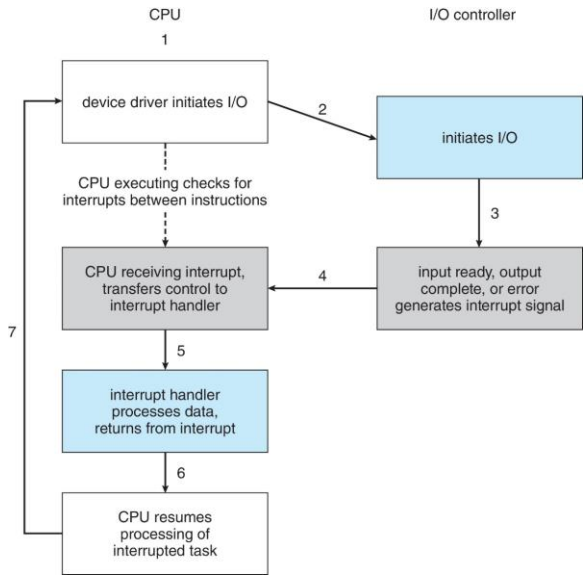
I / O Hardware

- 中断
  - 英特尔奔腾处理器事件向量表

vector number	description
0	divide error
1	debug exception
2	null interrupt
3	breakpoint
4	INTO-detected overflow
5	bound range exception
6	invalid opcode
7	device not available
8	double fault
9	coprocessor segment overrun (reserved)
10	invalid task state segment
11	segment not present
12	stack fault
13	general protection
14	page fault
15	(Intel reserved, do not use)
16	floating-point error
17	alignment check
18	machine check
19–31	(Intel reserved, do not use)
32–255	maskable interrupts

I / O Hardware

- 中断



中断驱动的I/O周期

中断

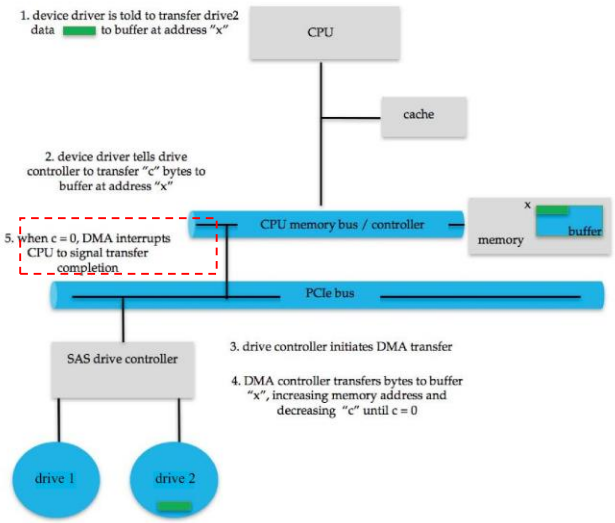
- 中断机制也用于异常
  - 除以零
  - 访问受保护或不存在的内存地址
  - 试图从用户模式执行特权指令
  - 硬件错误
  - ……
- 缺页中断也是引发中断的异常（14号中断）
  - 中断暂停当前进程并跳转到内核中的缺页中断处理程序。
- 陷阱（软件中断）
  - 系统调用通过陷阱执行，以触发内核执行请求。
  - 与分配给设备中断的中断优先级相比，陷阱的中断优先级相对较低。
- 多CPU系统可以同时处理中断。
  - 需操作系统设计时支持
- 问题讨论：中断、故障、陷阱和异常之间的区别。

直接内存访问(DMA)

- DMA用于大量数据移动，避免使用编程I/O（PIO，一次一个字节）
  - 需要DMA控制器
  - 绕过CPU直接在I/O设备和内存之间传输数据。
- OS将DMA命令块写入内存，包括：
  - 源地址和目标地址
  - 读或写模式
  - 字节数
- OS将命令块的位置写入DMA控制器。
  - DMA控制器从CPU获取总线。
    - 周期窃取自CPU，但仍然更有效。
  - 完成后，DMA控制器中断以发出完成信号。
- DVMA – 能够识别虚拟地址的版本可以更加高效。

I / O Hardware

- **直接内存访问**
  - 执行DMA传输的步骤

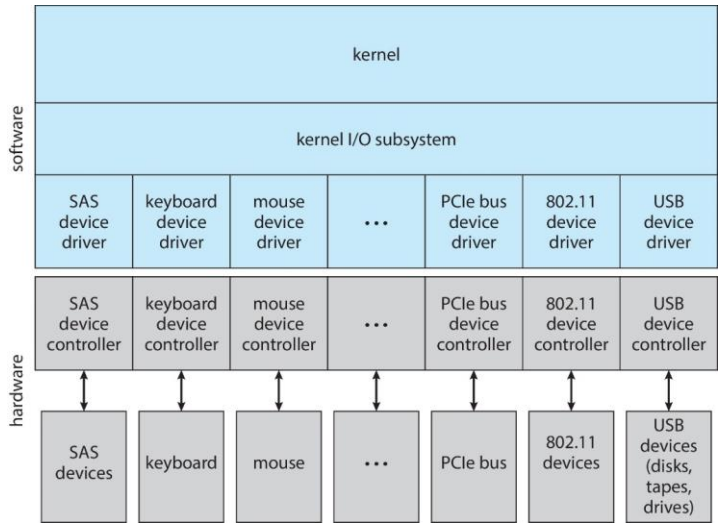


Application I/O Interface

- **应用程序I/O接口**
  - 每个操作系统都有自己的I/O子系统结构和设备驱动程序框架。通过标识一些通用类型，I/O设备中的详细差异被抽象掉了。每种通用类型都可以通过一组标准化的功能 — I/O接口进行访问。
    - I/O系统调用将设备行为封装在通用类中。
    - 设备驱动层对内核隐藏了I/O控制器之间的差异。
    - 使用已实现协议通信的新设备不需要额外的工作。
  - 使I/O子系统独立于硬件简化了操作系统开发人员的工作。这也有利于硬件制造商。

Application I/O Interface

- **应用程序I/O接口**



一种内核I/O结构

Application I/O Interface

- **应用程序I/O接口**
  - I/O设备在多方面存在差异
    - 数据传输模式
      - 字符流或块
    - 访问模式
      - 顺序存取或随机存取
    - 传输调度
      - 同步或异步（或两者兼有）
    - 共享
      - 可共享的或专用的
    - 操作速度
      - latency(处理等待)延迟、寻道时间、传输速率、delay延迟等
    - I/O方向
      - 读写、只读或只写

Application I/O Interface

- **应用程序I/O接口**

aspect	variation	example
data-transfer mode	character block	terminal disk
access method	sequential random	modem CD-ROM
transfer schedule	synchronous asynchronous	tape keyboard
sharing	dedicated sharable	tape keyboard
device speed	latency seek time transfer rate delay between operations	
I/O direction	read only write only read–write	CD-ROM graphics controller disk

I/O设备的特性

Application I/O Interface

- **应用程序I/O接口**
  - I/O设备的特性
    - 连续分配
    - 由设备驱动器处理设备的细节
    - I/O设备可以由操作系统大致分为：
      - 块I/O
      - 字符I/O（流I/O）
      - 内存映射文件访问
      - 网络套接字
    - 对于直接操作I/O设备的特定特性，通常使用转义 (escape，或后门) 将任意命令从应用程序透明地传递给设备驱动程序。
      - Unix ioctl()调用用于将任意位发送到设备控制寄存器，并将数据发送到设备数据寄存器。

Application I/O Interface

- **块和字符设备**
  - 块设备包括磁盘驱动器。
    - 命令包括read(), write(), seek()
    - 原始I/O，直接I/O，或文件系统访问
    - 内存映射文件访问是可能的
      - 将文件映射到虚拟内存和聚簇，通过请求调页调入。
    - DMA
  - 字符设备包括键盘、鼠标、串行端口。
    - 命令包括get(), put()
    - 上层的库允许行编辑

Application I/O Interface

- **网络设备**
  - 网络设备因块和字符的差异较大，因此有自己的接口。
  - Linux、Unix、Windows和许多其他系统都包括套接字接口。
    - 将网络协议与网络操作分离。
    - 包含select()查询功能。
  - 方法千差万别。
    - 管道、FIFO、流、队列、邮箱。

Application I/O Interface

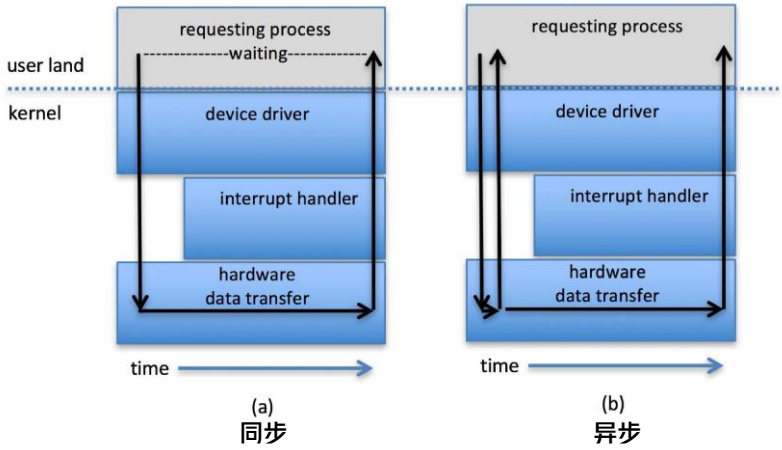
- **时钟与定时器**
  - 提供当前时间、经过的时间、定时器
  - 正常分辨率约为1/60秒
    - 有些系统提供更高分辨率的定时器。
  - 用于定时、周期性中断的可编程间隔定时器
  - UNIX上的ioctl()涵盖了I/O的一些非一般操作，如时钟和定时器。

Application I/O Interface

- **非阻塞和异步I/O**
  - 阻塞 - 进程暂停，直到I/O完成
    - 易于使用和理解
    - 不足以满足某些需要
  - 非阻塞 - I/O调用返回尽可能多的可用数据
    - 用户界面，数据拷贝（缓冲I/O）
    - 通过多线程实现
    - 快速返回读取或写入的字节数
    - select()以查询数据是否已就绪，再选择read()或write()进行传输
  - 异步 - 进程在I/O执行时运行
    - 难以使用
    - I/O完成时，I/O子系统发信号给相关进程处理

Application I/O Interface

- **非阻塞和异步I/O**
  - 两种I/O方法 — 同步和异步



Application I/O Interface

- **向量I/O**
  - 向量 (Vectored) I/O允许一个系统调用执行多个I/O操作
    - 例如，Unix readve()接受多个缓冲区的向量，并从源读入该向量或从该向量写入目标。
  - 这种分散收集(scatter-gather)方法可能优于多个单独的I/O系统调用
    - 它可以减少上下文切换和系统调用开销。
    - 有些版本提供原子性。
      - 确保所有I/O都在不中断的情况下完成
      - 如果其他线程也在执行涉及这些缓冲区的I/O，则避免数据损坏。

Kernel I/O Subsystem

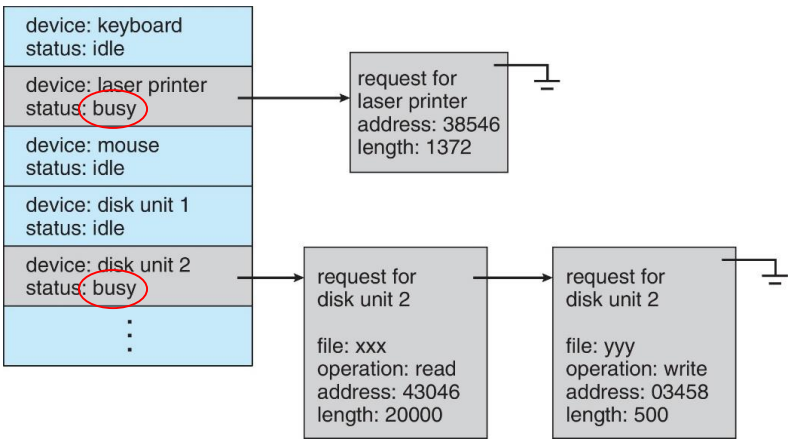
- **内核I/O子系统**
  - 内核的I/O子系统提供了几种基于硬件和设备驱动程序基础设施的服务。
    - 调度
    - 缓冲Buffering
    - 缓存Caching
    - 假脱机SPOOLing
    - 设备预订reservation
    - 错误处理
  - I/O子系统还负责保护自身免受错误进程和恶意用户的攻击。

Kernel I/O Subsystem

- **I/O调度**
  - I/O调度程序重排每个设备的请求的等待队列的顺序
    - 提高系统的整体性能
    - 在进程之间公平地共享设备访问
    - 减少等待I/O完成的平均等待时间
  - 设备状态表
    - 包含每个I/O设备的条目，指示设备的类型、地址和状态。
      - 设备处于三种状态之一
        - 不起作用
        - 闲的
        - 忙的
      - 如果设备忙于服务某请求，则会声明该请求的信息。
    - 内核管理这个表，支持异步I/O，同时跟踪所有I/O请求，调度I/O操作。

Kernel I/O Subsystem

- **I/O调度**
  - 设备状态表

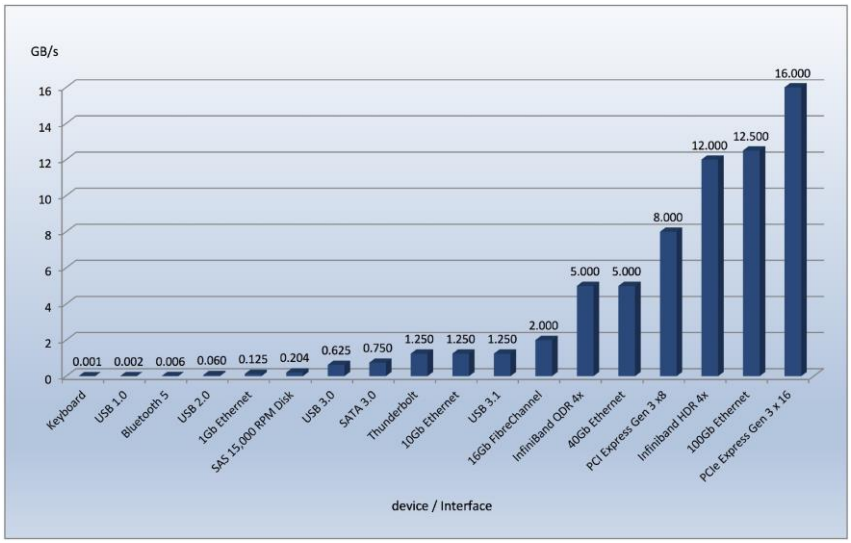


Kernel I/O Subsystem

- **缓冲**
  - 缓冲 - 在设备之间传输时将数据存储在内存中。
    - 处理设备速度不匹配问题
    - 处理设备传输大小不匹配
    - 维护“复制语义”
  - 复制语义 – 操作系统保证写入磁盘的数据版本是应用程序系统调用时的版本，与应用程序缓冲区中的任何后续更改无关。
    - 例如，对于write()系统调用，操作系统在将控制权返回应用程序之前将应用程序数据复制到内核缓冲区。磁盘写入是从内核缓冲区执行的，因此对应用程序缓冲区的后续更改不会产生任何影响。
  - 双缓冲 – 数据的两个副本。
    - 内核和用户
    - 大小不一
    - 已满/正在处理中且未满/正在使用
    - 在某些情况下，写时复制可用于提高效率

Kernel I/O Subsystem

- **缓冲**
  - 通用PC和数据中心I/O设备和接口速度。



Kernel I/O Subsystem

- **缓存**
  - 缓存是一个快速内存区域，用于保存数据副本。
    - 对缓存副本的访问比对原始数据的访问更高效。
  - 缓冲区和缓存是不同的。
    - 缓冲区可以保存数据项的唯一现有副本，而根据定义，缓存可以保存在更快存储上的位于其他设备位置的项的副本。
  - 缓存和缓冲是不同的功能，但有时可将一个内存区域用作两个目的
    - 当内核接收到文件I/O请求时，内核首先访问缓冲区缓存，以查看文件的该区域是否已在主内存中可用。如果是，则可以避免或推迟物理磁盘I/O
    - 磁盘的写入在缓冲区缓存中累积几秒钟，以便收集大量传输以实现高效的写入调度

假脱机和设备预订

- 假脱机(SPOOL)是一个缓冲区，用于保存无法接受交错数据流的设备（如打印机）的输出。
  - 某些设备（如磁带机和打印机）无法有效地多路复用多个并发应用程序的I/O请求。
- 假脱机是操作系统协调此类并发问题的一种方法。
  - 每个应用程序的输出都被假脱机到一个单独的辅助存储文件中
  - 当应用程序完成打印时，假脱机系统将相应的假脱机文件排入队列，以便输出到打印机。
  - 操作系统提供了一个控制界面，允许用户和系统管理员显示和维护假脱机队列。
- 设备预订是另一种处理并发设备访问的方法，它明确提供协作支持，实现设备的互斥访问。
  - 应用程序应该注意死锁。

错误处理

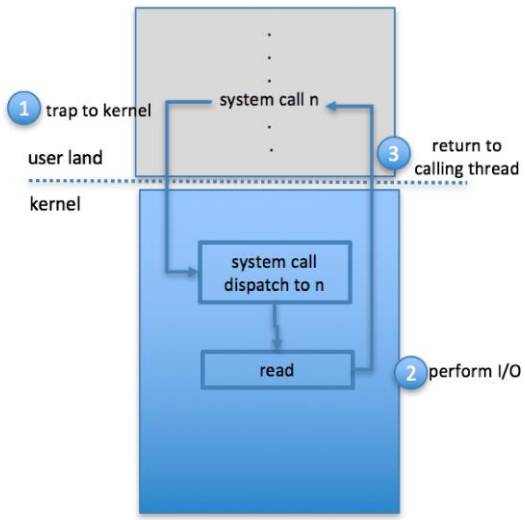
- 设备和I/O传输可能以多种方式失败。
  - 对于瞬时原因，如当网络过载时。
    - 操作系统通常可以通过重试请求来有效补偿瞬时故障。
  - 对于永久性的原因，如磁盘控制器出现故障。
    - 不幸的是，如果一个重要组件发生永久性故障，操作系统不太可能恢复。
- 通常，I/O系统调用将返回有关调用状态的信息，表示调用成功或失败。
  - 例如，Linux中的errno。
- 系统错误日志保存问题报告。

Kernel I/O Subsystem

- **I/O保护**
  - 用户进程可能会意外或故意试图通过发出非法I/O指令来中断系统的正常操作。
    - 错误与保护问题密切相关。
  - 保护：
    - 所有I/O指令都定义为特权指令。用户程序不能直接发出I/O指令，而是执行系统调用以请求操作系统代表其执行I/O。
    - 内存保护系统保护所有内存映射和I/O端口内存位置，以防止用户直接访问。

Kernel I/O Subsystem

- **I/O保护**
  - 使用系统调用执行I/O

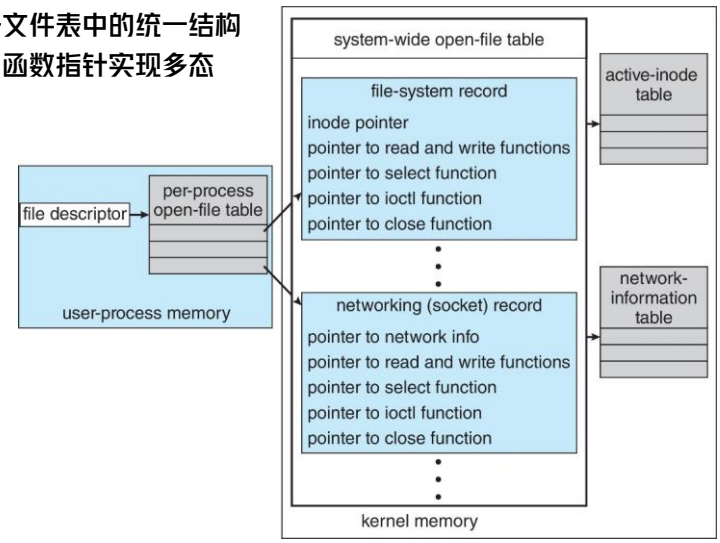


Kernel I/O Subsystem

- **内核数据结构**
  - 内核保存I/O组件的状态信息，包括打开的文件表、网络连接、字符设备状态等。
  - 许多复杂的数据结构用于跟踪缓冲区、内存分配、“脏”块等。
  - 一些操作系统使用面向对象的方法和消息传递来实现I/O。
    - Windows使用消息传递
      - 带有I/O信息的消息从用户模式传递到内核
      - 消息在流向设备驱动程序并返回到进程时被修改
      - 优点/缺点：消息传递方法，与采用共享数据结构的程序调用技术相比
        - 可能增加开销
        - 但是它简化了I/O系统的结构和设计，并增加了灵活性

Kernel I/O Subsystem

- **内核数据结构**
  - UNIX I/O内核结构
    - 打开文件表中的统一结构
    - 使用函数指针实现多态



Kernel I/O Subsystem

- **电源管理**
  - 电源管理通常基于设备管理。
    - 在引导时，固件系统分析系统硬件并在RAM中创建设备树，以管理与设备相关的活动，包括热拔插，了解和更改设备状态，以及电源管理。
    - 严格来说，它不属于I/O领域，但与I/O有很大关系。
    - 操作系统可以帮助管理和改进。
    - 移动计算将电源管理作为第一类操作系统问题。
  - 高级配置和电源接口(ACPI)
    - ACPI是一组固件代码，是现代通用计算机用于管理硬件这些方面的行业标准。
      - ACPI提供的代码作为内核可调用的例程运行，用于设备状态发现和管理、设备错误管理和电源管理。
      - 例如，当内核需静默一个设备时，它调用设备驱动程序，设备驱动程序调用ACPI例程，然后ACPI例程与设备对话。

Other Issues

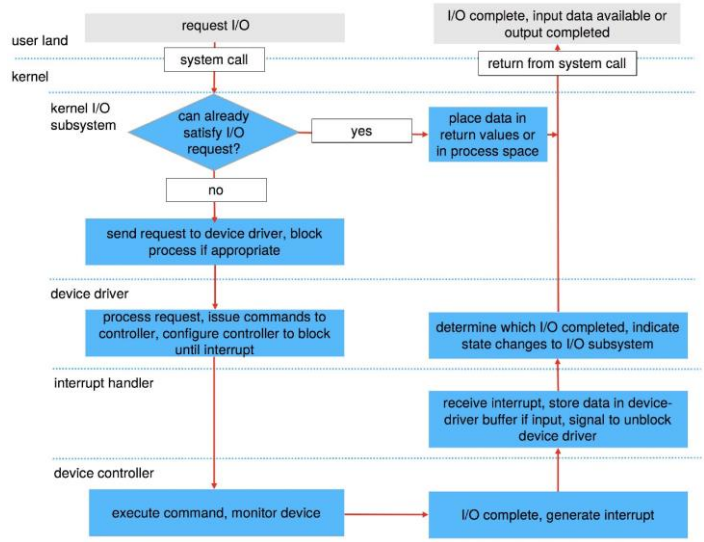
- **将I/O请求转换为硬件操作**
  - 考虑一个阻塞读取请求，从磁盘获取数据。
    (1) 进程对已打开文件的文件描述符发出阻塞read()系统调用。
    (2) 内核中的系统调用代码检查参数的正确性。在输入的情况下，如果数据已经在缓冲区缓存中可用，则数据将返回到进程，并且I/O请求完成。
    (3) 否则，必须执行物理I/O. 进程将从运行队列中删除，并放置在设备的等待队列中，I/O请求将被调度。最后，I/O子系统将请求发送到设备驱动程序。根据操作系统的不同，请求通过子程序调用或内核内消息发送。
    (4) 设备驱动程序分配内核缓冲空间以接收数据并调度I/O. 最终，驱动程序通过写入设备控制寄存器向设备控制器发送命令。
    (5) 设备控制器操作设备硬件以执行数据传输。

Other Issues

- **将I/O请求转换为硬件操作**
  - 考虑一个阻塞读取请求，从磁盘获取数据。
    - (6) 驱动程序可能会轮询状态和数据，也可能已经设置了到内核内存的DMA传输。假设传输由DMA控制器管理，该控制器在传输完成时生成中断。
    - (7) 通过中断向量表，正确的中断处理程序接收中断，存储任何必要的数据，向设备驱动程序发送信号，并从中断返回。
    - (8) 设备驱动程序接收信号，确定哪个I/O请求已完成，确定请求的状态，并向内核I/O子系统发出请求已完成的信号。
    - (9) 内核将数据或返回代码传输到请求进程的地址空间，并将进程从等待队列移回就绪队列。
    - (10) 将进程移动到就绪队列将取消阻塞该进程。当调度程序将进程分配给CPU时，进程将在系统调用完成时恢复运行。

Other Issues

- **将I/O请求转换为硬件操作**
  - I/O请求的生命周期

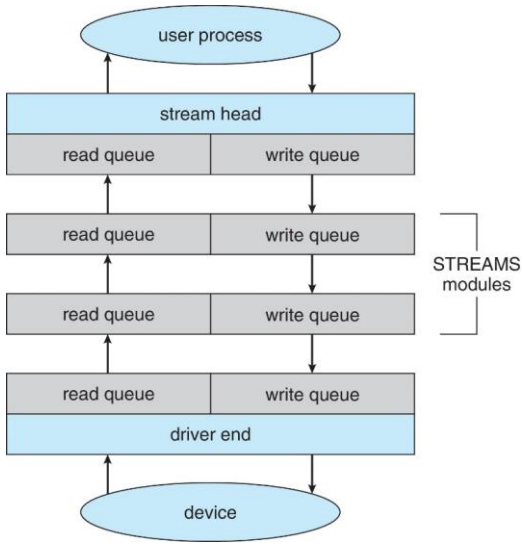


Other Issues

- **流**
  - 流(STREAMS), 在Unix System V及更高版本中，使应用程序能够动态地组装驱动程序代码管道。流是用户级进程和设备驱动程序之间的全双工通信信道。
  - 流包括：
    - 为用户进程提供接口的流头。
    - 控制设备的驱动端，它必须响应中断，处理所有传入的数据。
    - 在流头和驱动端之间的零个或多个流模块。
  - 每个流模块包含一个读队列和一个写队列。
    - 消息传递用于在队列之间传输数据。
    - 如果流头无法将消息复制到管道上的下一个队列，则流头可能会阻塞。
  - 模块提供流处理功能；通过使用ioctl()系统调用（在<stropts.h>中），模块被推送到流上。
    - 例如，进程可以通过流打开USB设备，并可以将模块推送到流以处理输入编辑。

Other Issues

- **流**
  - 流结构



Other Issues

- 流
  - 流量控制
    - 消息在相邻模块中的队列之间交换。一个模块中的队列可能会超过相邻队列。为了防止这种情况发生，队列可以支持流控制以指示可用或繁忙。
    - 在没有流量控制的情况下，队列接受所有消息，并立即将它们发送到相邻模块中的队列，而不缓冲它们。
    - 使用流量控制，队列缓冲消息，并且在没有足够缓冲空间的情况下不接受消息。此过程涉及相邻模块中队列之间的控制消息交换。
    - 驱动程序还必须支持流量控制。

Other Issues

- 流
  - 通过流进行读写
    - 用户进程使用write()或putmsg()系统调用将数据写入设备。write()系统调用将原始数据写入流，而putmsg()允许用户进程指定消息。
      - 流头将数据复制到消息中，并将其传递到队列中，供下一个模块使用。此消息将继续被复制，直到消息复制到驱动程序端，从而复制到设备。
    - 类似地，用户进程使用read()或getmsg()系统调用从流头读取数据。如果使用read()，则流头从其相邻队列获取消息，并将普通数据（非结构化字节流）返回给进程。如果使用getmsg()，则会向进程返回一条消息。

Other Issues

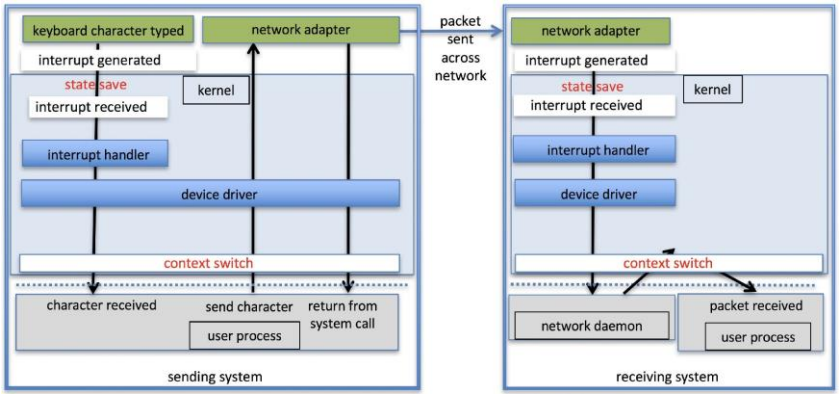
- **流**
  - 异步和同步操作
    - 除用户进程与流头通信外，流I/O是异步（或非阻塞）的。
    - 当写入流时，假设下一个队列使用流量控制，用户进程将阻塞，直到有空间复制消息。
    - 同样，当从流中读取数据时，用户进程将阻塞，直到数据可用为止。

Other Issues

- **性能**
  - I/O是影响系统性能的主要因素
    - I/O对CPU提出了很高的要求，以执行设备驱动程序代码，并在进程被阻塞和解除阻塞时公平高效地调度进程。由此产生的上下文切换会给CPU及其硬件缓存带来压力。
    - I/O还暴露了内核中中断处理机制的任何低效性。中断处理是一项相对昂贵的任务。每个中断都会导致系统执行状态更改、执行中断处理程序，然后恢复状态。
    - 在控制器和物理内存之间的数据复制期间，以及在内核缓冲区和应用程序数据空间之间的复制期间，I/O会加载内存总线。优雅地处理所有这些需求是计算机架构师最关心的问题之一。
  - 考虑计算机间的通信，网络流量也会导致高上下文切换率。

Other Issues

- **性能**
  - 计算机间通信

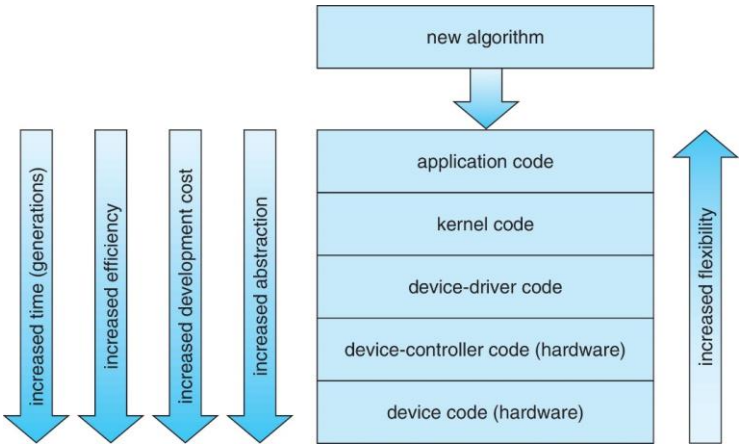


Other Issues

- **性能**
  - 改进性能
    - 一些系统为终端I/O使用单独的前端处理器，以减少主CPU上的中断负担。
      - 终端集中器
      - I/O通道
    - 我们可以采用几种原则来提高I/O的效率。
      - 减少上下文切换的次数
      - 减少在设备和应用程序之间传输数据时必须在内存中复制的次数。
      - 通过使用大数据量传输、智能控制器和轮询（如果忙等待最小化）减少中断频率。
      - 通过使用DMA智能控制器或通道进行简单的数据拷贝，以减轻CPU负载，提高并发性。
      - 将处理原语移植到硬件中，以允许它们在设备控制器中操作，与CPU和总线操作并发。
      - 平衡CPU、内存子系统、总线和I/O性能，因为任何一个区域的过载都会导致其他区域的空闲。

Other Issues

- **性能**
  - 设备功能进展
    - I/O功能应在何处实现？设备硬件、设备驱动程序或应用程序软件中?



Other Issues

- **性能**
  - 设备功能进展。
    - 最初，让I/O算法在应用程序级别实现。
      - 应用程序代码灵活，应用程序错误不太可能导致系统崩溃
      - 设备驱动程序不需要在每次代码更改后重新启动或重新加载。
      - 它可能效率低下。
        - 上下文切换的开销
        - 应用程序无法利用内部内核数据结构和内核功能。
    - 内核内实现可以提高性能，但开发工作更具挑战性。
    - 通过在设备或控制器中的专用硬件实现，可以获得最高性能。随着开发时间的延长和灵活性的降低，这一过程既困难又昂贵

Other Issues

- **性能**
  - 存储的I/O性能
    - 随着时间的推移，与计算的其他方面一样，I/O设备的速度一直在提高。
      - 非易失性存储设备越来越受欢迎，可用设备种类也越来越多。
      - NVM设备的速度从高到超高进化，下一代设备的速度将接近DRAM
    - 在这些发展情况下，为利用现有更快的读/写速度，I/O子系统以及操作系统算法的压力不断增加。

Other Issues

- **性能**
  - 存储的I/O性能
    - CPU缓存/SRAM、DIMM（双列直插式存储模块）DRAM、NVDIMM、PCIe/NVMe NVM、PCIe/NVMe SSD、SAS（串行连接SCSI）SSD、SAS HDD、SATA（串行高级技术连接）HDD…