華中科技大學
计算机科学与技术学院
School of Computer Science & Technology, HUST

操作系统原理

第二、三章 操作系统的物质基础、结构和用户接口

提纲

- 操作系统的物质基础
- 操作系统的结构
- 操作系统虚拟机
- 操作系统的生成和启动
- 程序的链接
- 操作系统的用户接口

操作系统的物质基础

- CPU特权级
- 中断机制
- 时钟
- DMA

为什么要有CPU特权级?

- 系统中两类程序的角色和区别

	管理程序	用户程序
处理器资源	调度和控制程序的执行	被调度、被控制
其他资源	管理系统资源	获得并使用系统资源

- 区分处理器状态的目的：保护操作系统

CPU特权级的区分

CPU状态的分类

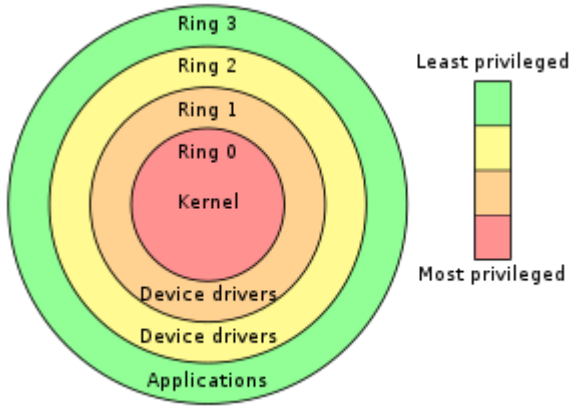
- 管态（supervisor mode，又称为“核态”）
  - 操作系统的管理程序执行时机器所处的状态，又称处理机的特权态。在此状态下处理机可使用**全部指令**（包括一组特权指令）、使用**全部系统资源**（包括整个存储区域）。
- 用户态（user mode）
  - 用户程序执行时机器所处的状态称为用户态。在此状态下**禁止使用特权指令，不能直接取用资源与改变机器状态**，并且**只允许用户程序访问自己的存储区域**。

CPU特权指令

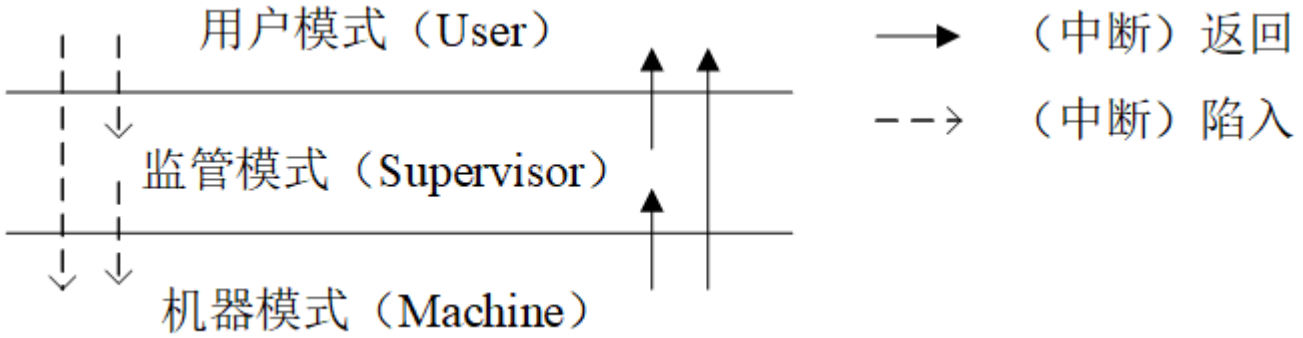
- I/O指令（如x86的in/out指令）
- MMIO地址空间访问指令
- 改变机器状态寄存器（MSR）的指令

CPU特权级（例子）

- Intel 80386特权级



- RISC-V CPU的特权级

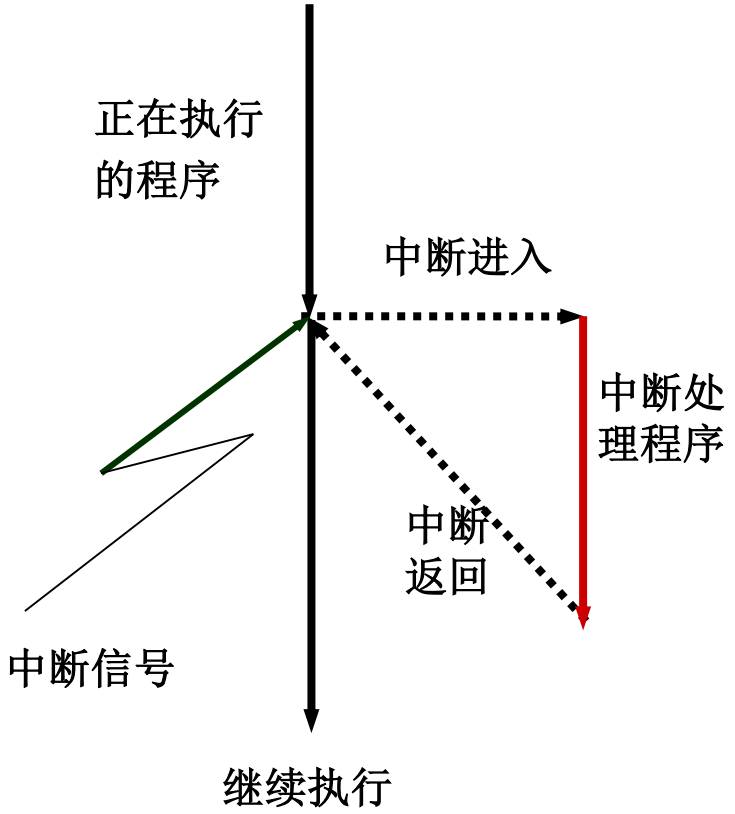


中断机制

- **为什么需要中断（interrupts）？**
  - 应用程序（用户态）的执行时常需要操作系统（内核态）的“帮助”，例如
    - 1）程序行为非法，需要OS干预
    - 2）调用操作系统功能（以完成I/O）
    - 3）接收和响应外部到来的I/O事件
  - 中断的本质是受保护的状态转换（protected control transfers），转换的过程中不允许存在让用户程序干预的可能。

中断响应的一般过程

- **中断进入（硬件实现）**
  - 改变机器状态，（从低优先级）进入高优先级
  - 注：硬件实现，不允许软件介入
- **中断处理（操作系统内核的软件实现）**
  - **保护**应用程序的**执行现场**到特定内存区域
  - 在高优先级实现中断的处理
  - 调用特定中断返回指令
  - 注：执行现场指的是CPU的通用寄存器中的内容
- **中断返回（硬件实现）**
  - 从特定内存区域**恢复执行现场**
  - 回到中断发生前的优先级
  - 注：硬件实现，不允许软件介入



中断的分类

中断（interrupts）的分类

- 异常（**exception**）
- 系统调用（syscall。有时被称为软件中断或者trap。建议用**syscall**）
- 中断请求（Interrupt ReQeust，IRQ。IRQ是Intel术语体系中的对应概念，RISC-V术语中称为interrupt。建议用IRQ。同时，这类中断也被称为“外部中断”）

异常与中断请求的区别

- An **exception** is a protected control transfer caused **synchronously** by the **currently running code**, for example due to a divide by zero or an invalid memory access.
- An **interrupt** is a protected control transfer that is caused by an **asynchronous** event usually **external to the processor**, such as notification of external device I/O activity.

中断的分类

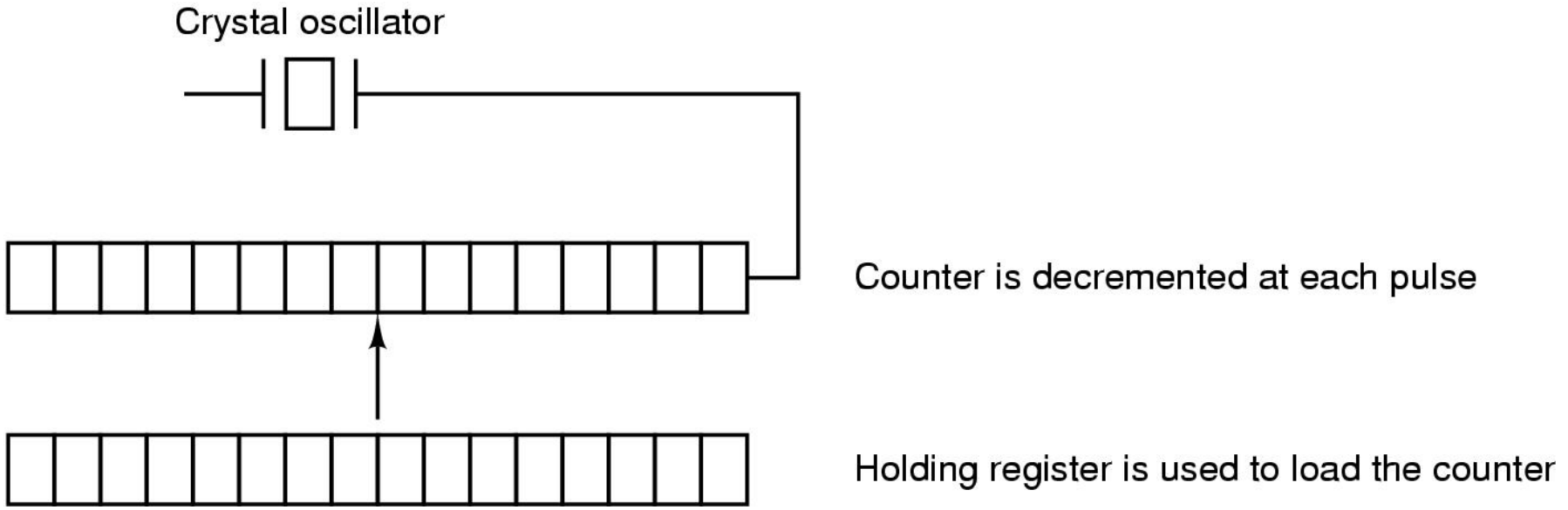
	Exception	Syscall	IRQ
产生原因	当前程序的执行所导致的同步事件	当前执行的程序需要调用OS功能	CPU以外的外部设备产生的异步事件
处理时机	发生异常的指令执行过程中	中断指令执行完成	指令执行的间隙
返回地址	发生异常的那条指令	下一条指令	下一条指令
举例	除零错、非法内存访问	x86机器中的int指令、RISC-V机器中的ecall指令	敲击键盘、磁盘数据传输完成等

时钟

三种常见时钟硬件

- 可编程间隔定时器（Programmable Interval Timer，PIT）
- 实时时钟（Real Time Clock，RTC）
- 时间戳计数器（Time Stamp Counter，TSC）

可编程间隔定时器的硬件原理



A programmable interval clock

可编程间隔定时器（PIT）

- Programmable interval clock的工作模式
  - One-shot mode
  - Square-wave mode
- 时钟滴答(Clock tick)
- 周期性地发生时钟中断（可编程设置间隔）
- 今天的高精度可编程间隔定时器又称为HPET（High Precision Event Timer）

RTC和TSC

实时时钟RTC

- 在PC机断电后仍能保存时间
- 通过主板上的电池供电；通常与CMOS RAM集成到一块芯片上，也称为 CMOS Timer
- 可在系统初启时读入并转换为相对于某一基准时间的时钟滴答数

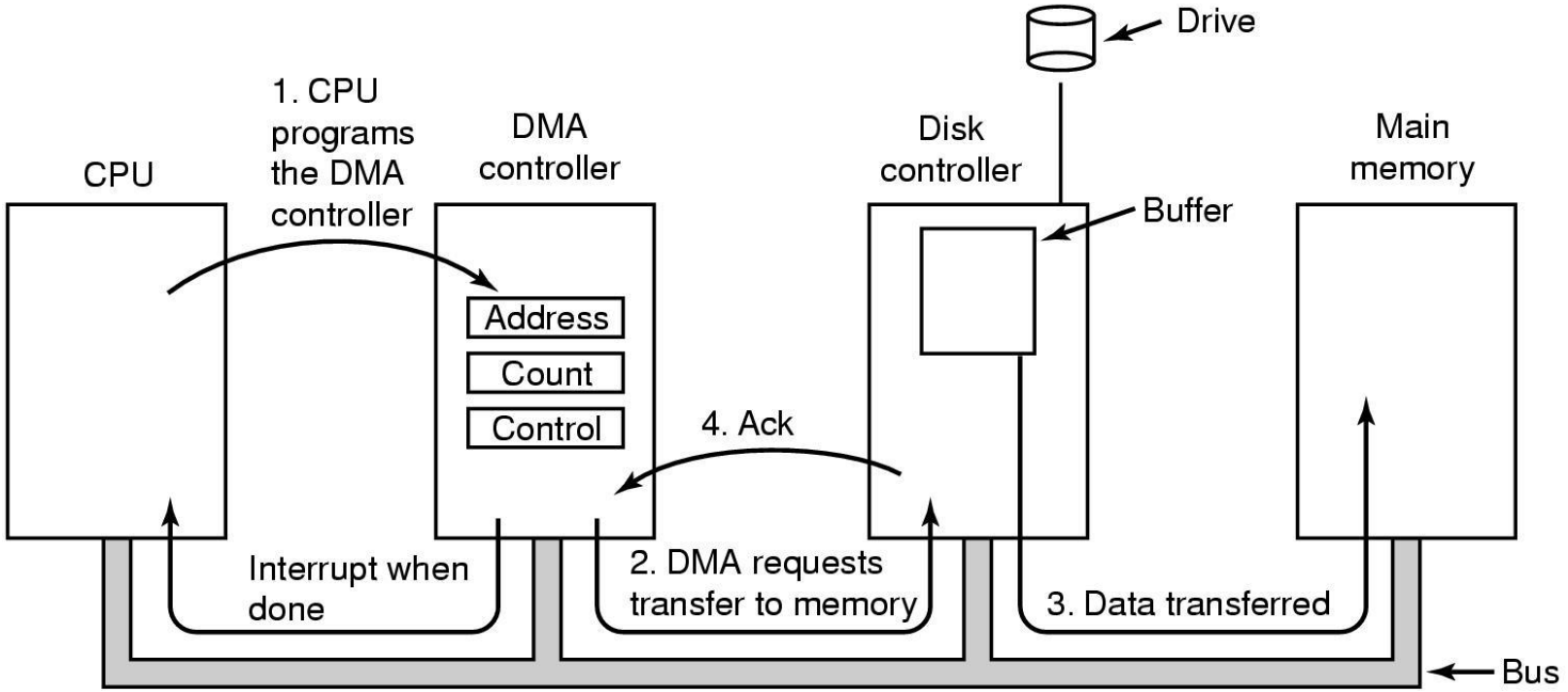
时间戳记数器TSC

- 某些处理器（如Intel的Pentium）包含的64位寄存器
- 在每一个振荡信号到达时，该计数器递增
- 可为操作系统提供更准确的时间度量

DMA

特点

- 传输过程无需CPU参与控制
- 需要中断支持
- 用于DMA的内存地址必须连续
- 传输过程占用总线资源



提纲

- 操作系统的物质基础
- 操作系统的结构
- 操作系统虚拟机
- 操作系统的生成和启动
- 程序的链接
- 操作系统的用户接口

操作系统的组件

- **计算机的核心组件**
  - CPU
  - 内存
- **计算机的外围组件**
  - （大容量）磁盘
  - 其他外设

- 操作系统的核心组件
  - 中断管理
  - 进程管理
  - 进程调度
  - 内存管理
- 计算机的外围组件
  - 设备管理
  - 文件系统

操作系统的结构分类

- **单一大内核（宏内核）结构**
  - 将所有操作系统组件全部放在内核中，全部运行在内核态
  - 优点：系统结构简单；缺点：内核非常庞大，容易崩溃
  - 例子：Linux
- **微内核结构**
  - 只将操作系统核心组件放在内核中，将外围组件放在用户态运行
  - 优点：OS内核非常小，容易保证稳定；缺点：采用IPC通讯，效率偏低
  - 例子：MACH、GNU HURD、QNX、鸿蒙
- **伴生内核结构**
  - 在同一台机器上运行多个OS，其中一个是主OS，其他为伴生OS
  - 伴生OS结构相对简单，但完整系统结构复杂
  - 例子：PKE（我们的OS实验）、云端虚拟机、Docker容器

提纲

- 操作系统的物质基础
- 操作系统的结构
- 操作系统虚拟机
- 操作系统的生成和启动
- 程序的链接
- 操作系统的用户接口

虚拟机的概念

- 虚拟机的基本思想是将单个计算机的硬件抽象为几个不同的执行部件。
- 创建虚拟机的根本原因是在并行运行几个不同的执行环境时能够共享相同的硬件。
- 通过利用CPU调度和虚拟内存技术，操作系统能创建一种幻觉，以至于进程认为有自己的处理器和自己的（虚拟）内存。

"All problems in computer science can be solved by another level of indirection" —— David Wheeler
See also: https://en.wikipedia.org/wiki/Indirection

虚拟机的分类

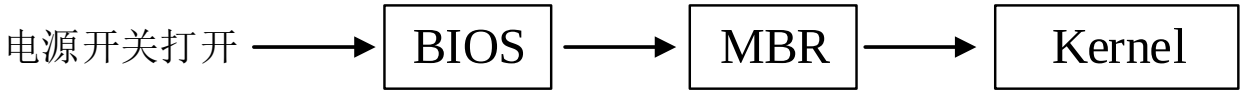
- **指令集（ISA）虚拟机**
  - 在给定ISA环境（如x86）下模拟采用另一套ISA的机器（如RISC-V）
  - 例如：spike、qemu
- **操作系统（OS）虚拟机**
  - 在A操作系统环境下创建B操作系统的执行环境
  - 例如：vmware、Xen、Docker
- **库函数（API）虚拟机**
  - 模拟操作系统的各类用户态API（通常是运行库的函数接口）
  - 例如：Wine
- **语言（language）虚拟机**
  - 虚拟语言的运行时环境（即runtime），通过解释或即时编译技术（Just-In-Time，JIT）来运行语言虚拟机指令，从而实现软件的跨平台特性
  - 例如：JAVA虚拟机

提纲

- 操作系统的物质基础
- 操作系统的结构
- 操作系统虚拟机
- 操作系统的生成和启动
- 程序的链接
- 操作系统的用户接口

操作系统的启动

x86机器上Linux的启动流程



1. （可选）内存自检
2. PCIe设备枚举，存储结果到MART、DART
3. 处理器进入实模式
4. 加载启动磁盘的MBR到0x7c00
4. jmp到0x7c00，将控制权交给MRB

1. 读入（并展开）Linux 内核到内存（到1MB以上的空间）
2. jmp到内核的入口地址，将控制权交给Linux内核

1. 进入保护模式，设置保护模式下的分段，扩大实际能够访问的内存空间
2. 设置中断处理、时钟中断等机制
3. 为内核构造页表，打开分页机制
4. 创建进程池，以及相应的就绪队列等机制，开始内存管理

RV64G机器上PKE的启动流程

1. 将模拟硬件信息，填入DTS
2. 加载PKE内核到模拟内存
3. 跳转到PKE的入口地址执行

1. 通过DTS设置HTIF、获取内存信息
2. 设置应用执行环境
3. 加载应用，并将其投入执行

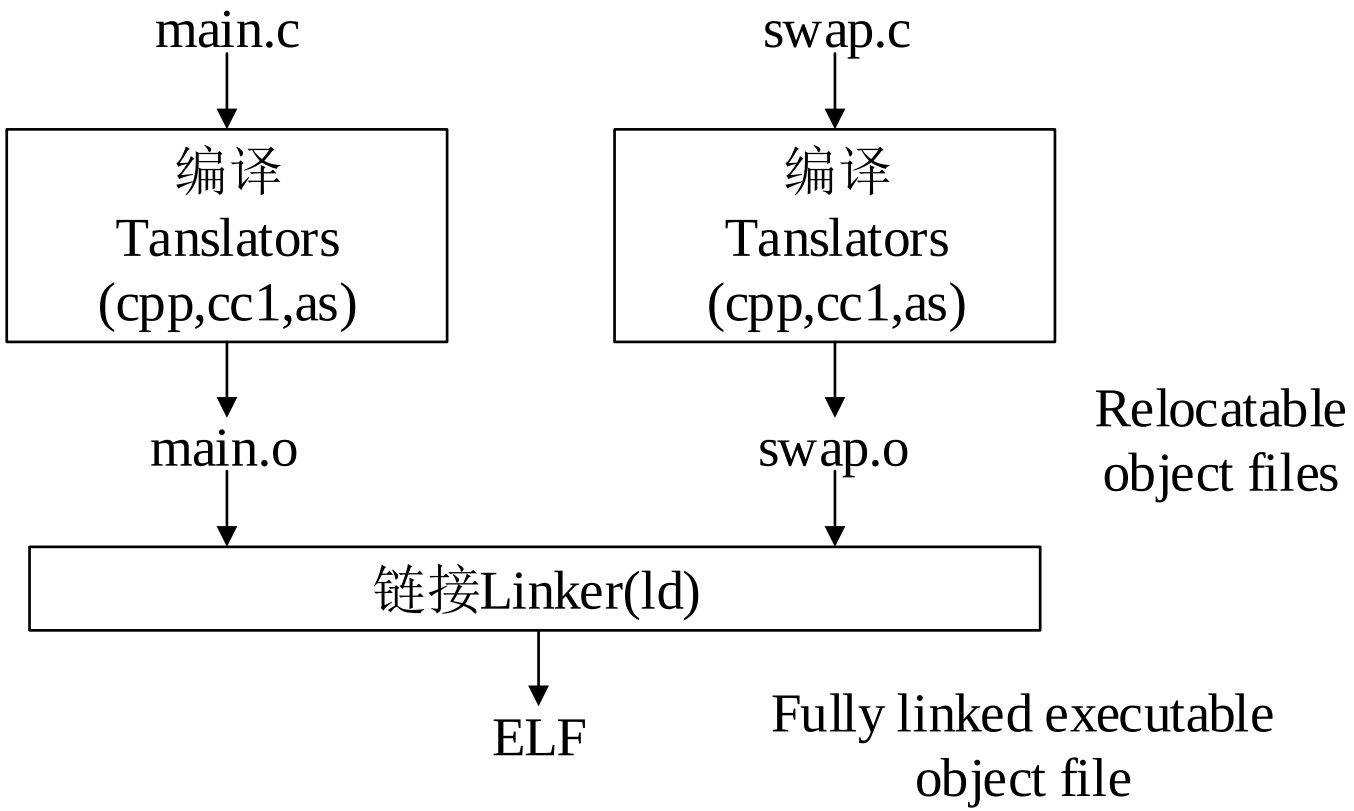
操作系统的生成

- 简单地说就是系统的安装，即将OS安装到启动装置（可以是磁盘，也可以是tftp服务器的某个目录）
- 设置启动环境，包括
  - Bootloader （如x86机器上的MBR）
  - OS内核
  - 文件系统
  - 执行环境（如用户态lib等）

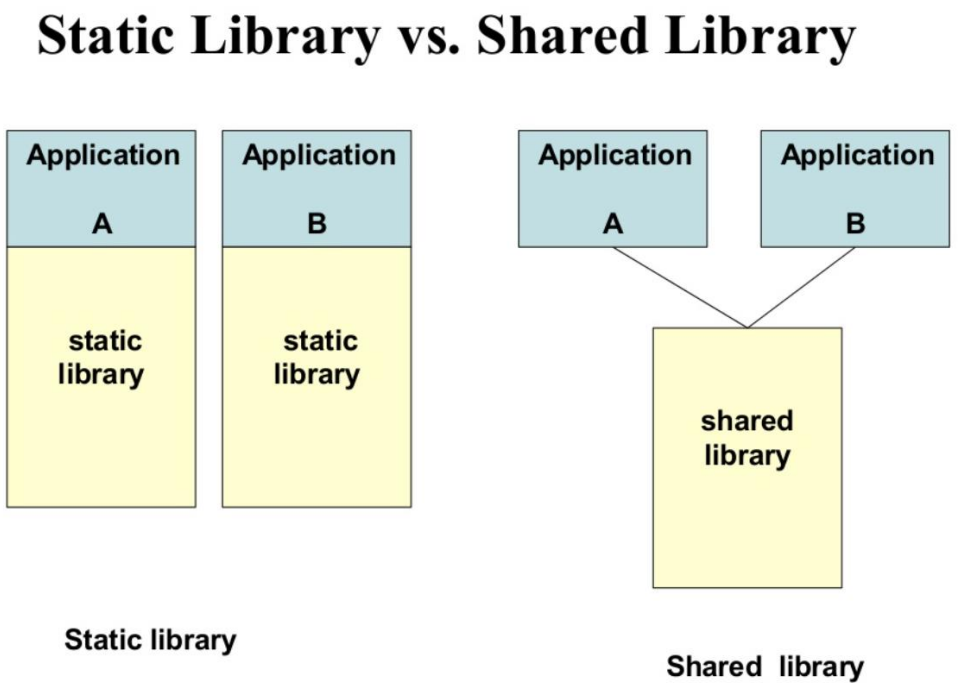
提纲

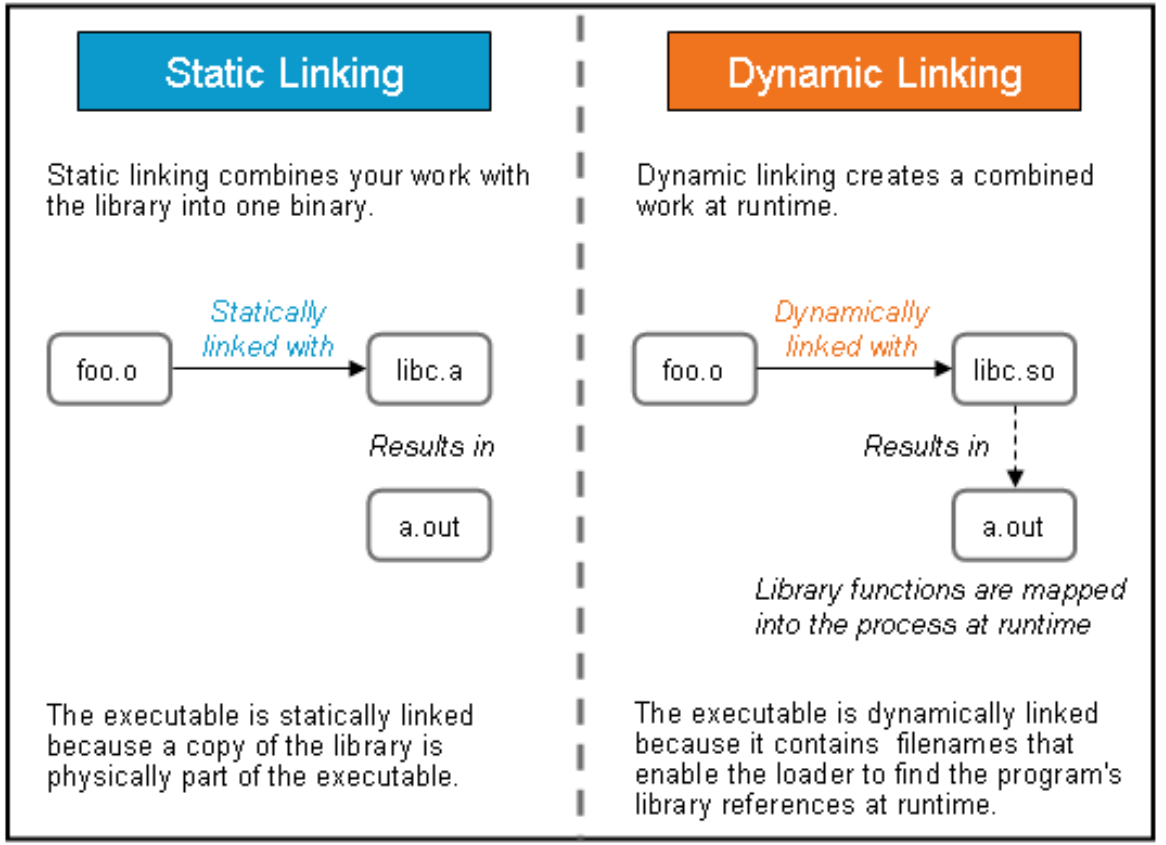
- 操作系统的物质基础
- 操作系统的结构
- 操作系统虚拟机
- 操作系统的生成和启动
- 程序的链接
- 操作系统的用户接口

程序的编译(compile)和链接(link)



程序的静态/动态链接


Static Library vs. Shared Library
Application A
Application B
static library
static library
Static library
Application A
Application B
shared library
Shared library


Static Linking
Static linking combines your work with the library into one binary.
foo.o
Statically linked with
libc.a
Results in
a.out
The executable is statically linked because a copy of the library is physically part of the executable.
Dynamic Linking
Dynamic linking creates a combined work at runtime.
foo.o
Dynamically linked with
libc.so
Results in
a.out
Library functions are mapped into the process at runtime
The executable is dynamically linked because it contains filenames that enable the loader to find the program's library references at runtime.

静态/动态链接

- **静态链接**
  - 静态连接将所需的外部函数链接到目标文件中形成为一个可执行文件。
  - 优点：可执行文件的执行不依赖函数库。
  - 缺点：若多个应用程序都调用了同一个库中的外部函数，那么，多个应用程序的目标文件中都会包含这个外部函数对应的代码。
- **动态链接**
  - 动态连接不需要将外部函数链接到目标文件中。而是在应用程序中需要调用外部函数的地方作记录，并说明要使用的外部函数名和引用入口号。
  - 优点：节约磁盘/内存空间，生成的可执行文件大小较小。
  - 缺点：可执行文件的执行需要执行环境找到并加载所有它依赖的函数库。

提纲

- 操作系统的物质基础
- 操作系统的结构
- 操作系统虚拟机
- 操作系统的生成和启动
- 程序的链接
- 操作系统的用户接口

操作系统的用户界面

- **操作系统的用户界面（或称接口）是操作系统提供给用户与计算机打交道的外部机制。用户能够借助这种机制和系统提供的手段来控制用户所在的系统。**

- **两类用户界面**
  - 操作界面 (命令接口)
    - 用户使用操作界面来组织工作流程和控制程序的运行。
    - 例如：键盘命令、图形用户界面。
  - 系统功能服务界面 (程序接口)
    - 用户程序在其运行过程中，使用系统功能调用来请求操作系统的服务。
    - 例如：系统调用（syscall）

小结

掌握

- 处理机的态，概念、设计目标、区别
- 中断的定义、分类，中断响应的一般过程
- 静态连接和动态链接的原理和区别
- 操作系统的用户接口，系统调用的概念及实现方法

理解

- 操作系统的物质基础，PIT、DMA等
- 操作系统的结构，各种结构的特点
- 操作系统虚拟机的概念，虚拟机的分类
- 系统的生成和启动