OPERATING SYSTEMS

Functional Views of OS

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

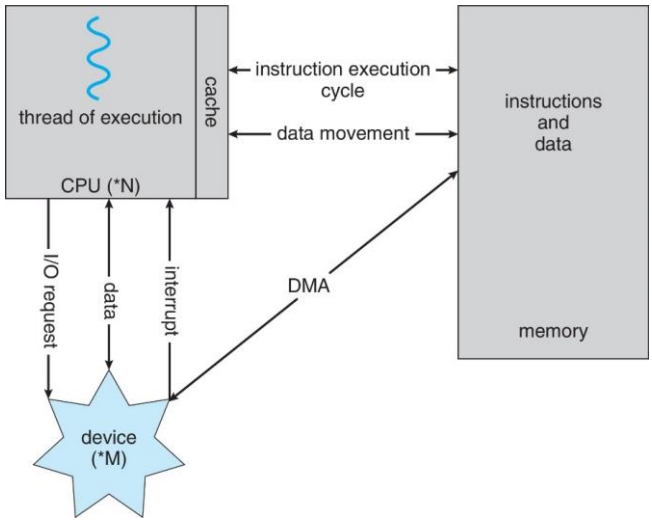
中山大學 SUN YAT-SEN UNIVERSITY

Functional Views of Operating Systems

- 目录
  - 计算机动力学
  - 硬件保护
    - 双模保护
    - I/O保护
    - 内存保护
    - CPU保护
  - 基本操作系统概念
    - 线程
    - 地址空间
    - 进程
    - 双模式操作
  - **资源管理简述（详见第04讲的通用操作系统组件）
    - 进程管理
    - 内存管理
    - 文件管理
    - 大容量存储管理
    - 缓存管理
    - I/O管理
  - 自由和开源操作系统

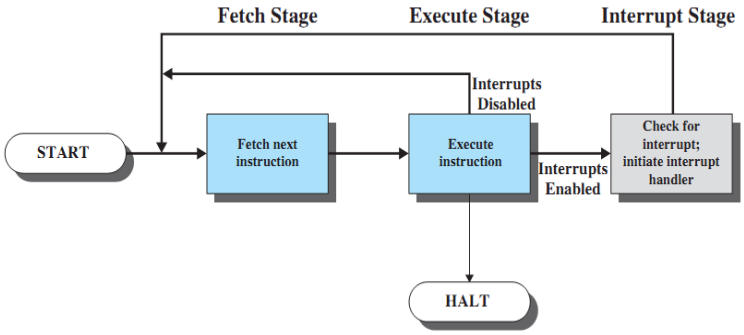
Computer Dynamics

- 计算机动力学
  - 现代计算机系统是如何工作的



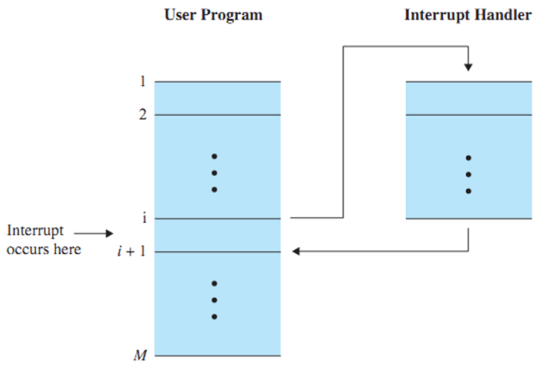
Computer Dynamics

- 计算机动力学
  - 带中断的指令周期
    - CPU在每条指令后检查中断
    - 如果没有中断，则获取当前程序的下一条指令
    - 如果中断挂起，则挂起当前程序的执行，并执行中断处理程序



Computer Dynamics

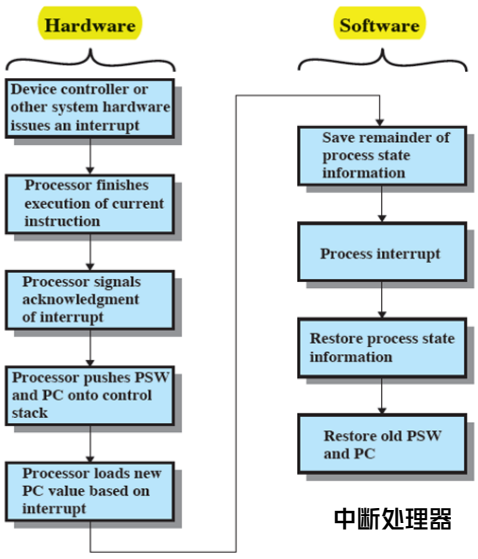
- 计算机动力学
  - 带中断的指令周期
    - CPU在每条指令后检查中断
    - 如果没有中断，则获取当前程序的下一条指令
    - 如果中断挂起，则挂起当前程序的执行，并执行中断处理程序



通过中断传输控制

Computer Dynamics

- 计算机动力学
  - 带中断的指令周期
    - 中断机制



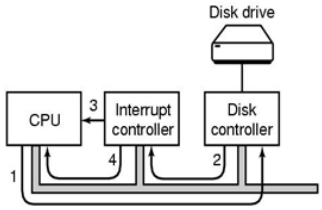
中断处理器

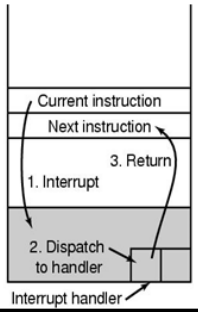
- 计算机动力学
  - 外部中断
    - 外部中断是由进程外部事件引起的进程暂停，中断执行后进程可以恢复
    - 导致外部中断的进程外部事件：
      - 输入输出
      - 定时器
      - 硬件故障

- 计算机动力学
  - 中断处理器
    - 中断处理程序是一个程序，它确定中断的性质并执行所需的任何操作
    - 当外部中断事件发生时，外部中断硬件通常通过包含所有中断服务例程（ISR）地址的中断向量将控制转移到中断处理程序
    - 中断体系结构必须保存程序的状态（PSW、PC、寄存器等的内容）
    - 在处理另一个中断时，将禁用传入中断，以防止中断丢失
    - 随后，控制必须转移回被中断的程序，使其可以从中断点恢复

Computer Dynamics

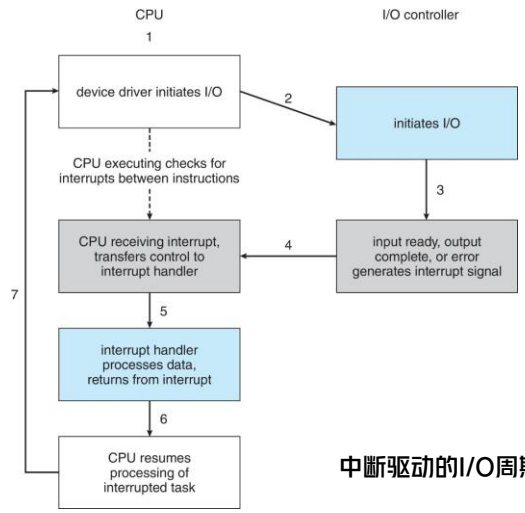
- 计算机动力学
  - 中断驱动I/O
    - I/O设备和CPU可以并行执行
    - 每个设备控制器负责特定的设备类型
    - 每个设备控制器都有一个本地缓冲区
    - CPU将数据在主存与本地缓冲区之间移动
    - I/O从设备到控制器的本地缓冲区，或反过来
    - 设备控制器通过引起外部中断来通知CPU它已完成其操作





Computer Dynamics

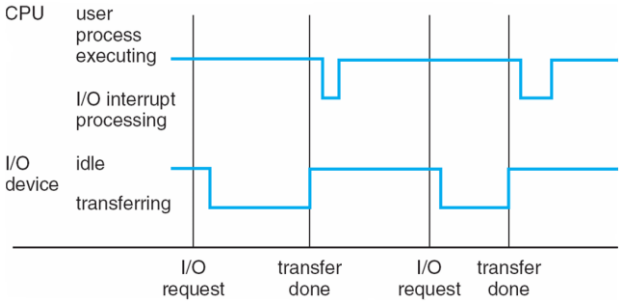
- 计算机动力学
  - 中断驱动I/O



中断驱动的I/O周期

Computer Dynamics

- 计算机动力学
  - CPU和I/O设备的中断时间线。

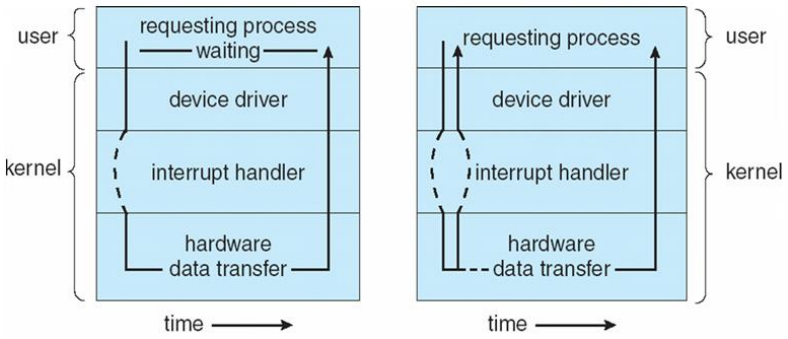


Computer Dynamics

- 计算机动力学
  - 同步和异步I/O方法
    - 同步I/O
      - I/O启动后，控件仅在I/O完成后返回到用户程序
      - Wait指令使CPU空闲，直到下一个中断
      - 等待循环（内存访问争用）
      - 一次最多有一个I/O请求未完成，没有同时的I/O处理
    - 异步I/O
      - I/O启动后，控制返回到用户程序，而无需等待I/O完成。
      - 系统调用对操作系统的请求，以允许用户等待I/O完成。
      - 设备状态表包含每个I/O设备的条目，指示其类型、地址和状态。
        - 操作系统索引到I/O设备状态表中，以确定设备状态并修改表项以包含中断。

Computer Dynamics

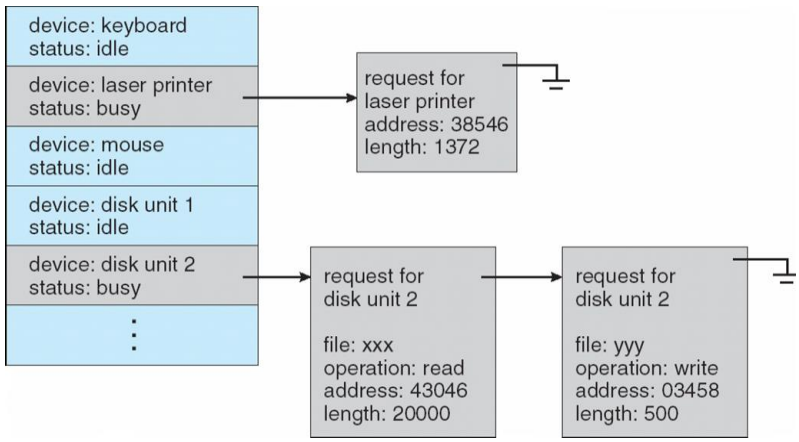
- 计算机动力学
  - 同步和异步I/O方法。



同步的　　异步的

Computer Dynamics

- 计算机动力学
  - 同步和异步I/O方法。



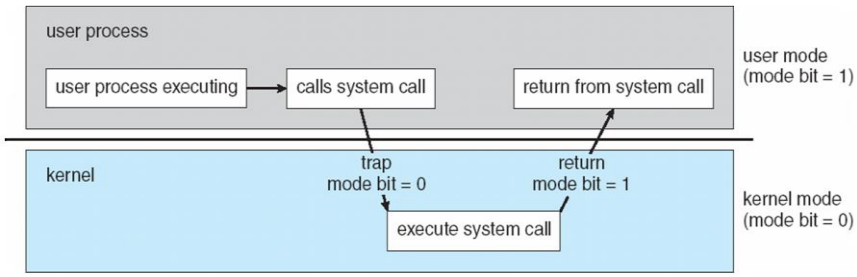
设备状态表

Computer Dynamics

- 计算机动力学
  - 直接存储器存取（DMA）
    - DMA用于智能高速I/O设备，能够以接近内存的速度传输信息。
    - DMA设备控制器将数据块从缓冲存储器直接传输到主存储器，无需CPU干预。
    - 每个块只生成一个中断，而不是每个字节生成一个中断。
    - 给定DMA控制器这些信息后，CPU返回工作：
      - 磁盘地址
      - 内存地址
      - 字节数

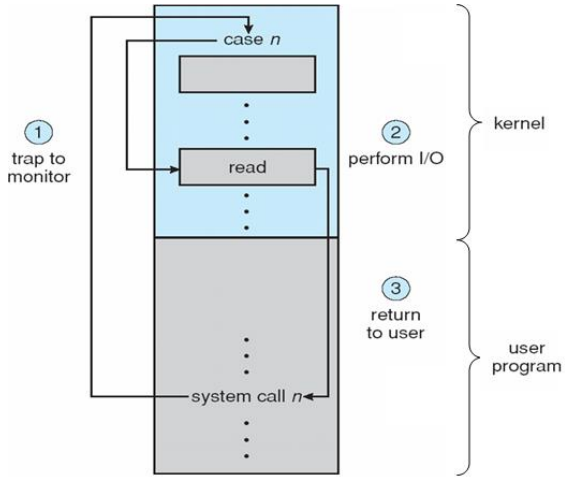
Computer Dynamics

- 计算机动力学
  - 系统调用
    - 系统调用是进程用于请求操作系统执行操作的方法：
      - 在参数准备好后，系统调用使用trap指令将控制转移到操作系统内核中请求的服务例程。
      - 系统验证参数是否正确合法，并执行请求。
      - 将控制权返回到系统调用后的指令。



Computer Dynamics

- 计算机动力学
  - 系统调用
    - 用于执行I/O的系统调用。



Hardware Protection

- 硬件保护
  - 双模保护
    - 双模式操作确保不正确的程序不会导致其他程序（包括内核进程）错误执行。
    - 实现：提供硬件支持，以区分至少两种操作模式。模式位被添加到计算机硬件以指示当前模式：内核模式（0）或用户模式（1）。
      - 用户模式（用户态或目态）– 代表用户执行
      - 内核模式（又名内核态, 管态或系统态)–代表操作系统执行
    - 特权指令只能在内核模式下发出。
    - 从用户模式转移到内核模式的三种非编程控制转移方式：
      - 系统调用
      - 中断
      - 陷阱

Hardware Protection

- **硬件保护**
  - 双模保护
    - 系统调用
      - 进程请求系统服务，例如退出。
      - 类似于函数调用，但在进程之外。
      - 没有要调用的系统函数的地址。
      - 类似于远程进程调用（RPC）。
      - 整理EAX中的syscall id，EBX、ECX、EDX中的参数和exec syscall（int 0x80）
    - 中断
      - 外部异步事件触发上下文切换
        - 例如，定时器、I/O设备
      - 独立于用户进程
    - 陷阱
      - 进程中的内部同步事件触发上下文切换。
        - 例如，保护违规（段错误），除以零...

Hardware Protection

- 硬件保护
  - I/O保护
    - 用户进程可能会意外或有目的地试图通过非法I/O指令中断正常操作
      - 所有I/O设备都需要受到保护，以防止用户的不当行为
        - 例如，防止当前程序读取下一个作业的控制卡
      - 所有I/O指令都必须是特权指令
    - 鉴于I/O指令具有特权，用户程序如何执行I/O？
      - 解决方案：来自用户程序的系统调用
  - 内存保护
    - 为了保护内存，基址寄存器和限制寄存器用于确定程序可以访问的合法地址范围。
    - 超出定义范围的内存受到保护。
    - 基本寄存器和限制寄存器的加载指令是特权指令

Hardware Protection

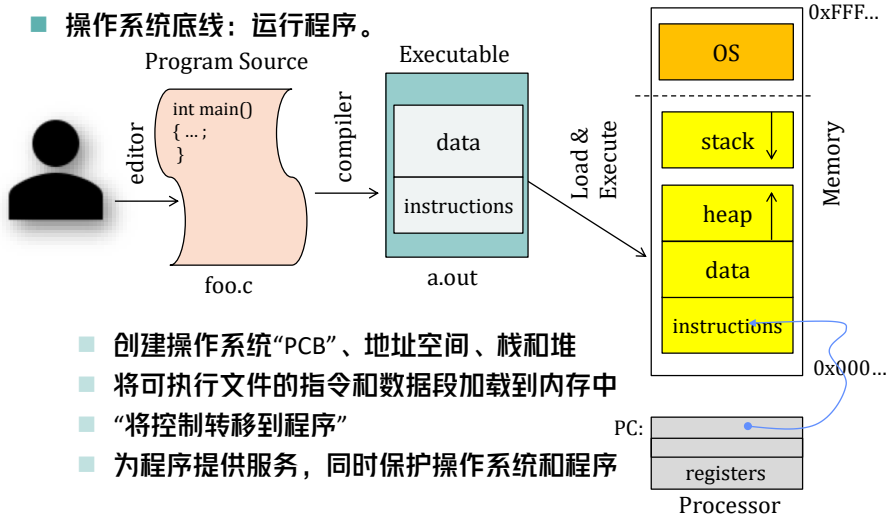
- 硬件保护
  - CPU保护
    - 定时器
      - 定时器在指定时间后中断计算机，以确保操作系统保持控制。
      - 可编程间隔定时器用于定时、周期性中断。
      - 设置定时器是一条特权指令。
    - 定时器通常用于实现分时系统。
    - 定时器设置为在一段时间后中断计算机。
      - 定时器在每个物理时钟滴答声中递减。
      - 当定时器达到值0时，发生中断。
      - 加载定时器（操作系统设置计数）是一条特权指令。
      - 在调度进程之前设置，以重新控制或终止超出分配时间的程序

Fundamental OS Concepts

- 基本操作系统概念
  - 线程：执行上下文
    - 完全描述程序状态
    - 程序计数器、寄存器、执行标志、堆栈
  - 地址空间（带或不带translation）
    - 程序可访问的内存地址集（用于读或写）
    - 可能不同于物理机的内存空间（程序在虚拟地址空间中运行）
  - 进程：正在运行的程序的实例
    - 受保护的地址空间+一个或多个线程
  - 双模式操作/保护
    - 只有“系统”才有能力访问某些资源
    - 结合translation，将程序彼此隔离，并将操作系统与程序隔离

Fundamental OS Concepts

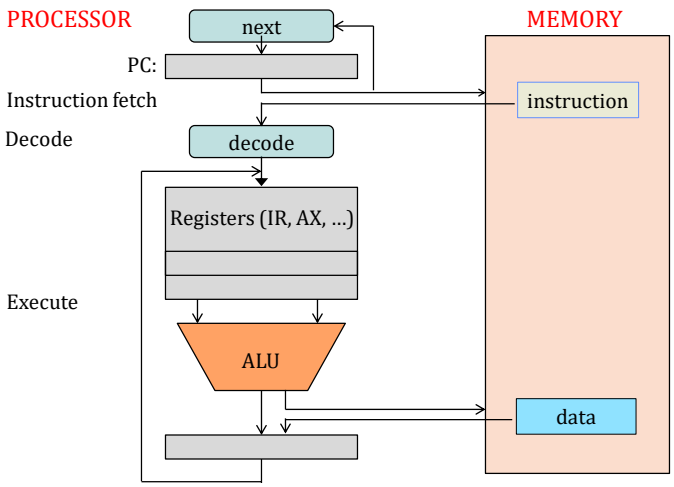
- 基本操作系统概念
  - 操作系统底线：运行程序。



  - 创建操作系统“PCB”、地址空间、栈和堆
  - 将可执行文件的指令和数据段加载到内存中
  - “将控制转移到程序”
  - 为程序提供服务，同时保护操作系统和程序

Fundamental OS Concepts

- 基本操作系统概念
  - 指令周期。

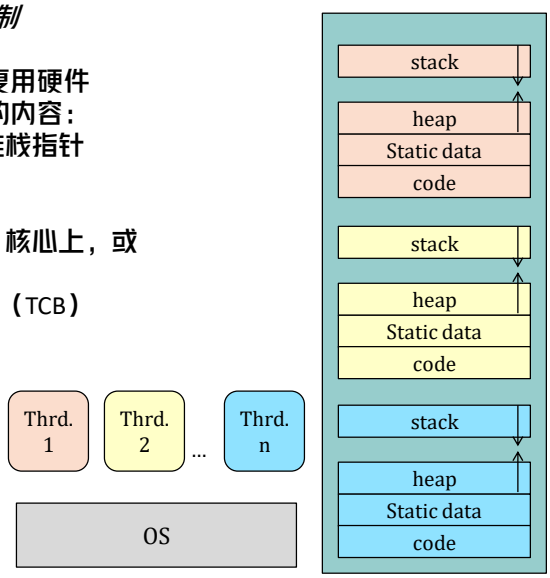


- 线程
  - 单一、唯一的执行上下文
    - 程序计数器、寄存器、执行标志、堆栈、内存状态
  - 当线程驻留在处理器寄存器中时，它正在处理器（核）上执行
  - 驻留意味着：寄存器保存线程的根状态（上下文）：
    - 包括程序计数器（PC）寄存器和当前正在执行的指令
      - PC指向存储在内存中的下一条指令。
    - 包括正在进行的计算的即时值
      - 可以包括实际值（如整数）或指向内存中值的指针
    - 堆栈指针保存线程自身堆栈顶部（内存中）的地址。
    - 剩下的都在“内存中”

- 线程
  - 当线程的状态未加载到处理器（非驻留）时，线程被挂起（不执行）。
    - 处理器状态指向其他线程
    - 程序计数器寄存器未指向此线程的下一条指令
    - 通常：每个寄存器的最后一个值的副本将存储在内存中

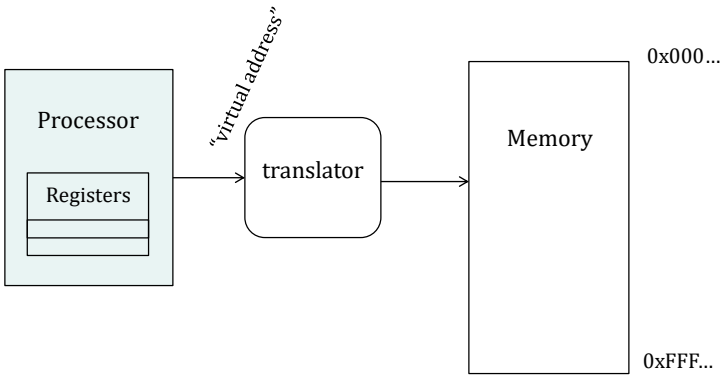
Fundamental OS Concepts

- **线程**
  - *多道程序-多线程控制*
  - 线程是虚拟核心。
  - 多线程：实时多路复用硬件
  - 虚拟核心（线程）的内容：
    - 程序计数器，堆栈指针
    - 寄存器
  - 在哪里？
    - 在真实（物理）核心上，或
    - 保存在内存中
      - 线程控制块（TCB）



Fundamental OS Concepts

- **地址空间**
  - 程序在不同于机器物理内存空间的地址空间中运行。

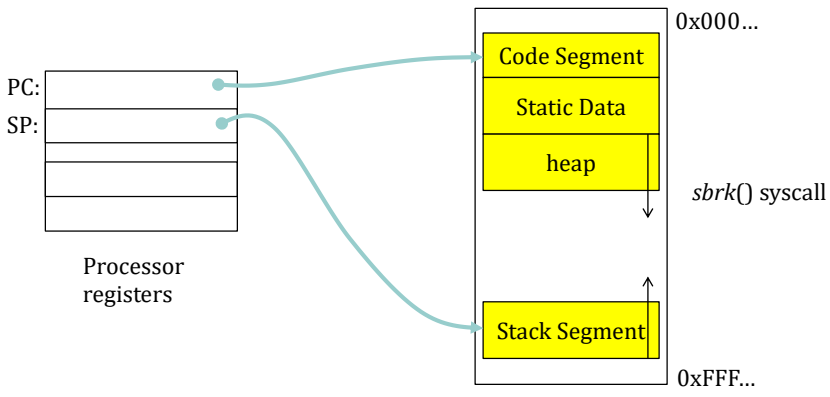


Fundamental OS Concepts

- 地址空间
  - 定义
    - 可访问地址集及其关联的状态
    - 32位机器上，约40亿 (2^{32})
  - 当你读或写一个地址时会发生什么？
    - 也许像常规内存一样操作
    - 可能导致I/O操作
      - （内存映射I/O）
    - 导致程序中止（segfault）？
    - 与其他程序通信
    - ...

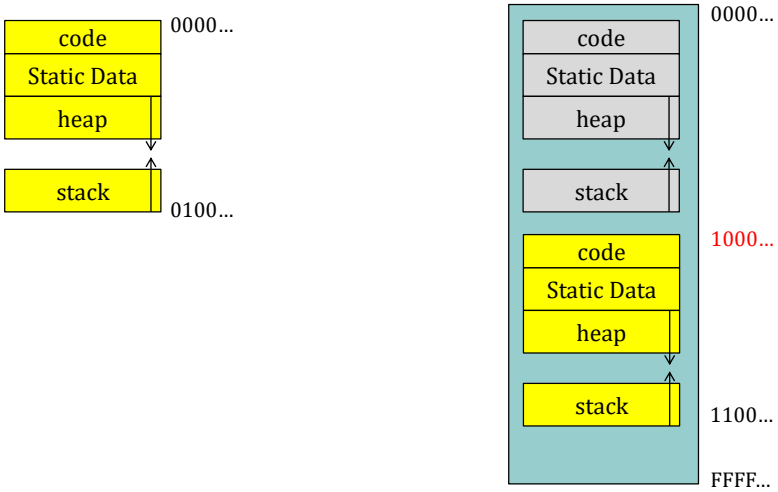
Fundamental OS Concepts

- 地址空间
  - 典型结构。



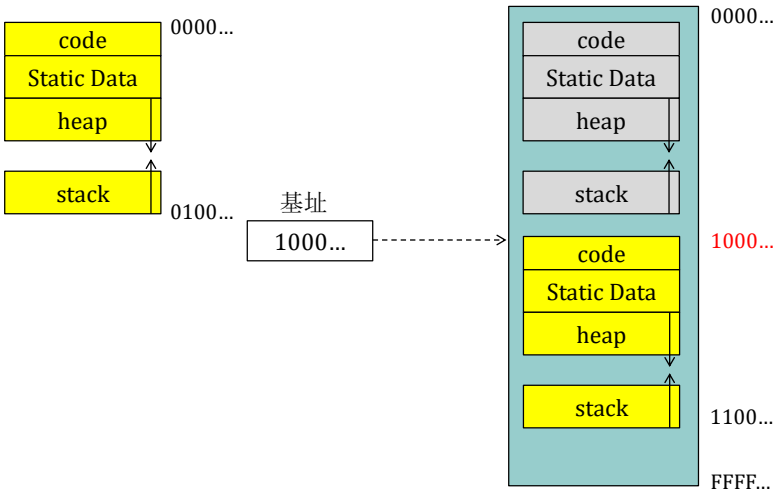
Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



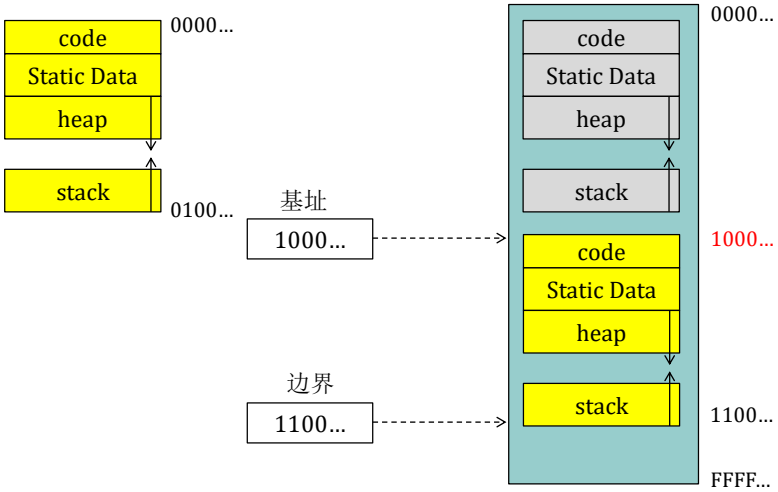
Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



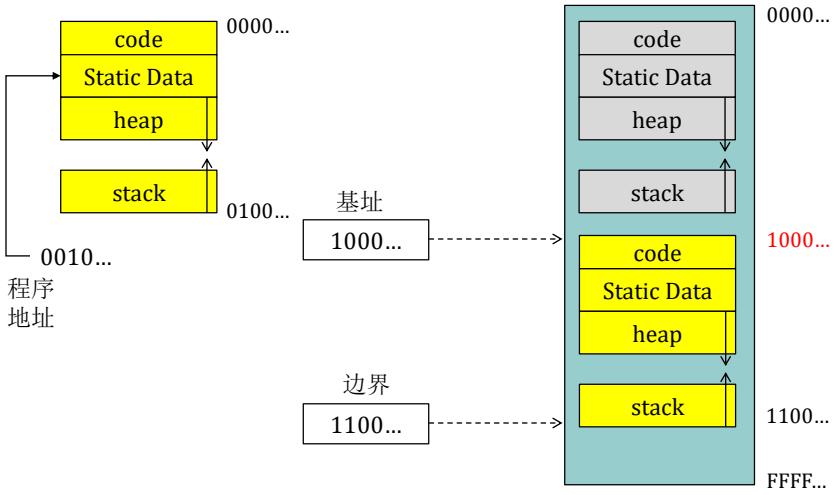
Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



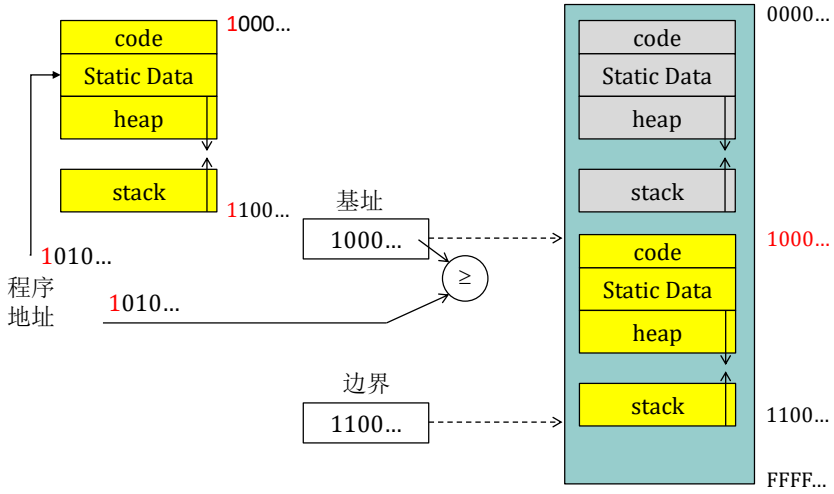
Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



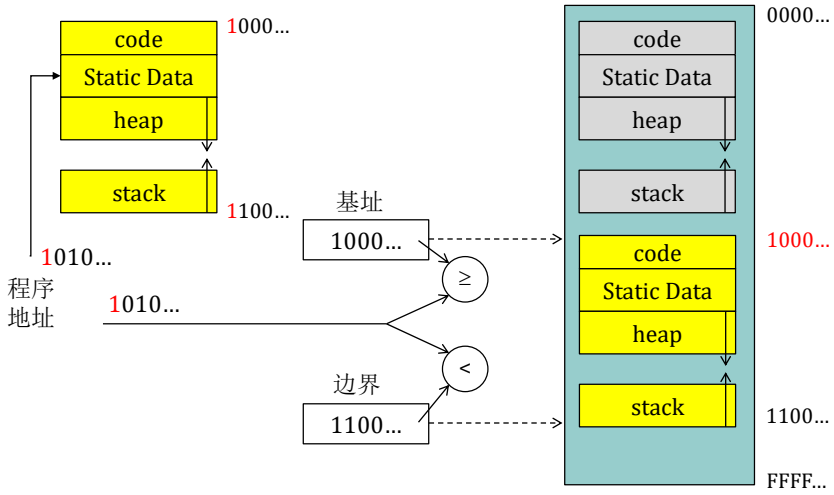
Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



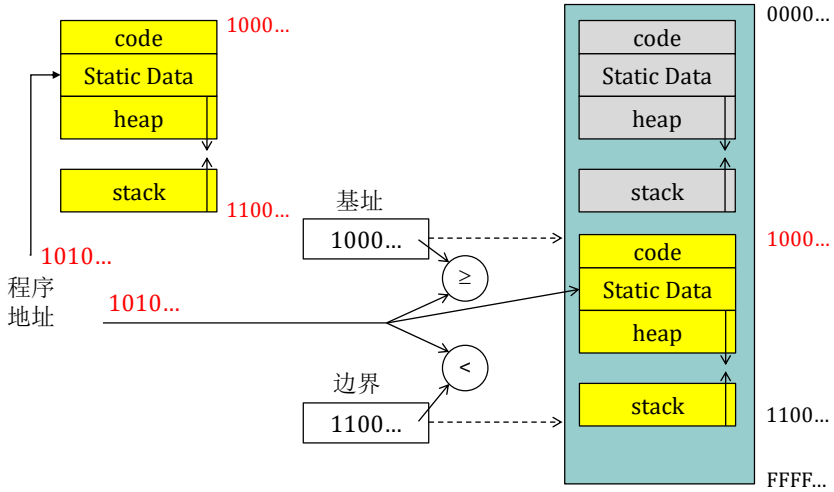
Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



Fundamental OS Concepts

- 地址空间
  - 基址和边界-保护操作系统并隔离程序。



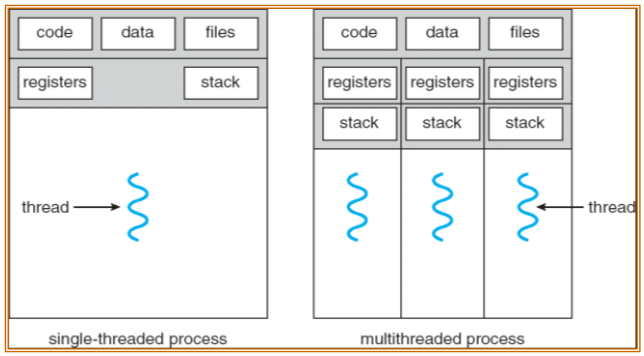
  - 装载时需要重新定位

Fundamental OS Concepts

- 进程
  - 定义
    - 权限受限的执行环境
      - 具有一个或多个线程的地址空间
      - 拥有内存（映射页）
      - 拥有文件描述符、文件系统上下文...
      - 封装共享进程资源的一个或多个线程
  - 应用程序作为进程执行。
    - 复杂的应用程序可以派生/执行子进程（fork/exec）
  - 为什么是进程？
    - 相互保护，操作系统不受它们的影响
    - 并发执行
    - 操作系统处理的基本单元

Fundamental OS Concepts

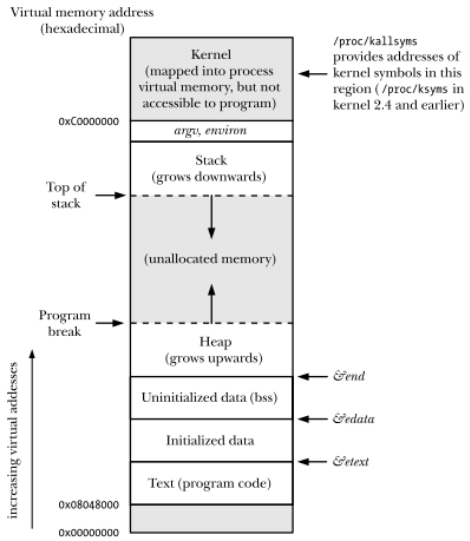
- **进程**
  - 单线程和多线程进程。



  - 线程封装并发：“活动”组件
  - 地址空间封装保护：“被动”部分
    - 防止有缺陷的程序破坏系统
  - 为什么每个地址空间有多个线程？

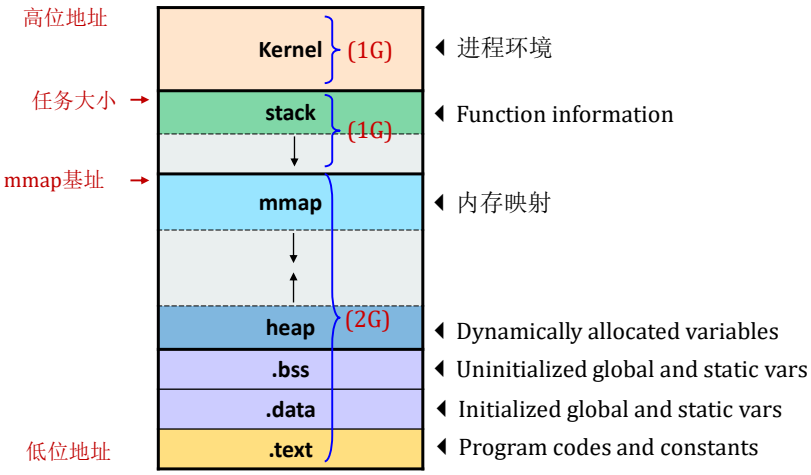
Fundamental OS Concepts

- **进程**
  - Linux/IA-32上进程的典型内存布局。



Fundamental OS Concepts

- **进程**
  - Linux/IA-32上进程的典型内存布局。



进程虚拟内存（PVM）

Resource Management

- **进程管理**
  - 进程是正在执行的程序
    - 程序是一个被动实体，静态，存储在磁盘上的文件内容
    - 进程是一个活动实体，动态，具有系统上下文
  - 进程需要资源来完成其任务
    - CPU（寄存器）、内存、I/O设备、文件
    - 数据初始化
    - 进程终止需要回收任何可重用资源
  - 进程是系统中的一个工作单元。操作系统负责与进程管理相关的以下活动：
    - 创建和删除用户和系统进程
    - 暂停和恢复进程
    - 提供进程同步机制
    - 提供进程通信机制
    - 提供死锁处理机制

Resource Management

- 内存管理
  - 主存储器是现代计算机系统运行的核心，是由CPU和I/O设备共享的快速访问数据的存储库。
    - 通常是CPU能够直接寻址和访问的唯一大型存储设备。
    - 要执行一个程序，所需的全部（或部分）指令和数据必须在主存中。
  - 操作系统负责与内存管理相关的以下活动：
    - 跟踪内存的哪些部分当前正在被谁使用
    - 决定何时以及哪些进程（或进程的一部分）和数据要移入和移出内存。
      - 优化CPU利用率和计算机对用户的响应。
    - 根据需要分配和释放内存空间

Resource Management

- 文件系统管理
  - 计算机在几种不同类型的物理介质上存储信息，每种介质都有自己的特性和物理组织。
    - 访问速度、容量、数据传输速率和访问方法（顺序或随机）。
  - 文件是相关信息的集合，通常用于表示程序和数据。
  - 操作系统通过管理大容量存储介质及其控制设备来实现文件的抽象概念。
  - 文件通常被组织到目录中。访问控制应用于大多数文件系统，以确定谁可以访问什么。
  - 操作系统的文件管理活动：
    - 创建和删除文件
    - 创建和删除目录
    - 支持用于操作文件和目录的原语Primitives
    - 将文件映射到大容量存储
    - 在稳定（非易失性）存储介质上备份文件。

大容量存储管理

- 通常，HDD和其他NVM设备等大容量存储器用于存储不适合主存储器的数据或必须保存“长”时间的数据。
  - 计算机运行的整体速度可能取决于磁盘子系统及其算法。
  - 对这些辅助存储的正确管理至关重要。
- 某些存储可能不需要很快：
  - 第三级存储包括光盘存储、磁带存储。
  - 支持WORM（写入一次，读取多次）或RW（读写）方式
- 操作系统的大容量存储管理活动：
  - 挂载和卸载
  - 可用空间管理
  - 存储分配
  - 磁盘调度
  - 分区
  - 保护

缓存管理

- 缓存是计算机系统的一个重要原理。由于缓存的大小有限，缓存管理是一个重要的设计问题。
- 在多处理器环境中，缓存管理变得更加复杂。除了维护内部寄存器外，每个CPU还包含一个本地缓存。
  - 缓存一致性
    - 例如，整数A的副本可能同时存在于多个缓存中。由于多个CPU都可以并行执行，因此必须确保对一个缓存中A值的更新会立即反映在A所在的所有其他缓存中。这种缓存一致性通常是一个硬件问题（在操作系统级别以下处理）。
- 在分布式环境中，这种情况变得更加复杂。同一文件的多个副本可以保存在不同的计算机上。由于可以同时访问和更新各种副本，因此一些分布式系统确保在一个位置更新副本时，所有其他副本都能尽快更新。

Resource Management

- I/O系统管理
  - 操作系统的一个目的是向用户隐藏特定硬件设备的特性。
  - I/O子系统由几个组件组成：
    - I/O的内存管理，包括
      - 缓冲Buffering
        - 在传输数据时临时存储数据。
      - 缓存Caching
        - 将部分数据存储在更快的存储器中以提高性能。
      - 假脱机SPOOLing
        - 一个作业的输出与其他作业的输入重叠。
    - 通用设备驱动程序接口
      - 特定硬件设备的设备驱动程序
        - 只有设备驱动程序知道分配给它的特定设备的特性。

Free and Open Source Operating Systems

- 自由和开源操作系统
  - 免费操作系统和开源操作系统都有源代码格式。
  - 自由软件（Free/Libre Software）不仅使源代码可用，而且许可无成本使用、重新分发和修改。但有些开源软件不是“免费的”
    - GNU/Linux是最著名的开源操作系统，有些发行版是免费的，有些发行版是开源的。
    - Microsoft Windows是一个著名的反例——封闭源代码。Windows是专利软件——微软拥有，限制使用，并小心地保护它的源代码。
    - 苹果的MacOS操作系统采用了混合方法：它包含一个名为Darwin的开源内核，但也包含专有的封闭源代码组件。

自由和开源操作系统

- 通过分析源代码学习操作系统
  - 学生可以修改操作系统的源代码，然后编译并运行来尝试这些更改，这是一个很好的学习工具。
- 开源社区
  - 一群感兴趣的、无偿的程序员通过帮助编写、调试、分析，提供支持和建议更改，为开源代码做出了贡献。
  - 可以说，开放源代码比封闭源代码更安全，因为有更多的眼睛在看代码。由于使用和查看代码的人数较多，开源代码的bug往往会更快地被发现和修复。

GNU项目

- 1984年, *Richard M.Stallman*开始开发一个免费的、与UNIX兼容的操作系统，名为GNU（"GNU's Not UNIX!"递归首字母缩写），并于1985年发表了GNU宣言。他还成立了自由软件基金会（FSF），目的是鼓励自由软件的使用和开发。
- GNU通用公共许可证（GPL）是发布自由软件的通用许可证。基本上，GPL要求源代码与任何二进制文件一起分发，并且所有副本（包括修改版本）都在同一GPL许可证下发布。

Free and Open Source Operating Systems

- GNU/Linux
  - GNU内核未在黄金时间做好准备。1991年，芬兰的一名学生Linus Torvalds使用GNU编译器和工具发布了一个基本的类UNIX内核，并邀请世界各地的人参与。互联网的出现意味着任何感兴趣的人都可以下载源代码，修改它，并向Torvalds提交更改。
  - 1992年，Torvalds在GPL下重新发布了Linux，使其成为自由软件。
  - 由此产生的GNU/Linux操作系统（内核正确地称为Linux，但完整的操作系统包括GNU工具，称为GNU/Linux的）产生了数百个独特的发行版或自定义版本。主要分配包括：
    - Red Hat
    - SUSE
    - Fedora
    - Debian
    - Slackware
    - Ubuntu.

Free and Open Source Operating Systems

- BSD UNIX
  - 1978年，BSD UNIX发布于加州大学伯克利分校（UCB），作为AT&T UNIX的衍生版本，需要AT&T的许可证。
    - 1994年发布的4.4BSD-lite是其功能齐全的开源版本。
  - BSD UNIX有许多发行版，包括FreeBSD、NetBSD、OpenBSD和DragonflyBSD。
    - FreeBSD源代码地址：https://svnweb.freebsd.org
  - Darwin是macOS的核心内核组件，它基于BSD UNIX，也是开源的。
    - Darwin源代码地址：http://www.opensource.apple.com/