OPERATING SYSTEMS

Introduction

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

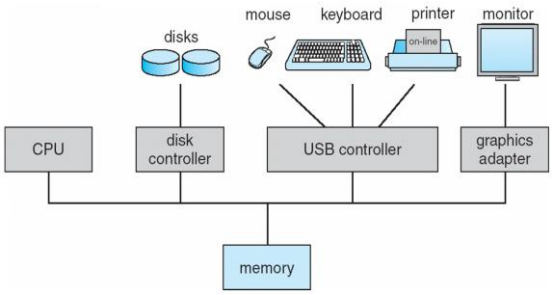
中山大學 SUN YAT-SEN UNIVERSITY

Introduction to Operating Systems

- 大纲
  - 操作系统做什么
  - 计算机系统组件
  - 操作系统视图
  - 操作系统的演变

What Operating Systems Do

- **操作系统做什么**
  - 现代计算机包括：
    - 一个或多个处理器
    - 主存
    - 大容量存储
    - 各种输入/输出设备
      - 打印机、鼠标、键盘、显示器等。
      - 网络适配器等。



计算机硬件组成

What Operating Systems Do

- **操作系统做什么**
  - 充当计算机用户和硬件之间的中介或接口的程序。
    - 复杂硬件设备的简单抽象
    - *保护对共享资源的访问*
      - *安全和认证*
    - *逻辑实体之间的通信*
  - 操作系统的目标是：
    - 控制/执行用户或应用程序
    - 使计算机系统便于使用
    - 简化用户问题的解决
    - 以有效的方式使用计算机硬件

What Operating Systems Do

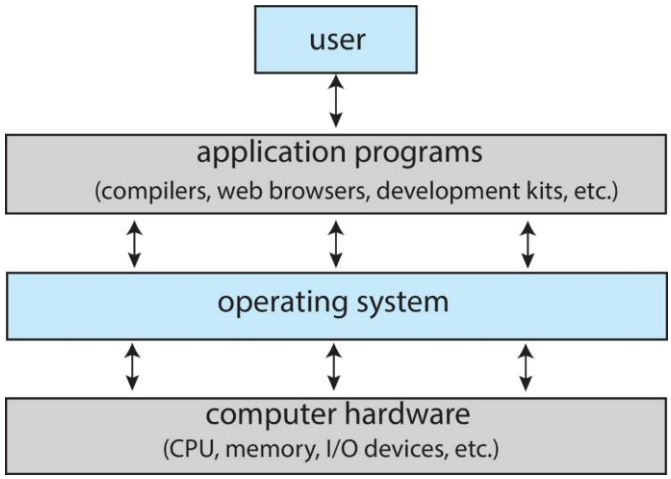
- **操作系统做什么**
  - 操作系统需要做什么的不同观点
    - 用户需要方便、易用和良好的性能
      - 他们不关心资源利用率
      - 大型机或小型计算机等共享计算机必须让所有用户都满意
    - 用户使用专用系统（例如工作站）的专用资源，但经常使用来自服务器的共享资源。
    - 手持式计算机资源匮乏，优化了可用性和电池寿命。
    - 有些计算机很少或没有用户界面，例如设备和汽车中的嵌入式计算机。

Computer System Components

- **计算机系统组件**
  - 硬件
    - 提供基本的计算资源
      - CPU、内存、I/O设备、通信。
  - 操作系统
    - 控制和协调不同用户的各种应用程序之间的硬件使用
  - 应用程序（应用系统）
    - 系统资源用于解决用户计算问题的方式
      - 文字处理器、编译器、网络浏览器、数据库系统、视频游戏。
  - 使用者
    - 人、机器、其他计算机。

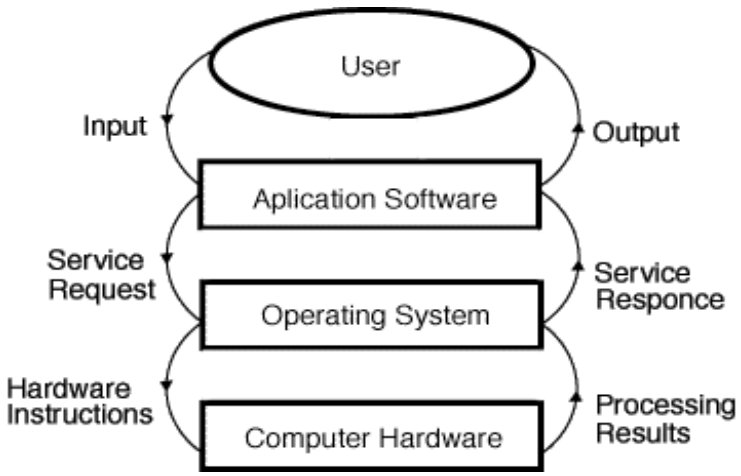
Computer System Components

- **计算机系统组件**
  - 计算机系统组件的抽象视图。



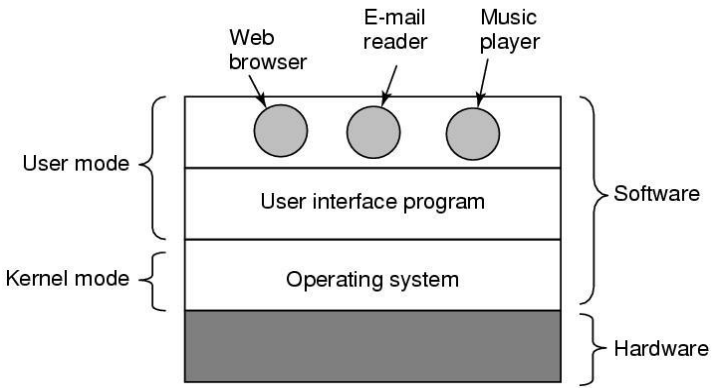
Computer System Components

- **计算机系统组件**
  - 动态视图。



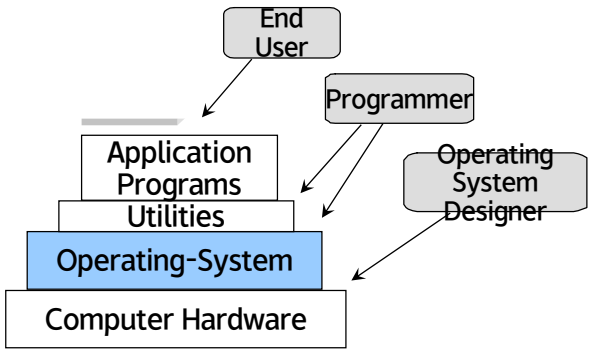
Computer System Components

- 计算机系统组件
  - 现代计算机系统的示意图。



Computer System Components

- 计算机系统组件
  - 计算机系统的层次。



Views of Operating Systems

- **操作系统视图**
  - 为应用程序提供一致的抽象，即使是在不同的硬件上
    - 文件系统、窗口系统、通信…
    - 进程、线程
    - 虚拟机、容器
    - 命名系统
  - 管理多个应用程序之间共享的资源
    - 内存、CPU、存储器…
  - 通过特定的算法和技术实现
    - 调度
    - 并发性
    - 交易
    - 安全
  - 跨越巨大的规模——从1到数十亿

Views of Operating Systems

- **操作系统视图**
  - 由操作系统提供的服务
    - 程序创建设施
      - 编辑器、编译器、链接器、调试器等。
    - 程序执行
      - 在内存中加载、I/O和文件初始化。
    - 访问I/O和文件
      - 处理I/O和文件格式的细节。
    - 系统访问
      - 解决资源争用的冲突(解决资源冲突)
      - 在获取资源和数据方面的保护

Views of Operating Systems

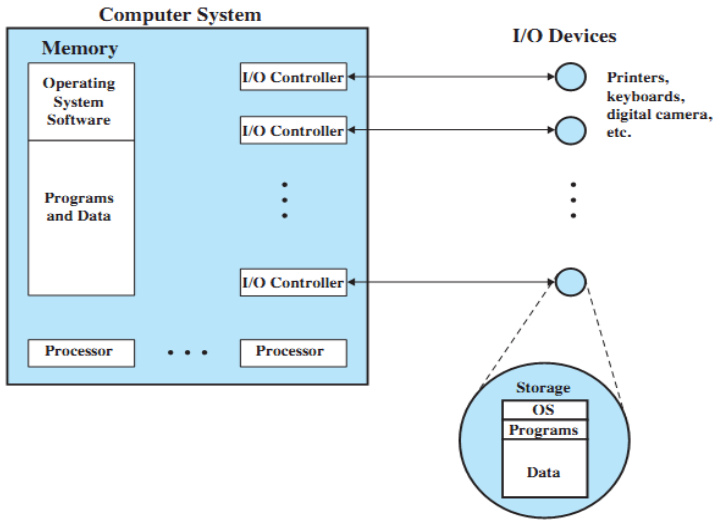
- **操作系统视图**
  - 操作系统有三种经典观点
    - 资源管理器
      - 管理和分配资源
        - 有点自下而上的观点
    - 控制程序
      - 控制用户程序的执行和I/O设备的操作
        - 有点像黑盒视图
    - 命令执行器
      - 提供运行用户命令的环境
        - 有点像自顶向下的视图
  - 一种现代操作系统观
    - 虚拟机

Views of Operating Systems

- **操作系统视图**
  - 作为资源管理器的操作系统
    - 管理和保护多个计算机资源
      - CPU、进程、内部/外部内存、任务、应用程序、用户、通信通道等。
    - 处理并将资源分配给多个用户或在同一时间和空间运行的多个程序
      - 例如，处理器时间、内存、I/O设备
    - 在相互冲突的请求之间作出决定，有效和公平地使用资源
      - 例如，最大化吞吐量，最小化响应时间
    - 某些操作系统以资源管理器视图命名
      - DEC RSX-Resource Sharing eXecutive资源共享执行官
      - MIT Multics-MULTiplexed Information & Computing Services多路信息与计算服务
      - IBM MFT/MVT-Multiple Fixed/Variable Tasks多个固定/可变任务
      - IBM MVS-Multiple Virtual Storage多虚拟存储
      - DEC VMS-Virtual Memory System虚拟内存系统
      - MVS TSO-Time Sharing Option分时选项
      - CTSS-Compatible Time Sharing System兼容分时系统

Views of Operating Systems

- **操作系统视图**
  - 作为资源管理器的操作系统



Views of Operating Systems

- **操作系统视图**
  - 作为控制程序的操作系统
    - 以集成方式管理复杂计算机系统的所有组件
    - 控制用户程序和I/O设备的执行，以防止错误和不当使用计算机资源
    - 检查并保护计算机
      - 监视器、主管、执行器、控制器、主控器、协调员…
    - 某些操作系统以控制程序视图命名
      - Unisys MCP–主控制程序
      - DR CP/M–控制程序/微型计算机
      - IBM VM/CP–虚拟机控制程序
      - IBM AIX–高级交互式执行器
      - DEC RSX–资源共享执行官

Views of Operating Systems

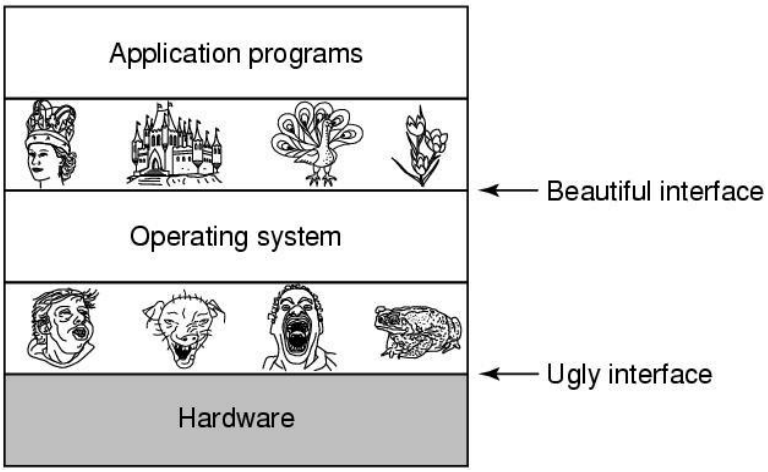
- **操作系统视图**
  - 作为命令执行器的操作系统
    - 用户和机器之间的接口
    - 向用户提供服务/公用设施
    - 为用户提供方便的CLI（命令行/语言接口或命令解释器），在UNIX中也称为Shell，用于输入用户命令
    - 某些操作系统以命令Executer视图命名
      - IBM AIX–高级交互式执行器
      - IBM VM/CMS–对话监控系统

Views of Operating Systems

- **操作系统视图**
  - 作为虚拟机的操作系统
    - 充当用户和硬件之间的接口，隐藏硬件的详细信息
      - 像I/O设备这样的硬件。
    - 从低级（物理）资源构造高级（虚拟）资源
      - 例如，文件
    - OS被定义为在裸硬件上执行的软件增强的集合，最终达到作为高级编程环境的高级虚拟机。
      - 虚拟机=软件增强=扩展机器=抽象机器

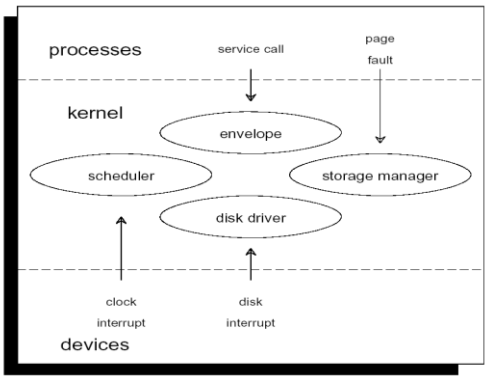
Views of Operating Systems

- **操作系统视图**
  - 作为虚拟机的操作系统



Views of Operating Systems

- **操作系统的定义**
  - 操作系统没有一个公认的定义。
    - “当您订购操作系统时，供应商提供的一切”是一个很好的近似值，但差异很大。
    - “在计算机上始终运行的一个程序”只是内核。
      - 其他一切要么是一个系统程序（实用程序，随操作系统一起提供），要么是一个应用程序。

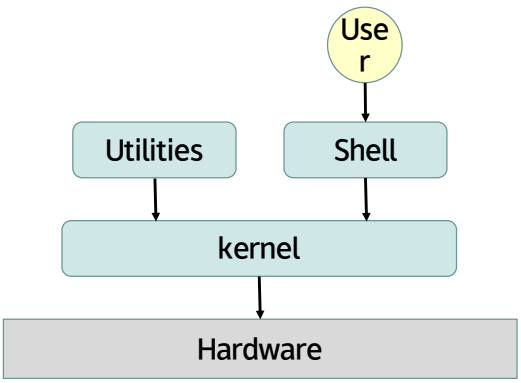


Views of Operating Systems

- **操作系统的定义**
  - 命令行界面（CLI）允许用户直接输入命令。
    - 有时在内核中实现，有时由系统程序实现
    - 有时会实现多种风格–shell
    - 主要从用户获取命令并执行它
    - 有时命令内置，有时只是程序名称；如果是后者，则添加新功能不需要修改shell。
  - shell以前在内核中，但现在是内核之外的（第一个）实用程序。
    - 易于更改/调试
    - 其中很多
      - sh、bsh、csh、ksh、tcsh、wsh、bash
    - 可以在它们之间切换
      - chsh

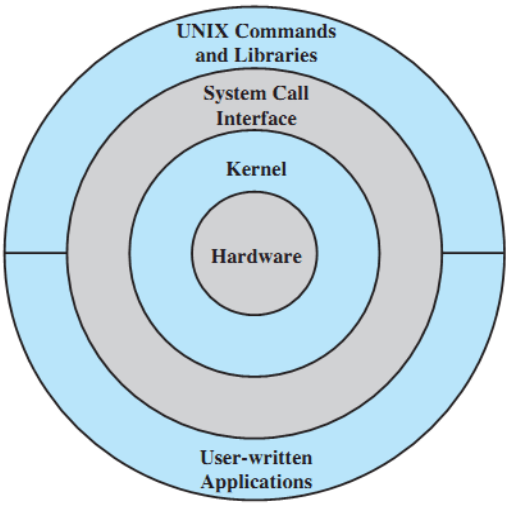
Views of Operating Systems

- **Unix shell和实用程序**



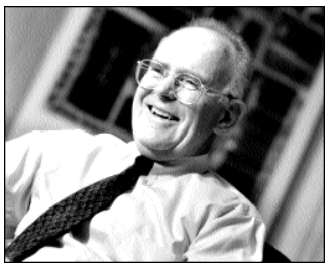
Views of Operating Systems

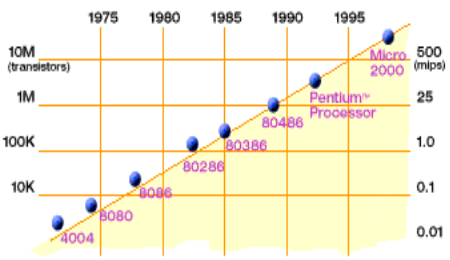
- 通用UNIX体系结构



Views of Operating Systems

- *摩尔定律*
  - *戈登·摩尔（英特尔联合创始人）在1965年预测，半导体芯片的晶体管密度大约每18个月翻一番。*
    - 微处理器变得更小、更密集、更强大。
  - *摩尔定律（正式）于2016年2月结束。*
    - 不再是每18个月翻番…
      - 甚至每24个月
    - 供应商转向3D堆叠芯片



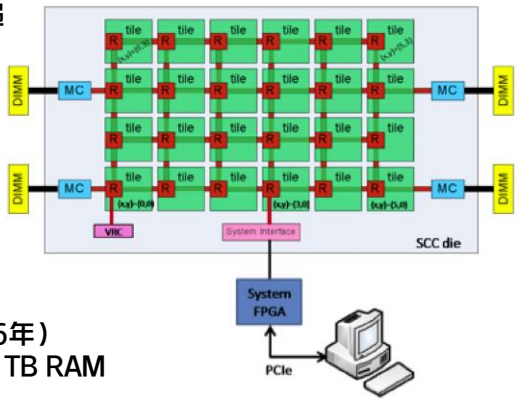


Views of Operating Systems

- **多核芯片**
  - “多核”是指每个芯片有多个处理器
    - 64？128？很难说确切的界限
  - 英特尔80核多核芯片（2007年2月）
    - 80个单芯
    - 两台FP引擎/核心
    - 网状网络
    - 1亿个晶体管
  - 如何使用多个核心芯片编程？
    - 使用2个CPU进行视频/音频处理
    - 1用于文字处理器，1用于浏览器
    - 使用76进行病毒检查？
  - 必须在所有级别上利用并行性。

Views of Operating Systems

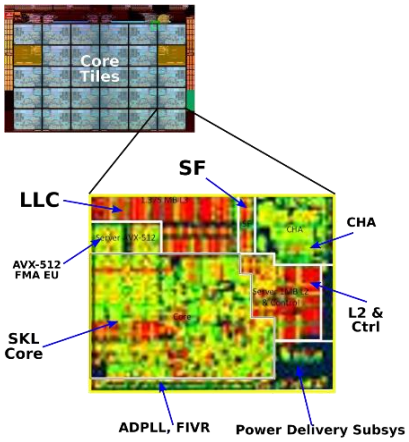
- **多核芯片**
  - 英特尔单芯片云计算机（2010年8月SCC）
    - 24“瓷砖”，每块瓷砖两芯（48芯）
    - 24路由器网状网络
    - 4个DDR3内存控制器
    - 硬件支持消息传递
  - 亚马逊X1实例（2016年）
    - 128个虚拟核，2 TB RAM



管理控制台PC

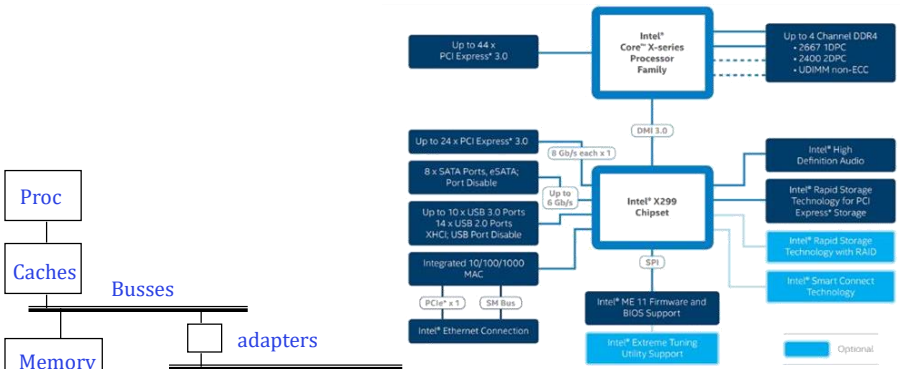
Views of Operating Systems

- 世界是平行的： Intel SkyLake英特尔天湖（2017）
  - 最多28芯，58个线程
    - 694 mm²模具尺寸（估计）
  - 许多不同的指令
    - 安全、图形
  - 片上缓存：
    - L2:28 MiB
    - 共享L3:38.5 MiB（非独占）
    - 基于目录的缓存一致性
  - 网络：
    - 片上网状互连
    - 快速片外网络直接
      支持8芯片连接
  - DRAM/芯片
    - 高达1.5 TiB
    - DDR4内存



Views of Operating Systems

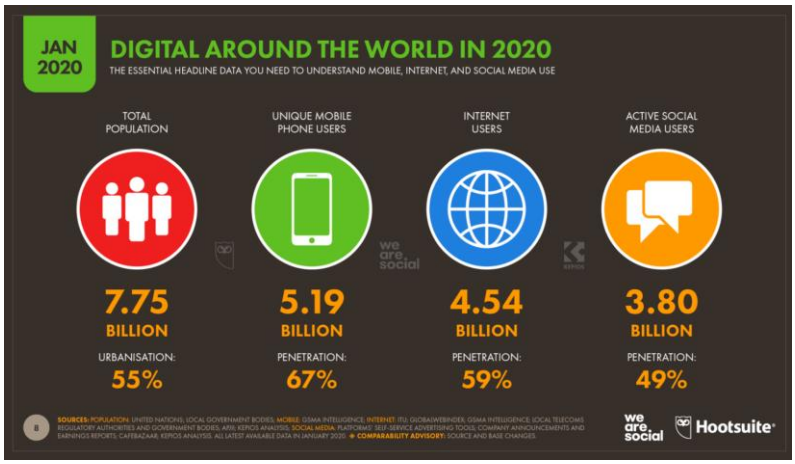
- 世界是平行的： Intel SkyLake 英特尔天湖（2017）
  - 硬件功能非常复杂。



英特尔Skylake-X I/O配置

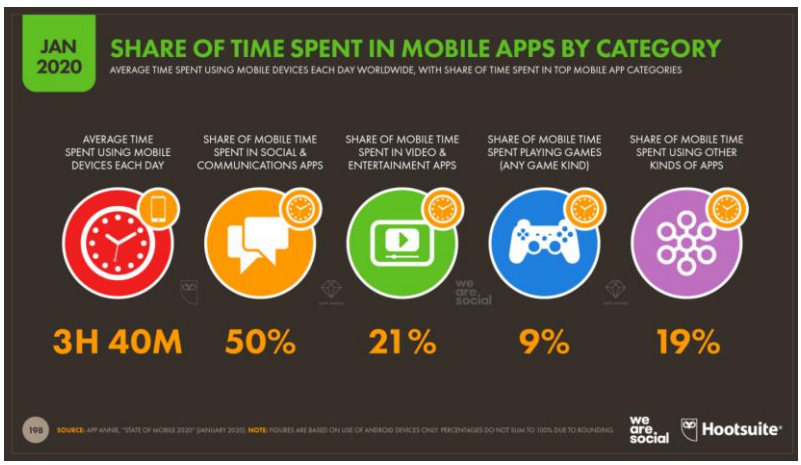
Views of Operating Systems

- 社会联系



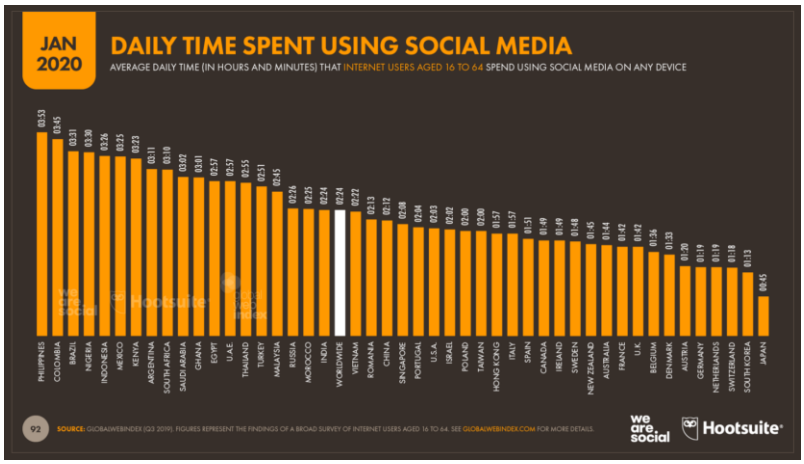
Views of Operating Systems

- 社会联系



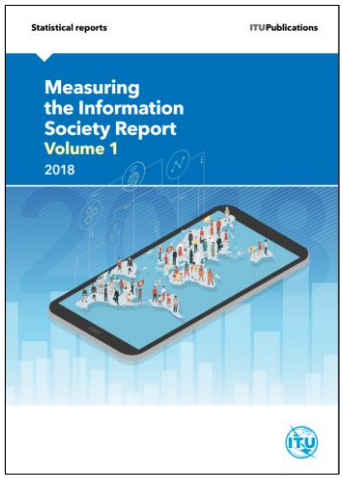
Views of Operating Systems

- **社会联系**



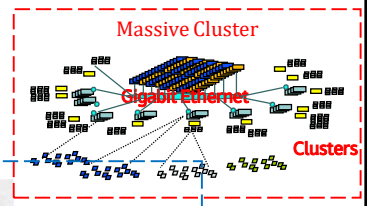
Views of Operating Systems

- **社会联系**
  - MISR-2018，ITU 2019



Views of Operating Systems

- **社会规模信息系统**
  - 世界是一个大型分布式系统
    - 一切事物中的微处理器
    - 背后的庞大基础设施



互联网连接

传感器网络的微机电系统

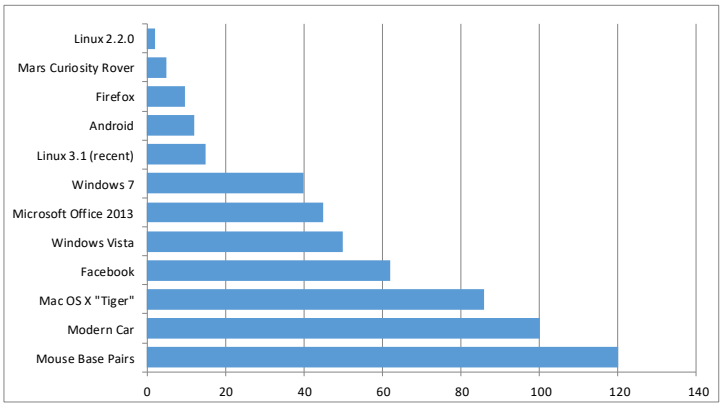
数据库
信息收集
远程存储
网络游戏
商业
…

Views of Operating Systems

- **复杂性的挑战**
  - 应用程序包括
    - …各种各样的软件模块…
    - …在各种设备(机器)上运行
      - …实现不同的硬件体系结构
      - …运行相互竞争的应用程序
      - …以意想不到的方式失败
      - …可能受到各种攻击

Views of Operating Systems

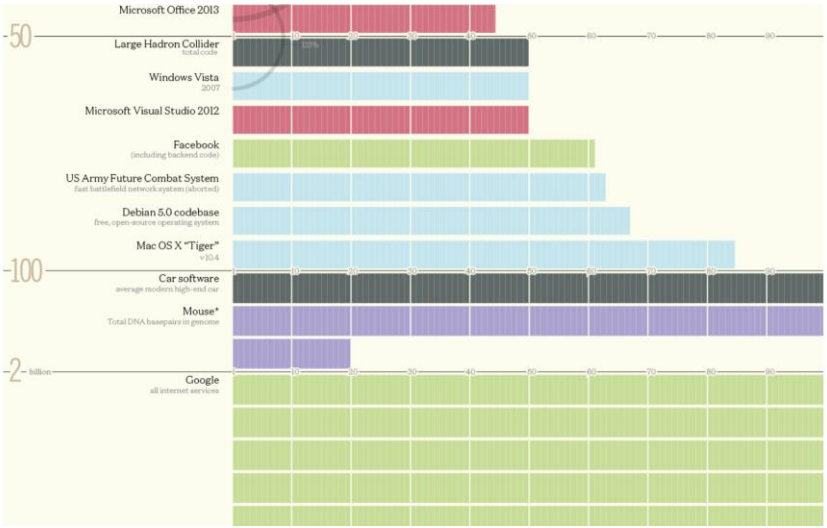
- **复杂性的挑战**
  - 逐步增加的软件复杂性
    - 针对所有可能的环境以及组件和设备的组合测试软件是不可行的。问题不在于是否存在bug，而在于它们管理得有多好。



数百万行软件源代码

Views of Operating Systems

- **复杂性的挑战**



数百万行软件源代码

Views of Operating Systems

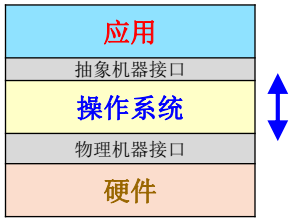
- **复杂性的挑战**
  - 每一块计算机硬件都是不同的
    - 不同的CPU
      - 奔腾、ARM、PowerPC、ColdFire、MIPS
    - 不同数量的内存、磁盘…
    - 不同类型的设备
      - 鼠标、键盘、传感器、摄像头、指纹读取器、触摸屏。
    - 不同的网络环境
      - 有线、DSL、无线…
  - 问题：
    - 程序员是否需要编写执行许多独立活动的单个程序？
    - 是否每个程序都必须针对每一个硬件进行修改？
    - 一个错误的程序会使一切崩溃吗？

Views of Operating Systems

- **示例：一些火星探测车（“探路者”）要求**
  - Pathfinder硬件限制/复杂性：
    - 20Mhz处理器、128MB DRAM、VxWorks操作系统
    - 照相机、科学仪器、电池、太阳能电池板和移动设备
    - 许多独立的过程一起工作
  - 不能轻易按下重置按钮！
    - 如有必要，必须自行重新启动
    - 必须始终能够接收来自地球的命令
  - 独立程序不得干扰
    - 假设MUT（火星通用翻译模块）有bug
    - 最好不要破坏天线定位软件！
  - 此外，所有软件都可能偶尔崩溃
    - 自动重启，诊断发送至地球
    - 保存结果的定期检查点？
  - 某些功能对时间至关重要：
    - 在撞到东西之前需要停下来吗
    - 通信必须跟踪地球轨道

Views of Operating Systems

- **操作系统抽象底层硬件**
  - 操作系统抽象
    - 处理器→ 线程
    - 机器→ 进程
    - 网络→ Sockets
    - 内存→ 地址空间
    - 磁盘、SSD…→ 文件
  - 操作系统目标
    - 消除软件/硬件转变（解决复杂性）
    - 优化方便性、利用率、可靠性…
      - 帮助程序员
    - 保护进程和内核
      - 运行多个应用程序时，防止它们互相干扰或破坏操作系统
  - 对于任何操作系统领域（如文件系统、虚拟内存、网络、调度），考虑：
    - 要处理什么*硬件接口*？（物理现实）
    - 提供什么*软件接口*？（更好的抽象）

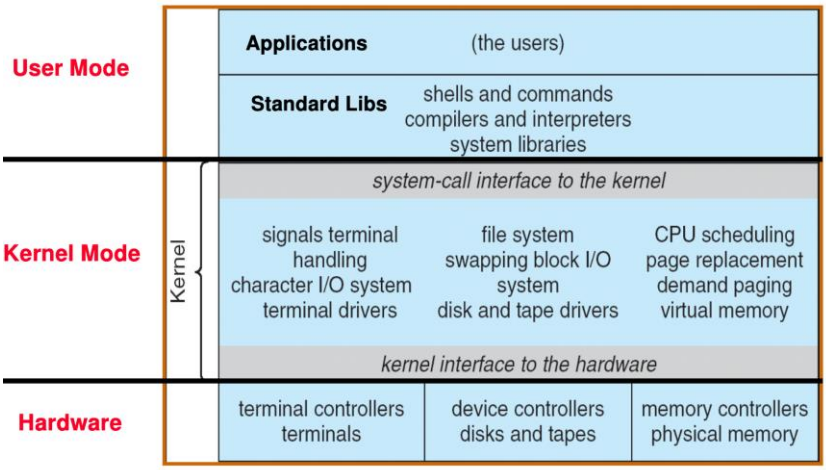


Views of Operating Systems

- **保护过程**
  - 使用硬件提供的两个功能来保护进程
    - 双模式操作
      - 硬件至少提供两种模式：
        - 内核模式（或“监督”/“受保护”模式）
        - 用户模式
      - 进程以用户或内核模式运行。在用户模式下运行时禁止某些操作。
      - 仔细控制用户模式和内核模式之间的转换
        - 系统调用、中断、异常。
    - 地址转换
      - 每个进程都有一个独立的地址空间。
      - 硬件将虚拟地址转换为物理地址。
      - 政策
        - 任何程序都不能读取或写入另一个程序或操作系统的内存。

Views of Operating Systems

- 保护过程
  - UNIX结构



Views of Operating Systems

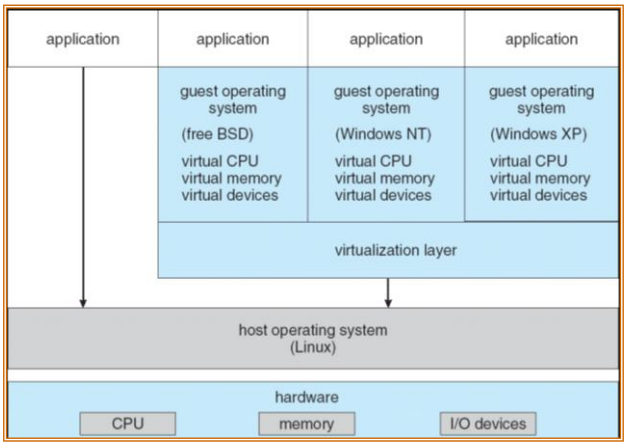
- 虚拟机
  - 虚拟机是一个程序，它完美地虚拟了硬件配置的每个细节，我们可以在上面运行操作系统（和许多应用程序）。
    - 例如，VMWare Fusion、Virtual Box、Parallels Desktop、Xen、Vagrant。
    - 历史悠久（60年代IBM操作系统开发）
    - 提供隔离
    - 完全绝缘，防止变化
    - 云中的常态
      - 服务器整合（SCON）
  - 两种类型的虚拟机（VM）
    - 进程VM：支持单个程序的执行；此功能通常由操作系统提供
    - System VM：支持整个操作系统及其应用程序的执行（例如，VMWare Fusion、Virtual box、Parallels Desktop、Xen）

Views of Operating Systems

- 虚拟机
  - 进程虚拟机
    - 编程简单性
      - 每个进程都认为它拥有所有内存/CPU时间。
      - 每个进程都认为它拥有所有设备。
      - 不同的设备似乎具有相同的高级接口。
      - 设备接口比原始硬件更强大
        - 位图显示 ⇒ 窗口系统
        - 以太网卡 ⇒ 可靠、有序的网络（TCP/IP）
    - 故障隔离
      - 进程无法直接影响其他进程
      - Bug不能使整个机器崩溃
    - 保护和可移植性
      - 例如，Java接口在许多平台上都是安全和稳定的

Views of Operating Systems

- 虚拟机
  - 系统虚拟机
    - 对操作系统开发有用
      - 崩溃的操作系统仅限于一个VM
      - 帮助在其他操作系统上进行程序测试



Views of Operating Systems

- **什么是操作系统，…实际上**
  - 很可能：
    - 内存管理
    - I/O管理
    - 处理器调度
  - 那么…
    - 文件系统？
    - 多任务/多道程序设计？
    - 用户界面？
    - 多媒体支持？
    - 通讯？
      - 电子邮件属于操作系统吗？
    - 互联网浏览器？

Views of Operating Systems

- **什么是操作系统，…实际上**
  - 操作系统提供虚拟机抽象来处理各种硬件
    - 操作系统通过提供标准服务简化了应用程序开发
  - 操作系统协调资源并保护用户彼此不受影响
    - 操作系统可以提供一系列的故障控制、容错和故障恢复

Evolution of Operating Systems

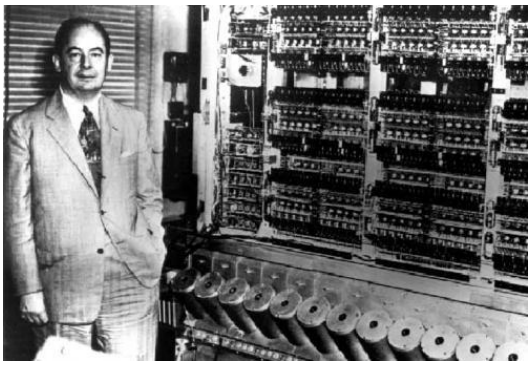
- **操作系统的演变**
  - 操作系统的发展直接依赖于计算机系统的发展以及用户如何使用它们。
  - 在时间轴上快速浏览过去几十年的计算系统。
    - 早期系统（1950年）
    - 简单批处理系统（1960年）
    - 多道程序批处理系统（1970年）多道批处理系统
    - 分时和实时系统（1970年）
    - 个人/台式计算机（1980年）
    - 多处理器系统（1980）
    - 网络化/分布式系统（1980年）
    - 基于网络的系统（1990年）

Evolution of Operating Systems

- **早期系统（1950年）**
  - 一些重要的时间点
    - 1945：宾夕法尼亚大学穆尔工程学院
    - 1949年：EDSAC和EDVAC
    - 1949年：BINAC——ENIAC的继任者
    - 1951年：雷明顿大学
    - 1952年：IBM 701
    - 1956年：中断
    - 1954-1957：FORTRAN语言

Evolution of Operating Systems

- **早期系统（1950年）**
  - *约翰·冯·诺依曼和埃尼亚克*

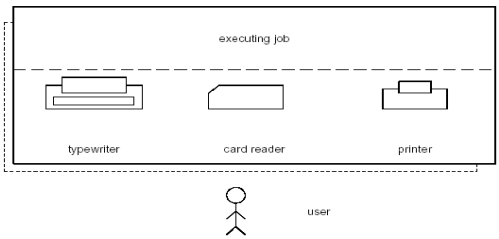


Evolution of Operating Systems

- **早期系统（1950年）**
  - 20世纪50年代末的操作系统
    - 到20世纪50年代末，操作系统得到了很好的改进，并开始支持以下用途：
      - 单流批处理
      - 使用通用、标准化的输入/输出例程进行设备访问
      - 计划转换功能，以减少启动新作业的开销
      - 作业异常终止后要清理的错误恢复
      - 允许用户指定作业定义和资源要求的作业控制语言

Evolution of Operating Systems

- **早期系统（1950年）**
  - 结构
    - 单用户系统
    - 程序员/用户作为操作员（开放式商店）
    - 从控制台运行的大型机器
    - 纸带或穿孔卡片



Evolution of Operating Systems

- **早期系统（1950年）**
  - 特点
    - 早期软件
      - 汇编程序
      - 通用子程序库（I/O、浮点等）
      - 设备驱动程序
      - 编译程序
      - 链接器。
    - 需要大量的设置时间
    - 极慢的I/O设备
    - 非常低的CPU利用率
    - 高安全

Evolution of Operating Systems

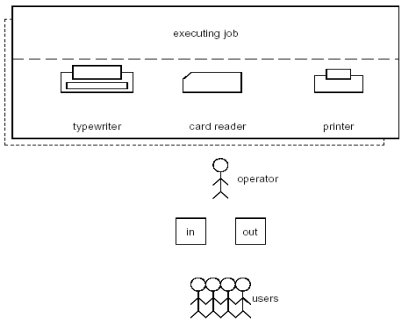
- 20世纪60年代的操作系统
  - 一些重要的时间点
    - 20世纪60年代：磁盘成为主流
    - 1961年：微型计算机的曙光
    - 1962年：麻省理工学院的兼容分时系统（CTSS）
    - 1963年：B5000系统的Burroughs主控制程序（MCP）
    - 1964年：IBM System/360
    - 1964年及以后：Multics
    - 1966年：小型计算机变得更便宜、功能更强大，且非常有用
    - 1967-1968：鼠标
    - 1969年：贝尔电话实验室的UNIX分时系统

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 使用高级语言、磁带
    - 作业按语言类型批处理在一起。
    - 雇佣一名操作员来执行加载作业、启动计算机和收集输出的重复任务（操作员驱动的车间）。
    - 用户直接检查内存或修补程序是不可行的。

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 使用高级语言、磁带。
    - 作业按语言类型批处理在一起。
    - 雇佣一名操作员来执行加载作业、启动计算机和收集输出的重复任务（操作员驱动的车间）。
    - 用户直接检查内存或修补程序是不可行的。



Evolution of Operating Systems

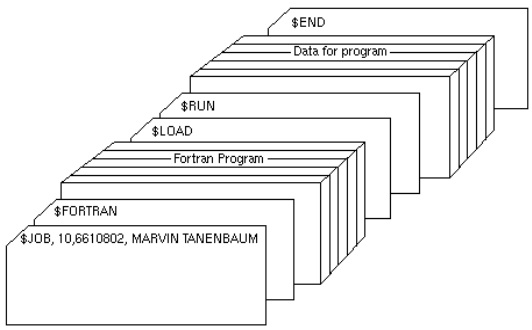
- 20世纪60年代的操作系统
  - 简单批处理系统
    - 操作
      - 用户向计算机操作员提交作业（写在卡片或磁带上）。
      - 计算机操作员在输入设备上放置一批多个作业。
      - 一个特殊的程序，监视器，管理批处理中每个程序的执行。
      - 在需要时加载监视器实用程序。
      - 常驻监控程序始终在主内存中，可供执行。

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 简单批处理系统的思想
      - 通过批处理类似作业减少设置时间
      - 用户程序和监控程序之间的交替执行
      - 依靠可用硬件有效地从内存的各个部分交替执行
      - 使用自动作业排序：当一个作业完成时，自动将控制权从一个作业转移到另一个作业
    - 问题
      - 监视器如何知道作业的性质（例如，Fortran或汇编）或要执行哪个程序？
      - 监视器如何区分：（a）不同的工作？（b）来自程序的数据？
    - 解决方案
      - 作业控制语言（JCL）和控制卡

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 示例：作业的卡片组



Evolution of Operating Systems

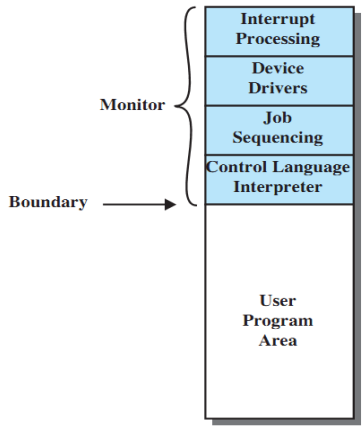
- 20世纪60年代的操作系统
  - 简单批处理系统
    - 作业控制语言（JCL）
      - JCL是一种向监视器提供指令的语言：
        - 使用什么编译器
        - 使用什么数据
        - 作业格式示例：
          - $FTN加载编译器并将控制权转移给它。
          - $LOAD加载目标代码（代替编译器）。
          - $RUN将控制转移到用户程序。
      - 每个读取指令（在用户程序中）导致读取一行输入。
      - 调用（OS）输入例程的原因：
        - 检查是否未读取JCL行
        - 用户程序完成时跳到下一行JCL

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 常驻监视器
      - *常驻监视器（作业序列器）是第一个基本的操作系统(常驻监控程序是第一种初具雏形的操作系统).*
        - 初始控制在监视器中。
        - 加载下一个程序并将控制权转移给它
        - 当作业完成时，控制权转移回监视器。
        - 控制权自动从一个作业转移到另一个作业，程序之间没有空闲时间

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 常驻监视器
      - *常驻监视器（作业序列器）是第一个基本的操作系统(常驻监控程序是第一种初具雏形的操作系统).*



Evolution of Operating Systems

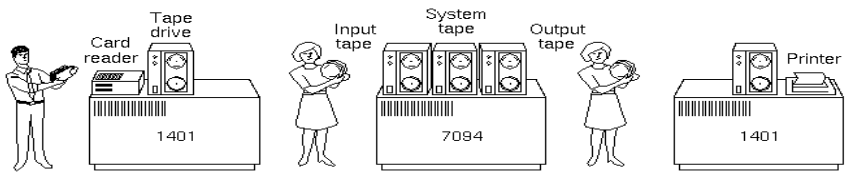
- 20世纪60年代的操作系统
  - 简单批处理系统
    - 常驻监视器的组成：
      - 控制语言解释器
        - 负责阅读和执行卡片上的指令
      - 装载机
        - 将系统程序和应用程序加载到内存中
      - 设备驱动程序
        - 了解每个系统I/O设备的特殊特性和属性

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 理想的硬件特性
      - 内存保护
        - 不允许用户程序更改包含监视器的内存区域
      - 特权指令
        - 只能由常驻监控器执行
        - 如果程序尝试这些指令，就会出现陷阱。
      - 中断
        - 提供向用户程序放弃控制和从用户程序重新获得控制的灵活性
        - 计时器中断可防止作业独占系统。

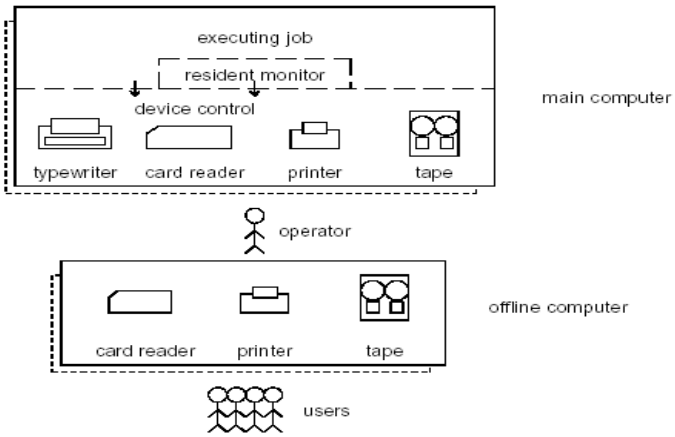
Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 脱机操作
      - 问题:
        - 读卡器和打印机速度慢（与磁带相比）。
        - I/O和CPU不能重叠。
      - 解决方案：离线操作（卫星计算机）
        - 通过将作业从磁带加载到内存中来加快计算速度，同时使用较小的机器离线进行读卡和行打印



Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 脱机操作
      - 主计算机和离线计算机

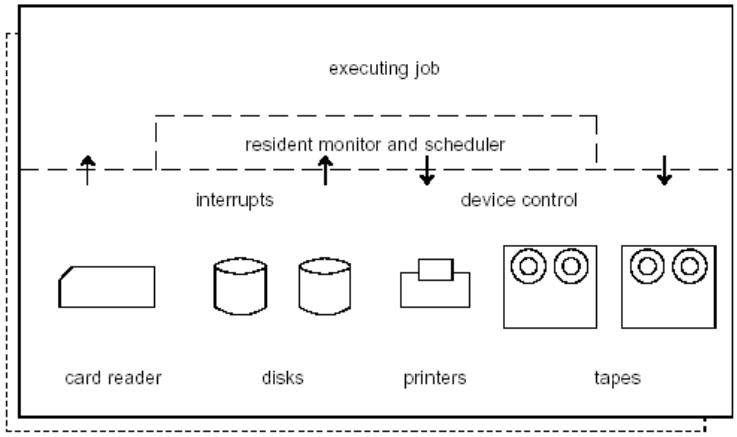


Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 假脱机
      - 问题:
        - 读卡器、行打印机和磁带机速度慢（与磁盘相比）。
        - I/O和CPU不能重叠。
      - 解决方案：假脱机
        - 将一个作业的I/O与另一个作业的计算重叠（使用双缓冲、DMA等）
        - 这项技术被称为SPOOL：外围设备联机并行操作，假脱机

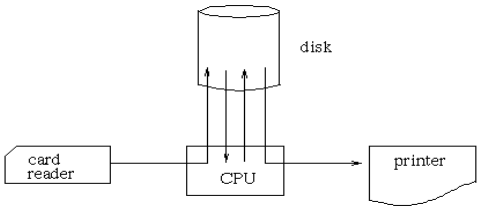
Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 假脱机
      - 假脱机系统组件



Evolution of Operating Systems

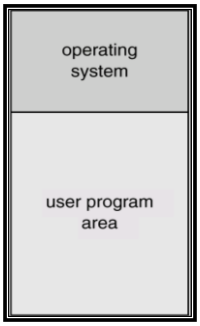
- 20世纪60年代的操作系统
  - 简单批处理系统
    - 假脱机
      - 在执行一个作业时，操作系统：
        - 将读卡器中的下一个作业读入磁盘上的存储区域（作业池）
        - 将上一个作业的打印输出从磁盘输出到打印机

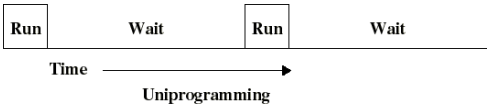


      - 作业池
        - 允许操作系统选择下一步运行哪个作业以提高CPU利用率的数据结构

Evolution of Operating Systems

- 20世纪60年代的操作系统
  - 简单批处理系统
    - 在单一编程系统中
      - I/O操作非常慢（与指令执行相比）。
      - 一个包含极少量I/O操作的程序将花费大部分时间等待I/O。
      - 因此：当内存中只有一个程序时，CPU使用率很低。





一个单一编程系统的内存布局

Evolution of Operating Systems

- 1970年以后的成就
  - 一些重要的时间点
    - 1971年：英特尔宣布推出微处理器
    - 1972年：IBM推出了虚拟机操作系统VM
    - 1973年：UNIX第四版出版
    - 1973年：以太网
    - 1974年：个人电脑时代开始了
    - 1974年：比尔·盖茨和保罗·艾伦为Altair写了BASIC
    - 1976年：苹果II
    - 1981年：IBM于8月12日推出IBM PC
    - 1983年：微软开始开发微软Windows
    - 1984年：苹果Macintosh问世
    - 1990年：微软Windows 3.0问世
    - 1991年：GNU/Linux
    - 1992年：第一个Windows病毒问世
    - 1993年：Windows NT
    - 2007年：iOS
    - 2008年：Android操作系统