操作系统

Operating system

胡燕

大连理工大学

内容纲要

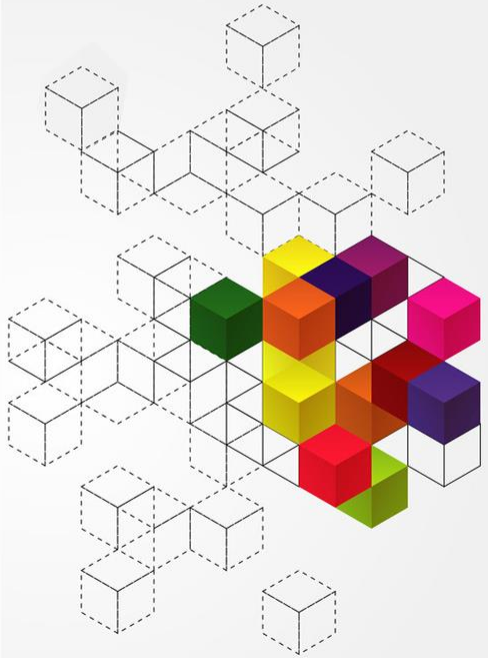
13.5 IO软件设计

一、 IO软件设计层次

二、 中断与设备驱动层

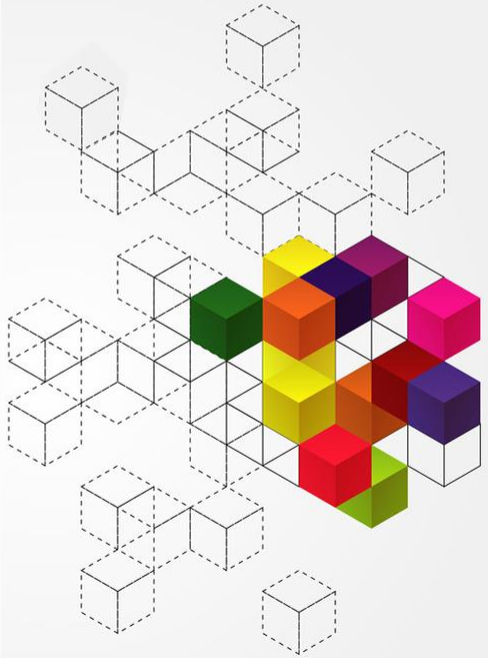
三、 设备无关IO软件层

四、 用户层IO软件模块



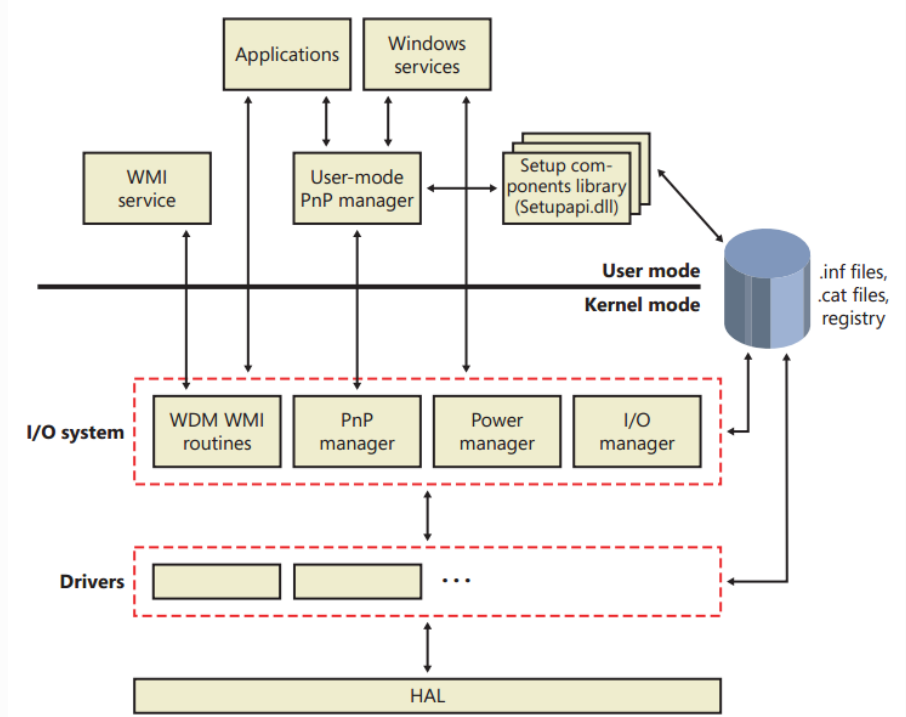
一、IO软件设计层次

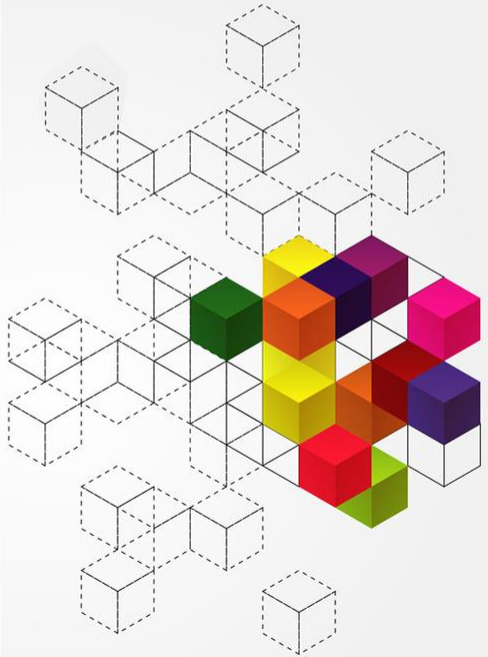
IO软件的设计思路：把IO软件组织成层次结构，低层软件用来屏蔽硬件细节，高层软件向用户提供简洁、友善的界面



一、IO软件设计层次

IO软件分层设计示例：Windows IO子系统



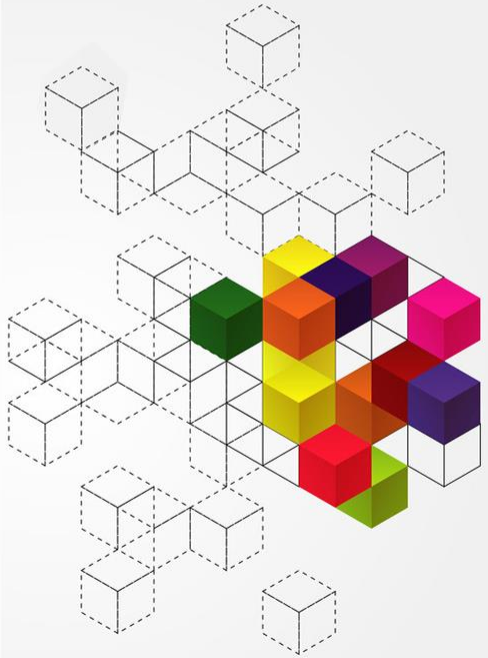


二、I/O中断处理程序与设备驱动程序

I/O中断处理程序

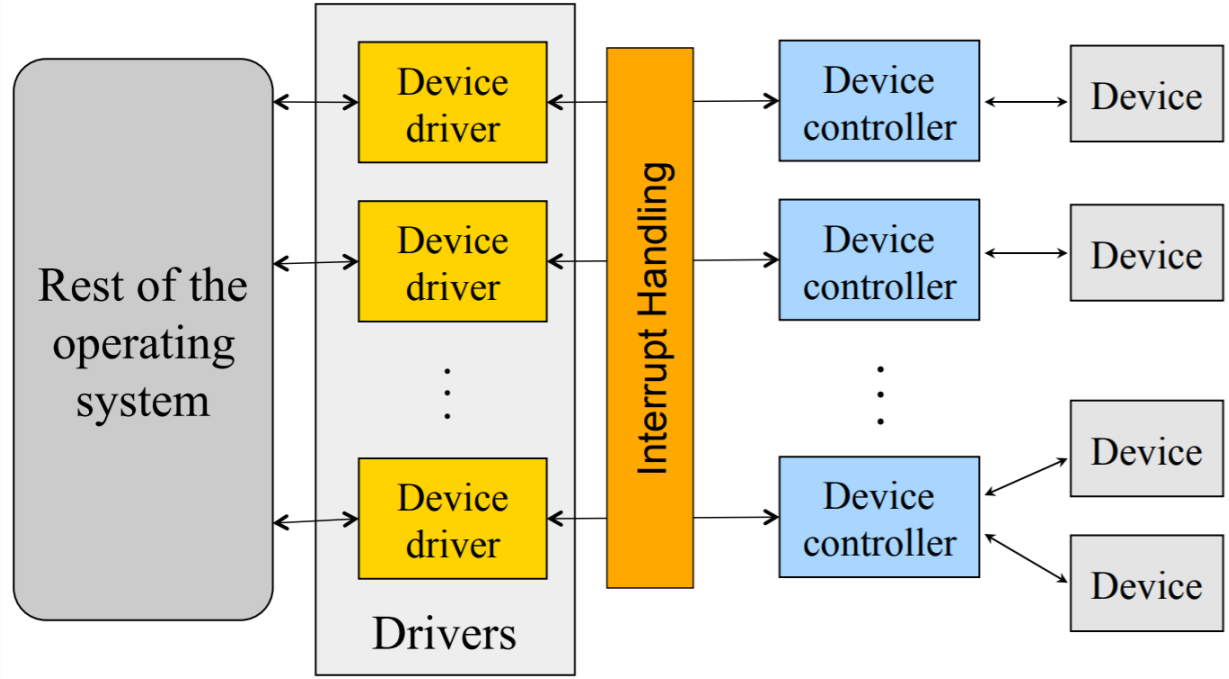
I/O中断处理程序，位于操作系统底层，与硬件设备密切相关

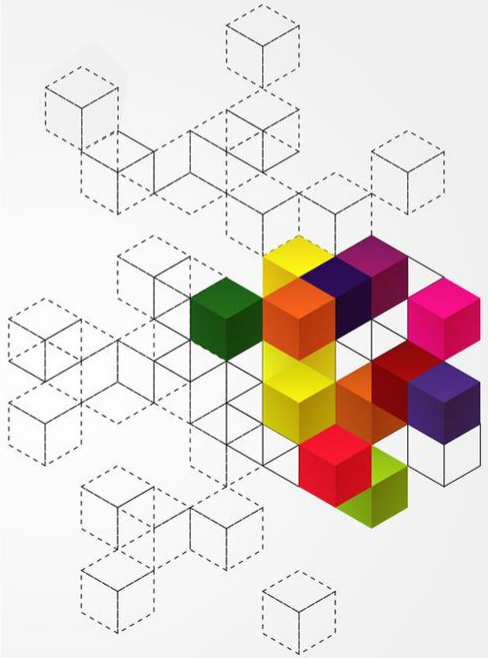
功能：
检查设备状态寄存器，判断发生中断的原因，根据I/O操作的完成情况进行相应处理
如果数据传输有错，向上层软件报告设备的出错信息
如果I/O正常结束，则唤醒等待传输完成的进程，使其转为就绪态



二、 I/O中断处理程序与设备驱动程序

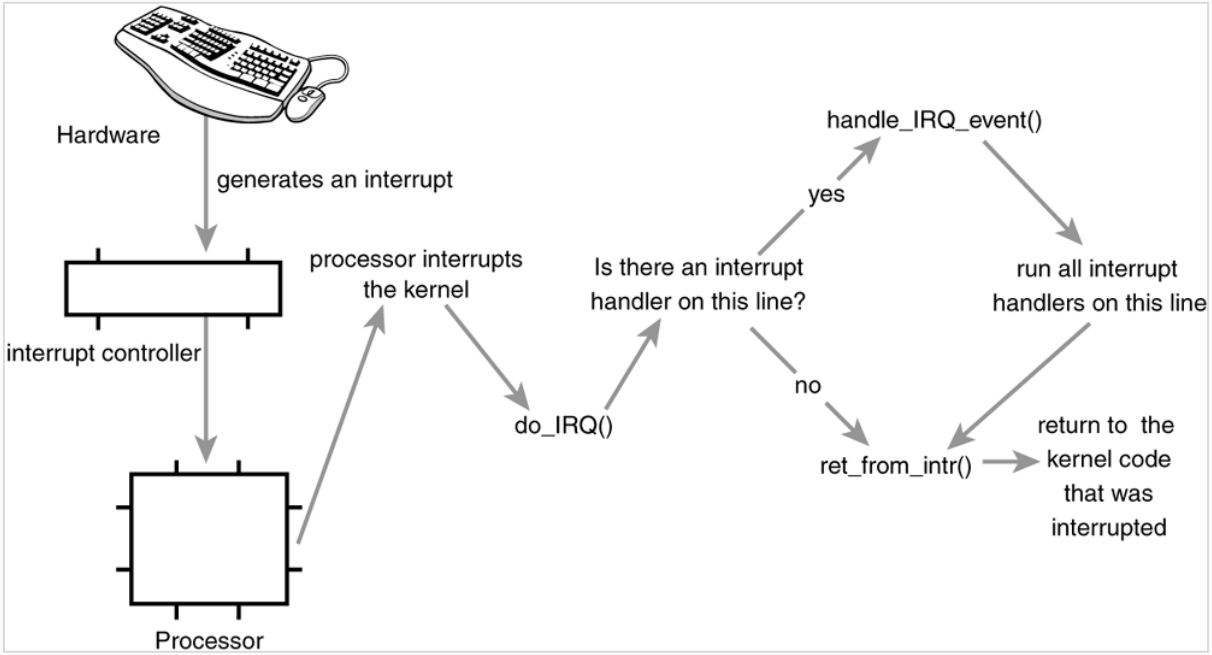
Interrupt Handler

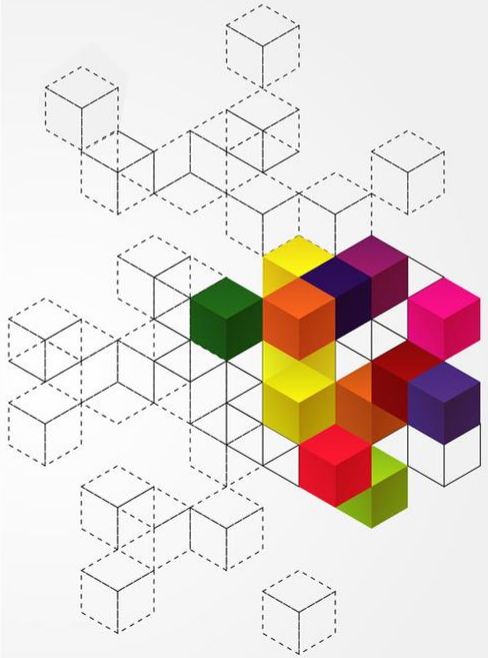




二、 I/O中断处理程序与设备驱动程序

Keyboard&mouse Interrupt Handling

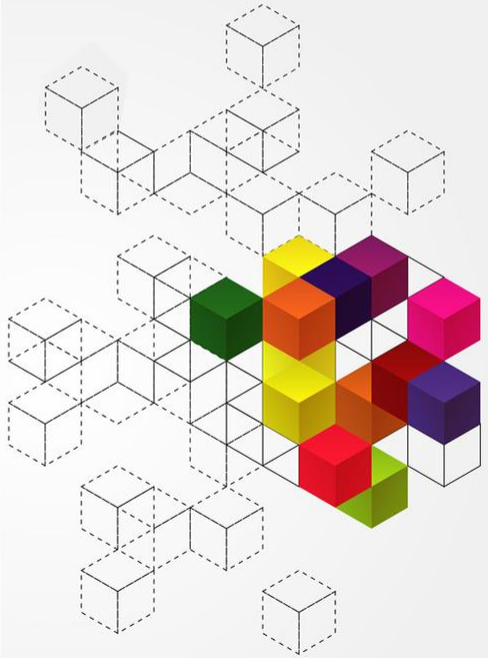




二、I/O中断处理程序与设备驱动程序

Device Drivers

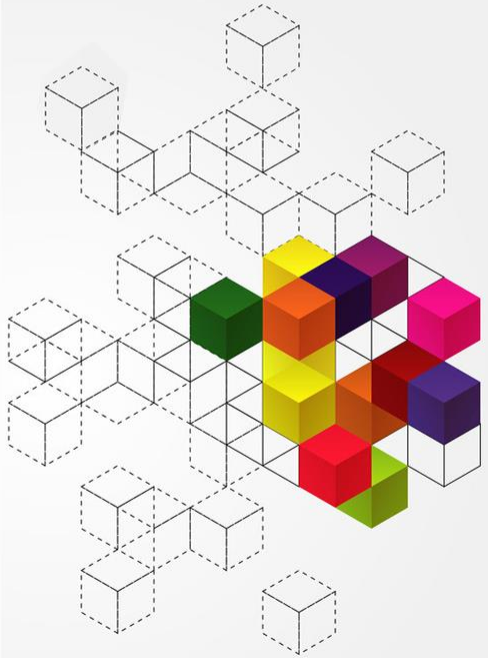
设备驱动程序是操作系统中用于对设备进行操作或控制的代码，为操作系统和其他应用程序使用硬件提供易用的软件接口。
良好设计的设备驱动程序，应该能够使得应用程序在访问设备时无需了解硬件设备的工作细节。



二、 I/O中断处理程序与设备驱动程序

What Device Drivers Do

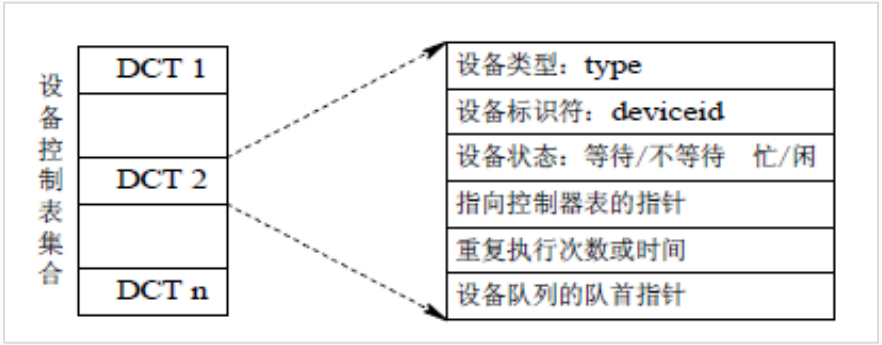
- Provide “the rest of OS” with APIs
  - Init, Open, Close, Read, Write, …
- Interface with controllers
  - Commands and data transfers with hardware controllers
- Driver operations
  - Initialize devices
  - Interpreting outstanding requests
  - Managing data transfers
  - Accept and process interrupts
  - Maintain the integrity of driver and kernel data structures

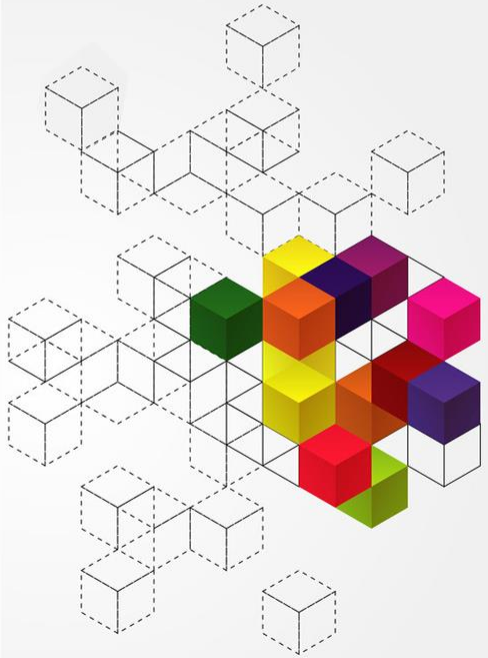


三、设备无关IO软件层

IO设备管理表。

DCT
SDT
COCT
CHCT

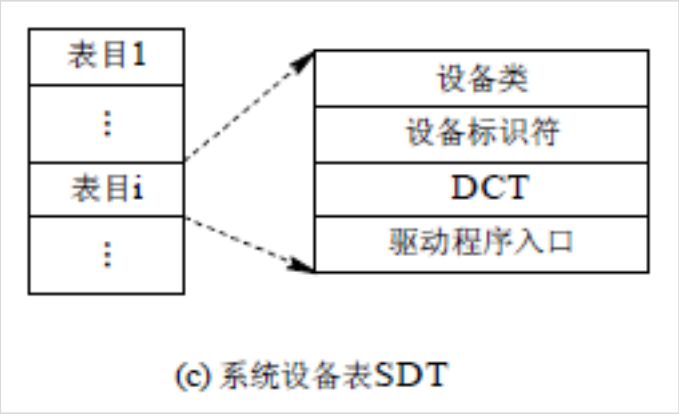




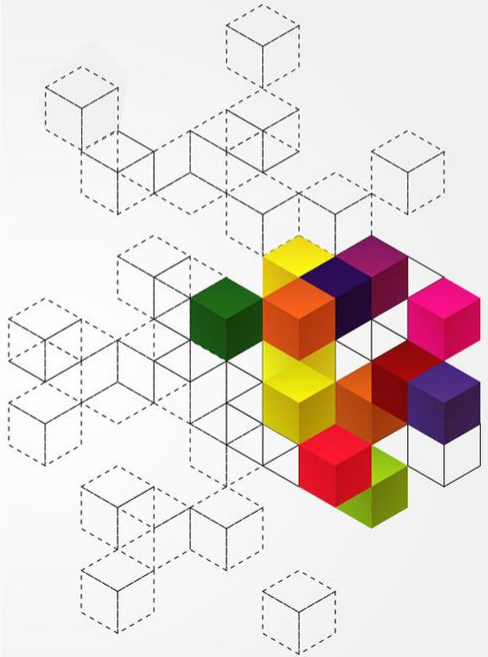
三、设备无关IO软件层

IO设备管理表。

DCT
SDT
COCT
CHCT



(c) 系统设备表SDT



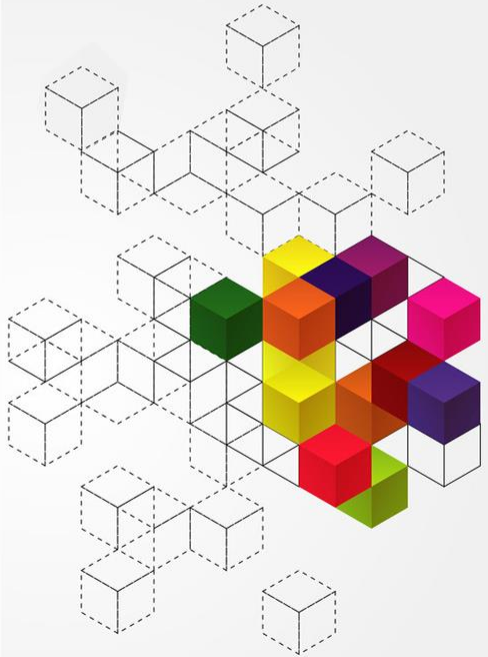
三、设备无关IO软件层

IO设备管理表。

DCT
SDT
COCT
CHCT

控制器标识符：controllerid
控制器状态：忙/闲
与控制器连接的通道表指针
控制器队列的队首指针
控制器队列的队尾指针

(a) 控制器表COCT



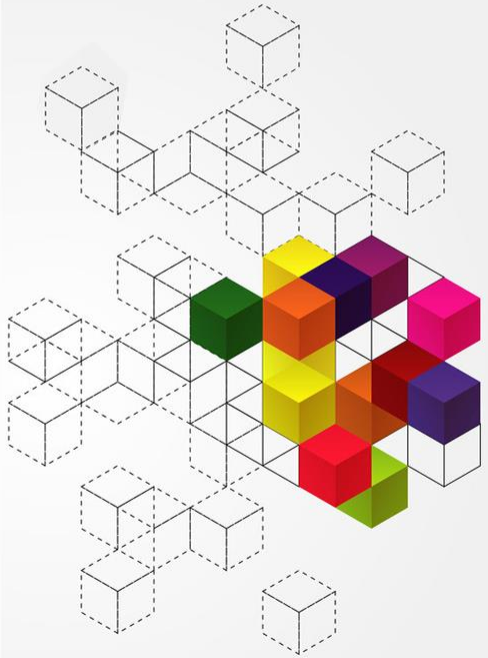
三、设备无关IO软件层

IO设备管理表。

DCT
SDT
COCT
CHCT

通道标识符：channelid
通道状态：忙/闲
与通道连接的控制器表首址
通道队列的队首指针
通道队列的队尾指针

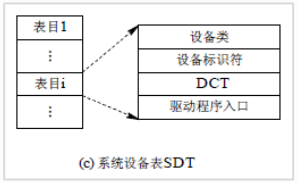
(b) 通道表CHCT



三、设备无关IO软件层

设备分配过程

1. 根据逻辑设备名查找SDT，找出设备的DCT，分配设备
2. 根据DCT找出COCT，分配设备控制器
3. 根据COCT找出CHCT，分配通道



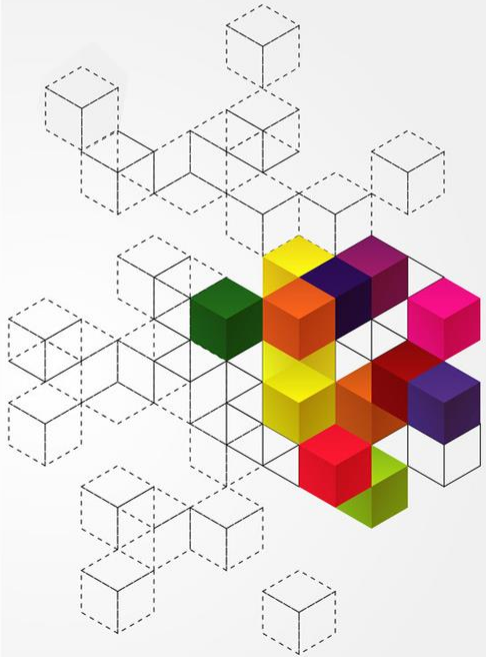
(c) 系统设备表SDT

控制器标识符：controllerid
控制器状态：忙/闲
与控制器连接的通道表指针
控制器队列的队首指针
控制器队列的队尾指针

(a) 控制器表COCT

通道标识符：channelid
通道状态：忙/闲
与通道连接的控制器表首址
通道队列的队首指针
通道队列的队尾指针

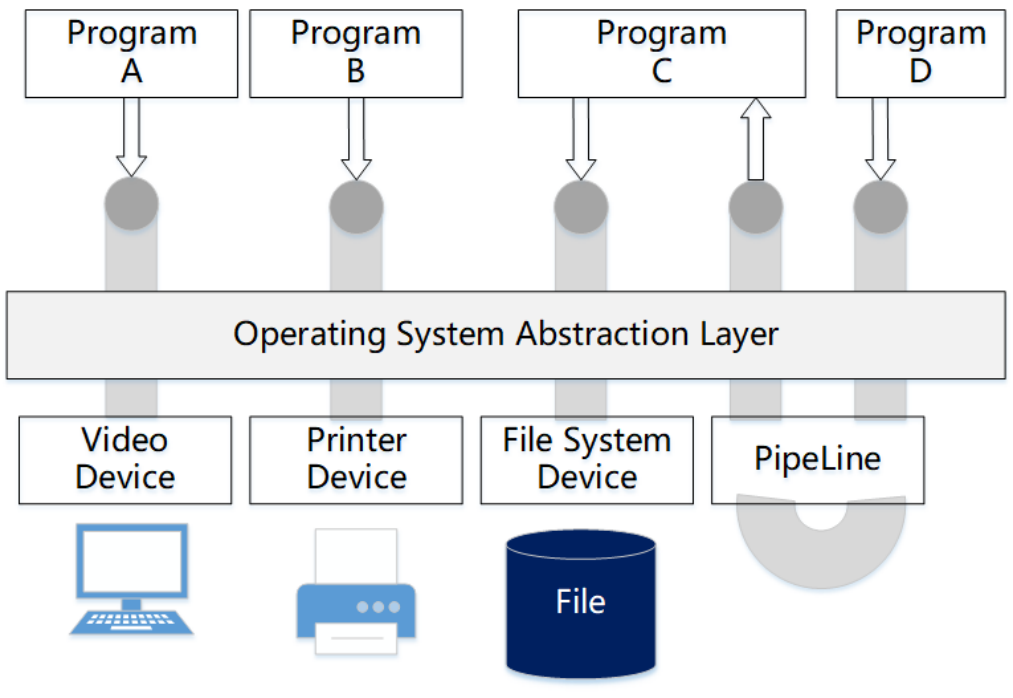
(b) 通道表CHCT

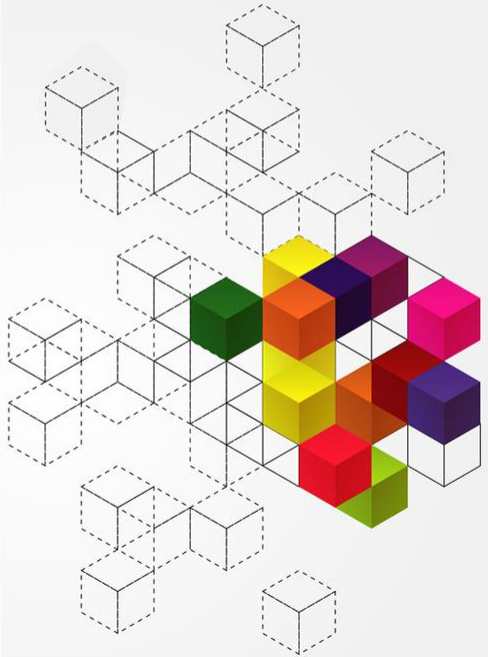


设备独立性：应用程序访问设备时，使用设备的逻辑名称，而不是物理设备名称。

三、设备无关IO软件层

通过硬件抽线层（HAL）达到IO独立的目的
提供众多通用IO管理功能



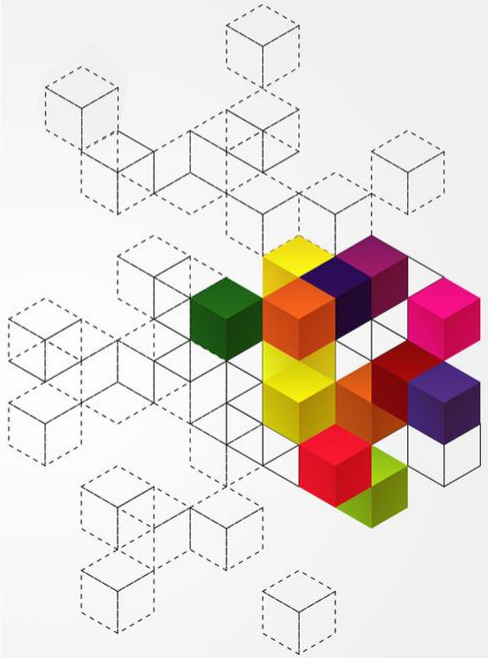


三、设备无关IO软件层

SPOOLING: 使独占设备可共享

Device Reservation: 将设备变为某进程独占访问

system calls for acquiring or releasing exclusive access to a device (care required)



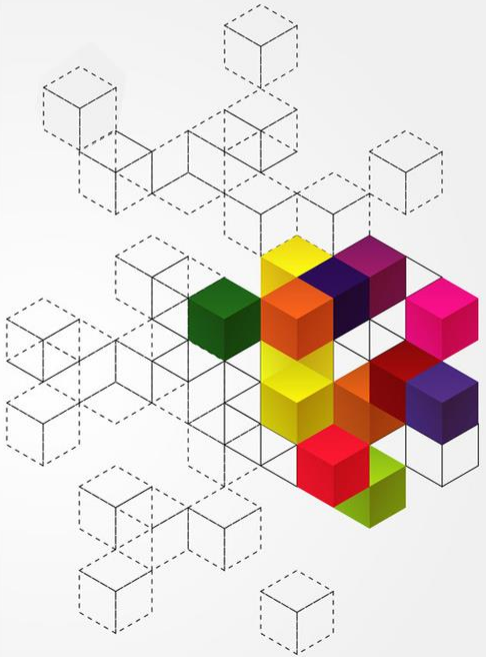
三、设备无关IO软件层

Linux Device Tree

官方描述：

The primary purpose of Device Tree in Linux is to provide a way to describe non-discoverable hardware. This information was previously hard coded in source code.

ARM设备树出现之前的电路板硬件的细节被硬编码到内核中了，导致内核代码臃肿难以维护，因此通过设备树将内核与那些臃肿的硬件代码解耦，方便维护。



三、设备无关IO软件层

Linux Device Tree

Linus 2011年的一封信

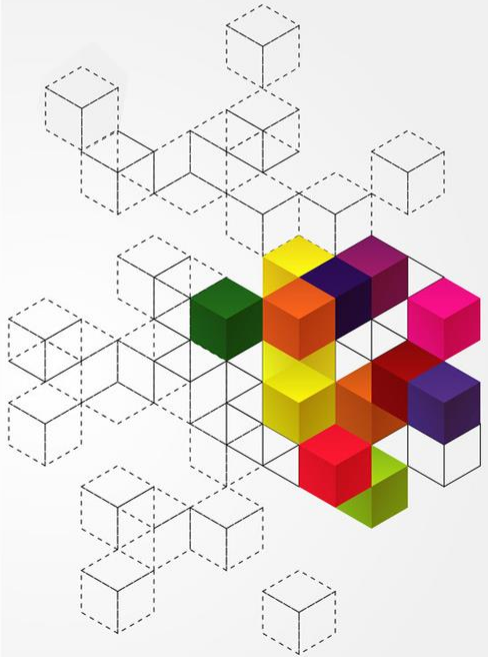
From Linus Torvalds <>
Date Thu, 17 Mar 2011 19:50:36 -0700
Subject Re: [GIT PULL] omap changes for v2.6.39 merge window

On Thu, Mar 17, 2011 at 11:30 AM, Tony Lindgren <tony@atomide.com> wrote:
>
> Please pull omap changes for this merge window from:

Gaah. Guys, this whole ARM thing is a f*cking pain in the ass.

You need to stop stepping on each others toes. There is no way that your changes to those crazy clock-data files should constantly result in those annoying conflicts, just because different people in different ARM trees do some masturbatory renaming of some random device. Seriously.

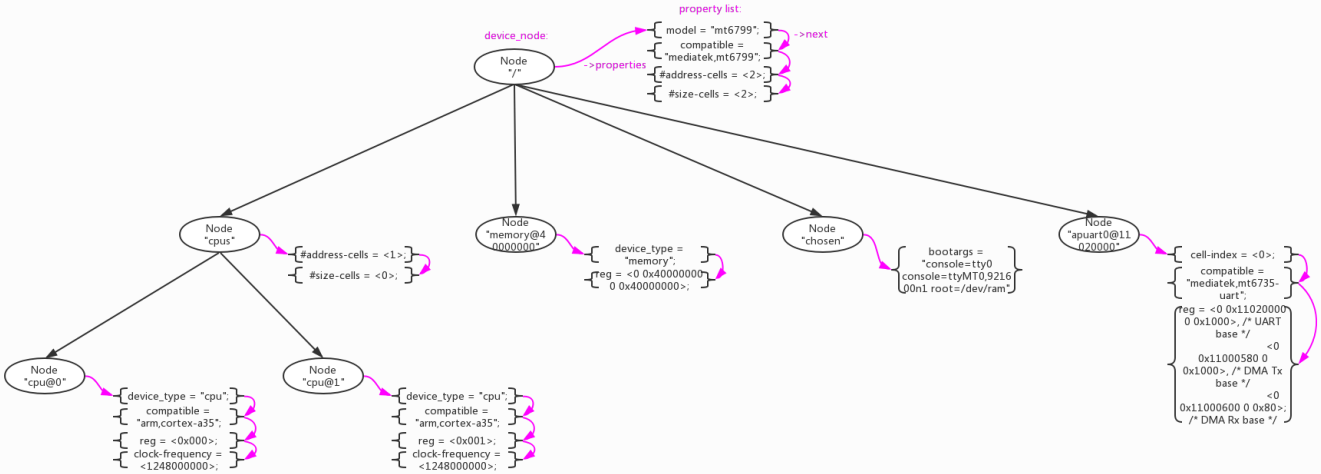
That usb_musb_init() thing in arch/arm/mach-omap2/usb-musb.c also seems to be totally insane. I wonder what kind of insanity I'm missing just because I don't happen to see the merge conflicts, just because people were lucky enough to happen to not touch the same file within a few lines.

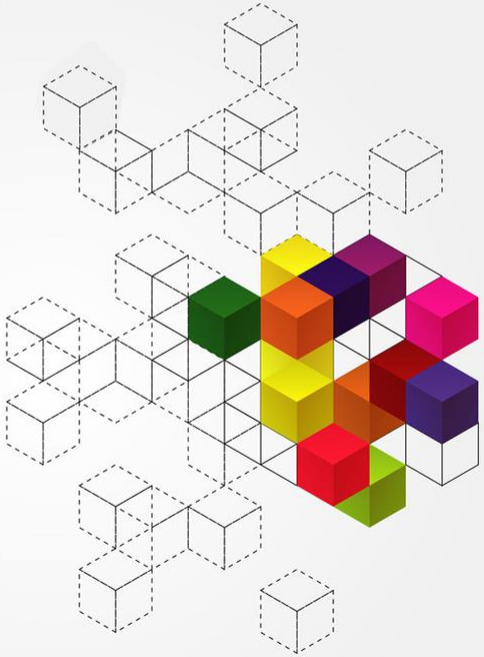


三、设备无关IO软件层

Linux Device Tree

设备树示意图

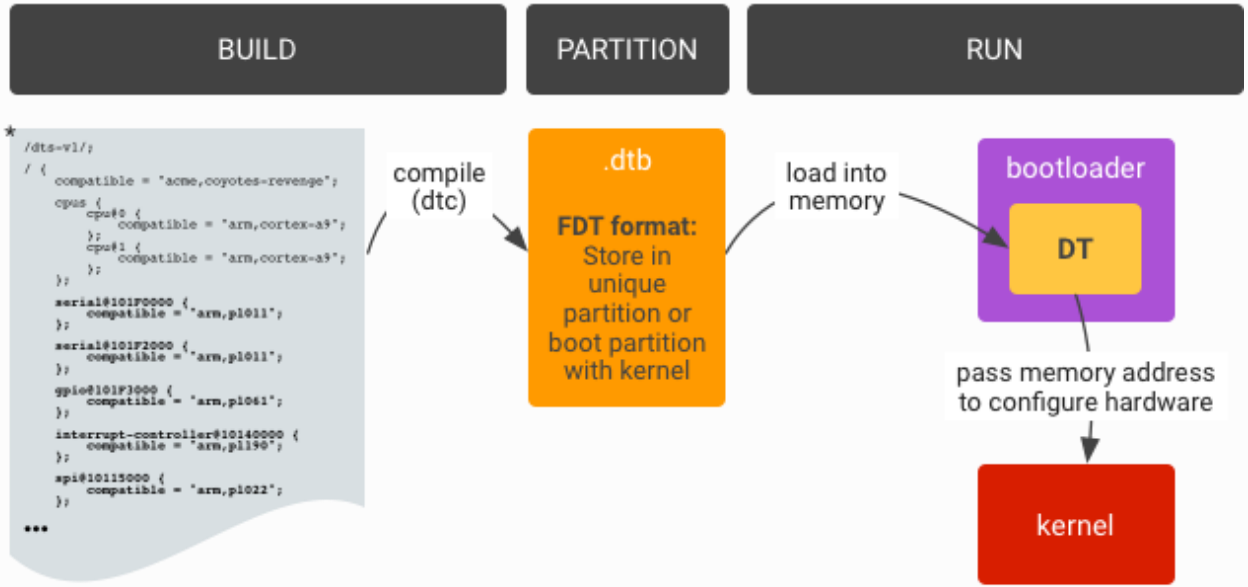




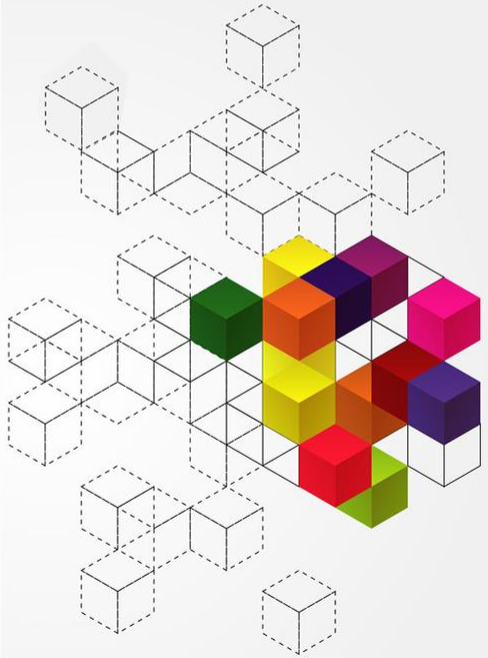
三、设备无关IO软件层

Linux Device Tree

Convert DTS to DTB with DTC



*http://elinux.org/Device_Tree_Usage#Devices



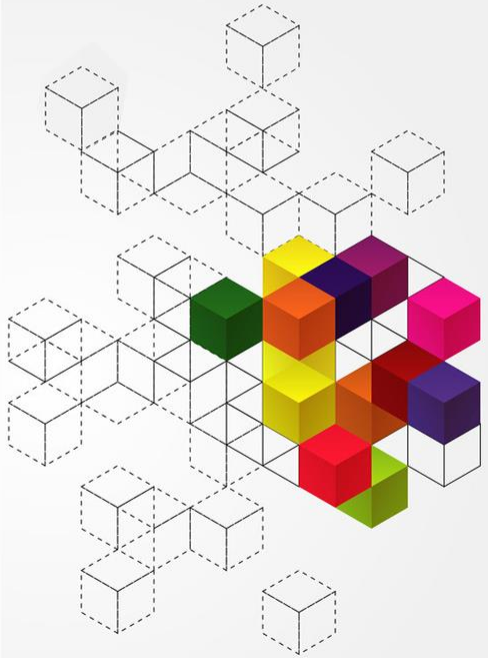
四、用户层IO软件模块

用户态的IO操作库

I/O子系统中绝不允许用户进程直接去执行特权态的I/O操作

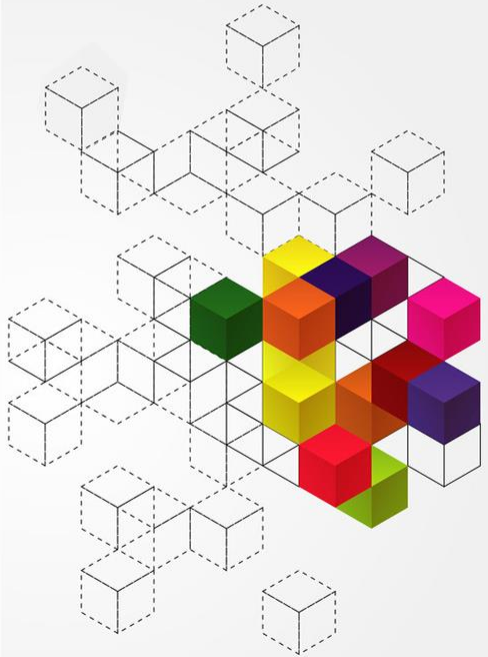
应用程序要实施I/O操作时，必须通过系统调用形式请求内核服务，在内核I/O相关系统调用代码内实现对I/O设备的操作

为了方便用户使用，对I/O相关系统调用进行功能封装后，以用户态库函数的形式供应用程序调用。



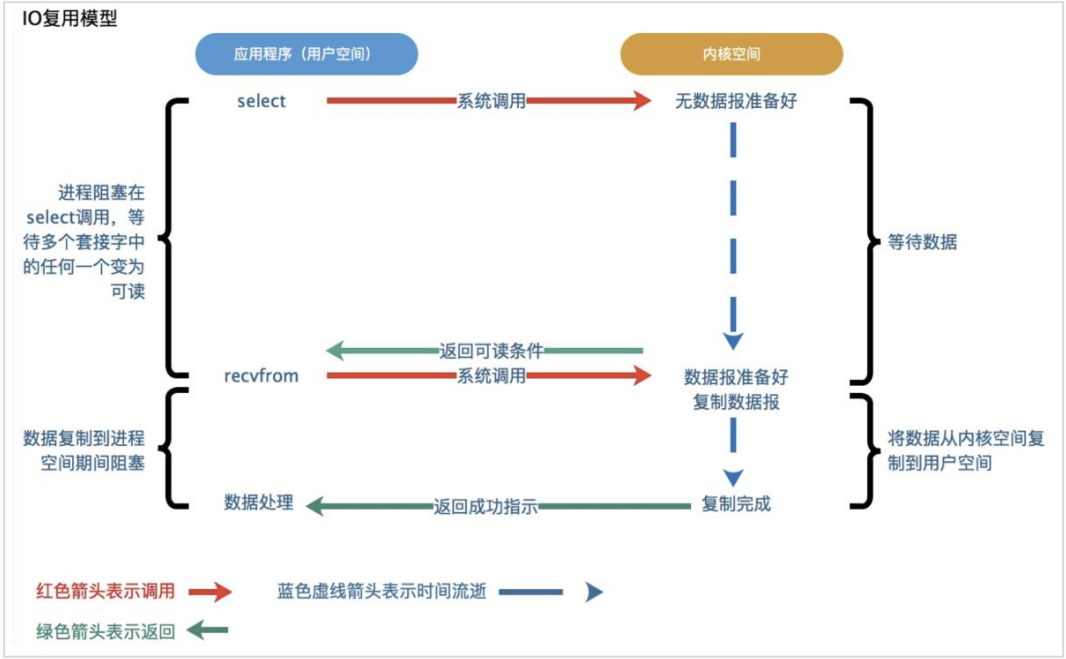
四、用户层IO软件模块

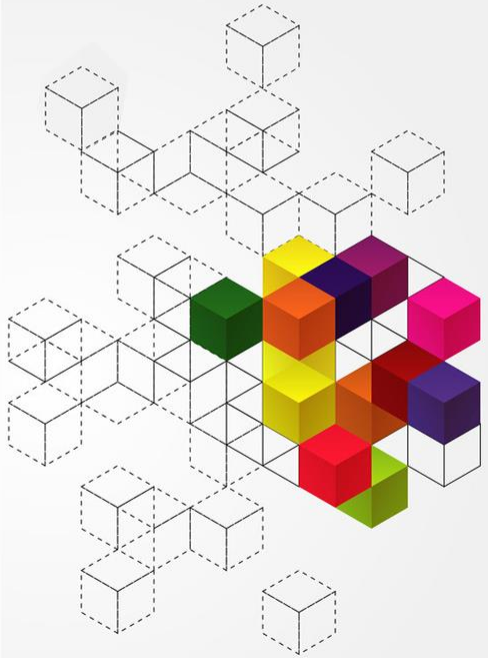
Linux IO模型



四、用户层IO软件模块

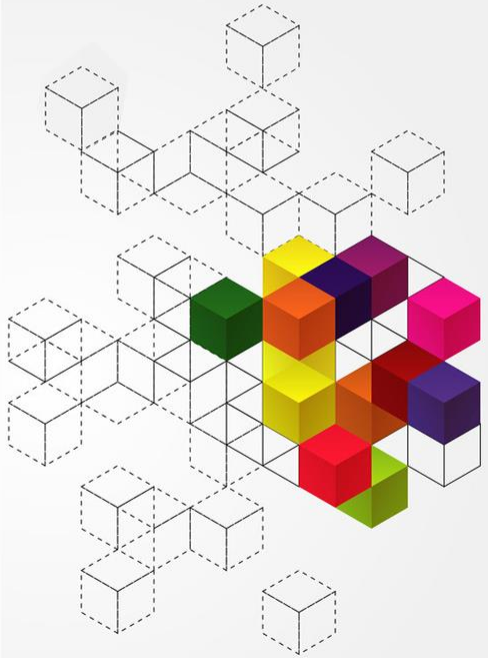
Linux IO接口示例：多路复用IO接口 select





本讲小结

- IO软件设计层次
- IO中断处理与设备驱动
- 设备无关IO软件层
- 用户层IO软件模块



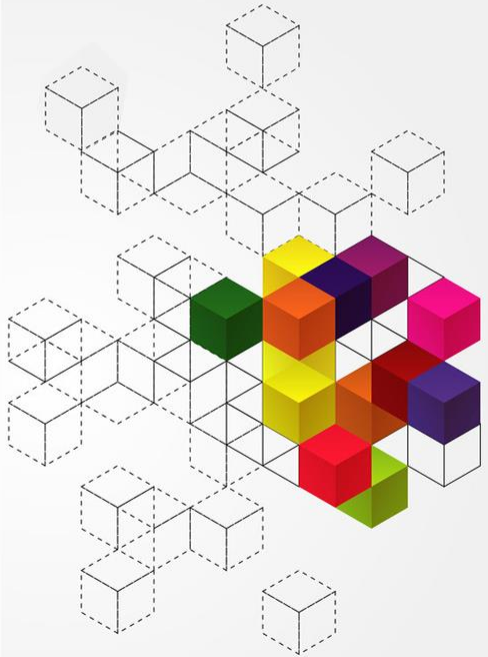
内容纲要

13.6 SPOOLING技术

一、SPOOLING技术背景

二、SPOOLING原理与概念

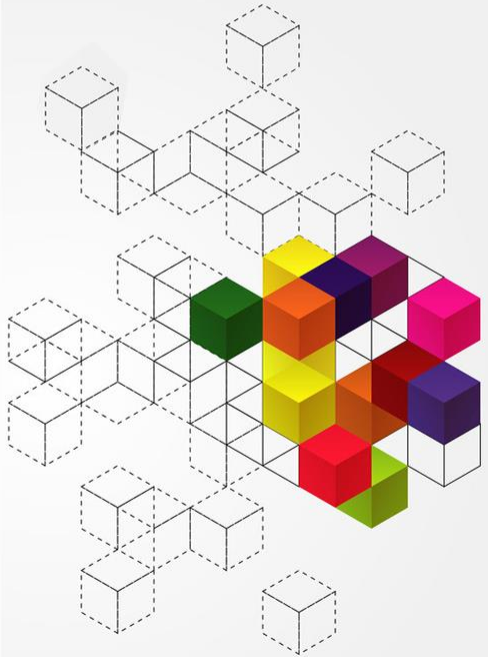
三、SPOOLING示例



一、SPOOLING技术背景

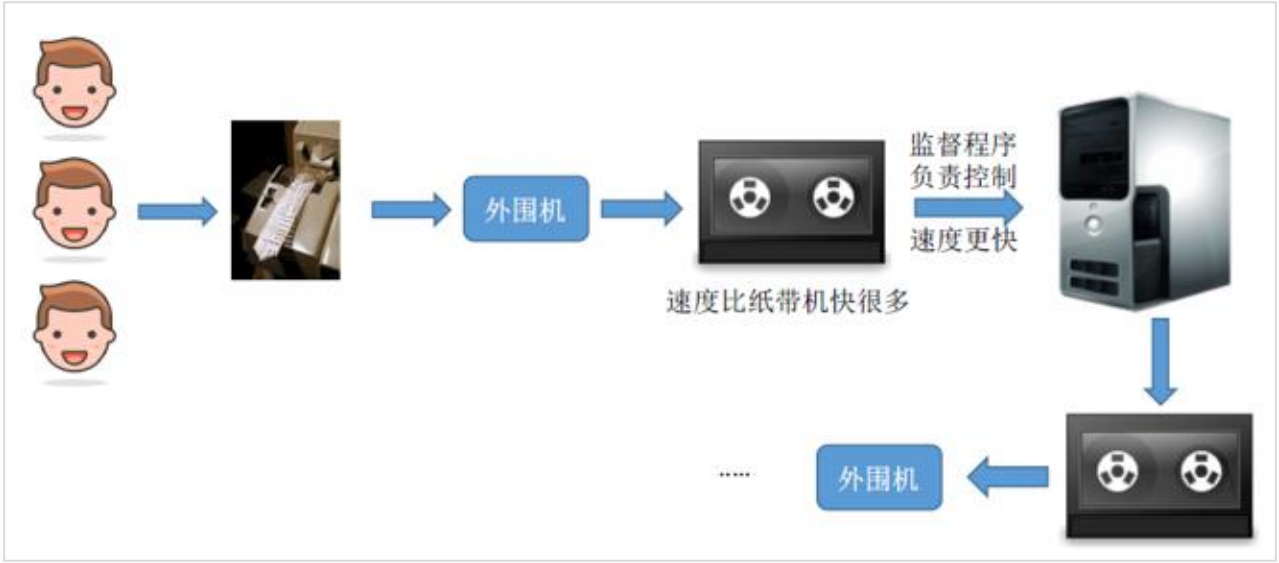
没有出现操作系统之前，
I/O依赖手工操作

IO速度非常慢，主机浪费很多时间等待
=>效率低

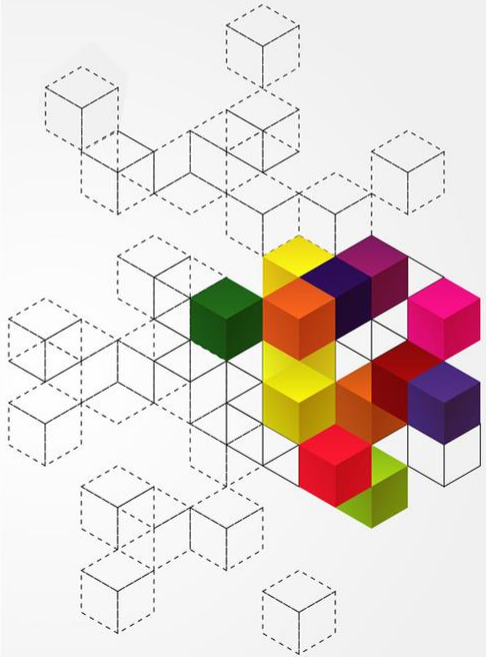


一、SPOOLING技术背景

脱机技术



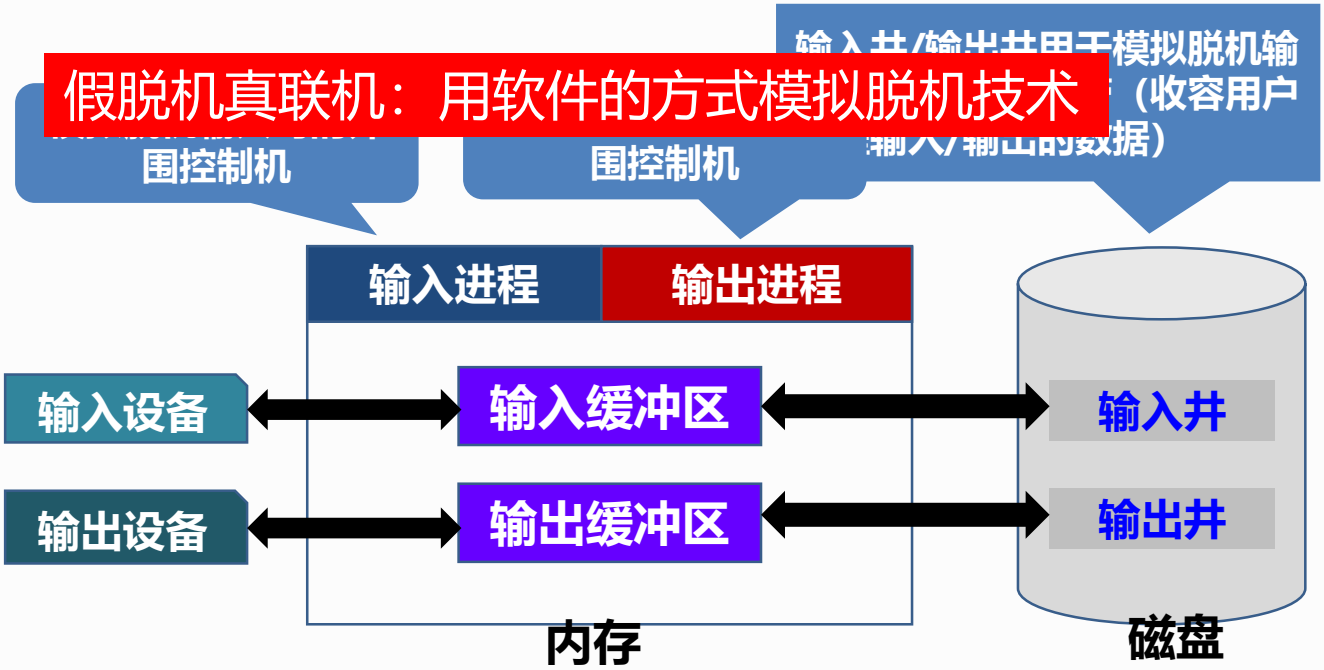
通过外围机将数据预先传送到较快速的磁带，
再由主机上专门的监督程序从磁带传入主机磁盘
=>**缓解IO设备与CPU速度不匹配的矛盾**

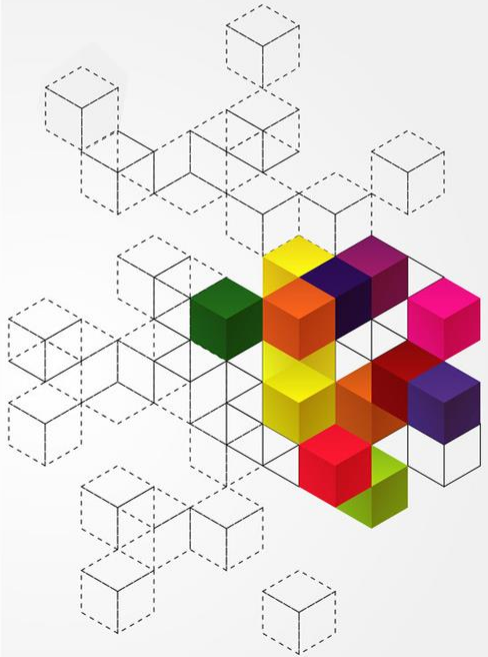


二、SPOOLING原理与概念

SPOOLING:假脱机技术

-在CPU速度极大提升、磁盘普及价格降低的情况下，以软件模拟替代专用外围机作用



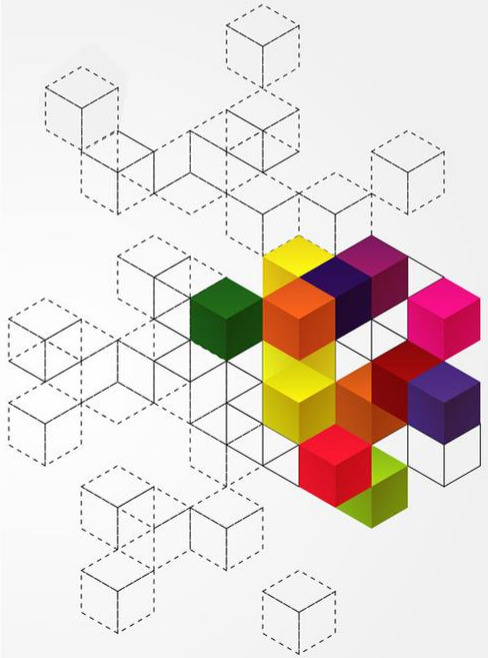


二、SPOOLING原理与概念

SPOOLING （外部设备联机并行操作）

-Simultaneous Peripheral Operations On-line

Spooling is an acronym for simultaneous peripheral operations on line. Spooling refers to putting data of various I/O jobs in a buffer. This buffer is a special area in memory or hard disk which is accessible to I/O devices.

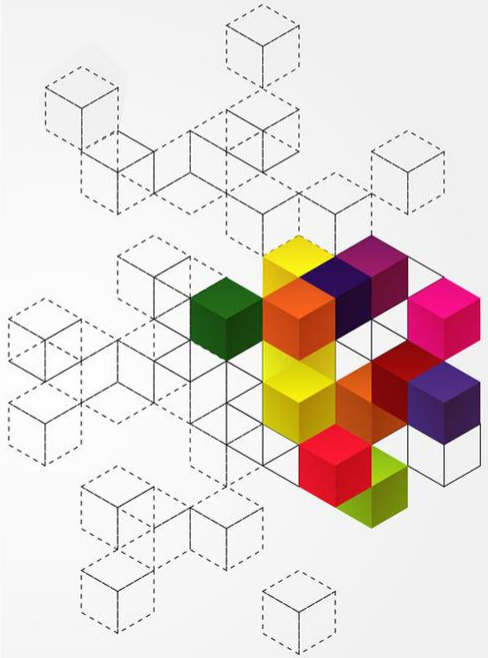


二、SPOOLING原理与概念

SPOOLING特点

缓解CPU与IO速度不匹配的矛盾

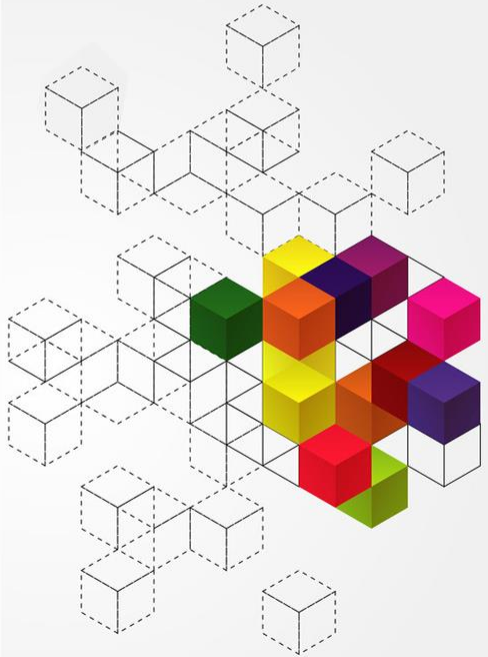
通过设立输入井和输出井作为缓冲，从对低速I/O设备进行的I/O操作变为对输入井或输出井的操作，使得CPU与I/O设备可以异步并发模式工作，解放了CPU



二、SPOOLING原理与概念

SPOOLING特点

- 缓解CPU与IO速度不匹配的矛盾
- 将独占设备改造为共享设备



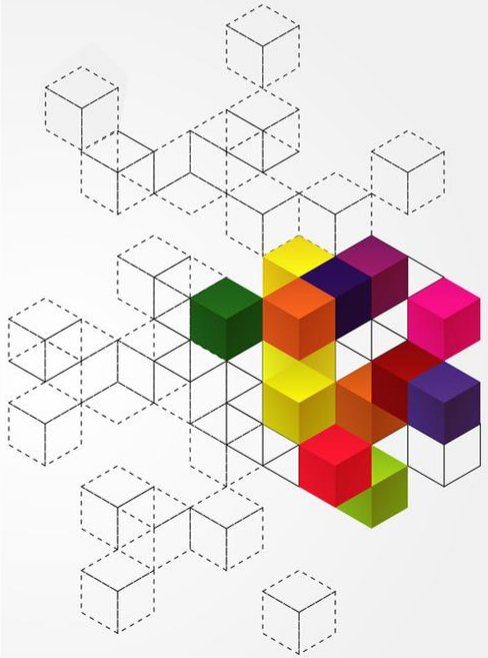
在SPOOLing系统的系统中，实际上并没为任何进程分配设备，而知识在输入井或输出井中为进程分配一个存储区和建立一张I/O请求表。

二、SPOOLING原理与概念

SPOOLING特点

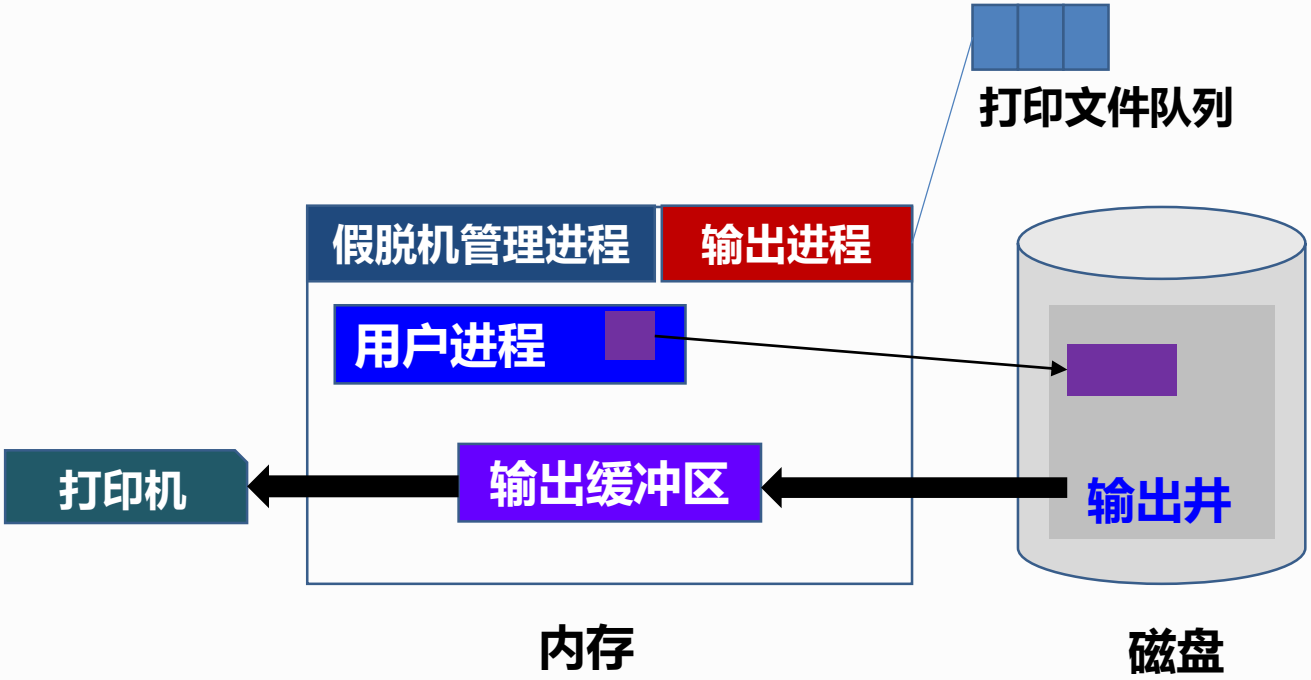
- 缓解CPU与IO速度不匹配的矛盾
- 将独占设备改造为共享设备
- 实现了虚拟设备功能

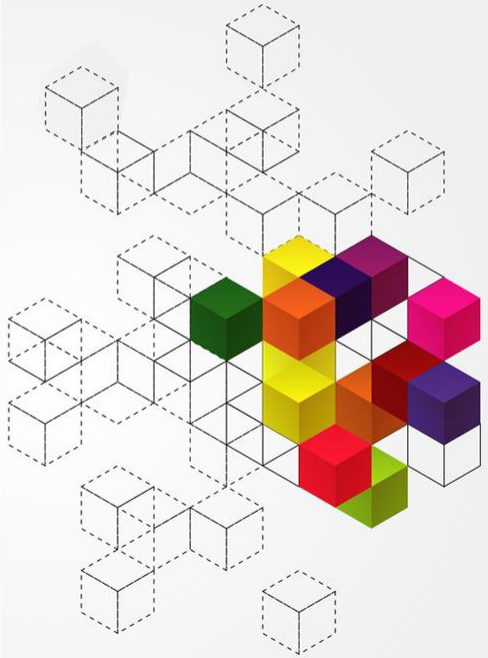
多个进程同时使用一独享设备，而对每一进程而言，都认为自己独占这一设备，从而实现了设备的虚拟分配



三、SPOOLING示例：共享打印机

打印机是典型的独占型设备，现代OS会通过SPOOLING技术将打印机模拟成共享设备。





本讲小结

- SPOOLING