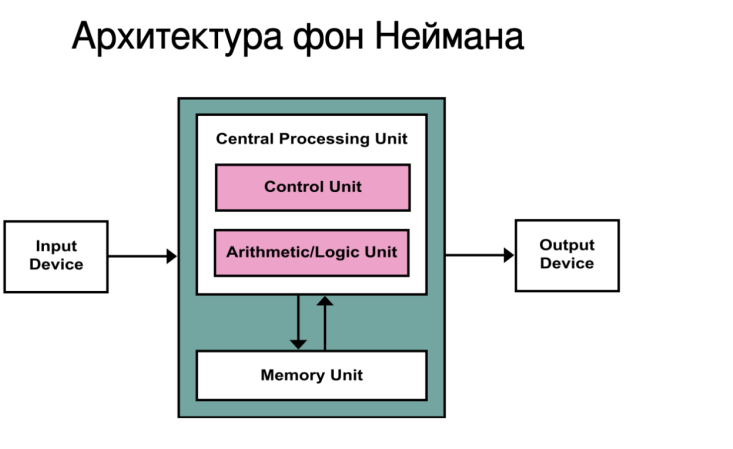
1. 计算机系统的体系结构。 Architecture Font-Nd 架构。 冯·诺依曼和哈弗的架构原理。 NUMA 和 UMA 架构。

架构主要有两种类型：冯·诺依曼和哈佛。

冯诺依曼存储器将数据和指令存储在同一存储器中



冯·诺依曼结构处理器具有以下几个特点：

* 必须有一个存储器；
* 必须有一个控制器；
* 必须有一个运算器，用于完成算术运算和逻辑运算；
* 必须有输入和输出设备，用于进行人机通信

冯诺依曼原理将待执行的程序转换成二进制形式的指令序列，并将这些指令存储在计算机的内存储器中：

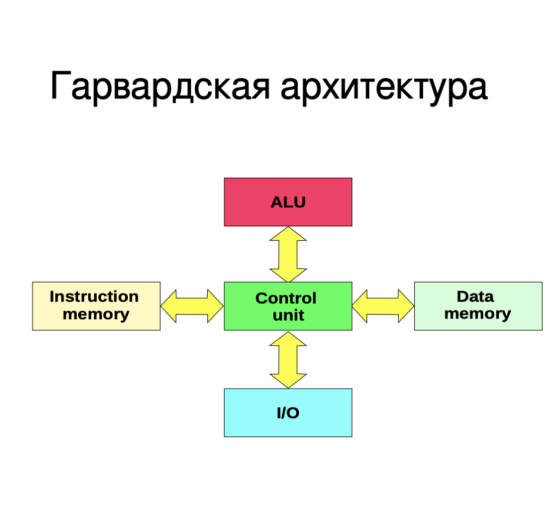
- 命令和数据存储在一起

- 寻址原理（每个存储单元都有自己的处理器可以访问的地址）

- 程序控制原理（计算以由一系列命令组成的程序的形式呈现）

- 二进制编码原理

哈佛存储器则具有单独的指令存储器和单独的数据存储存储器。



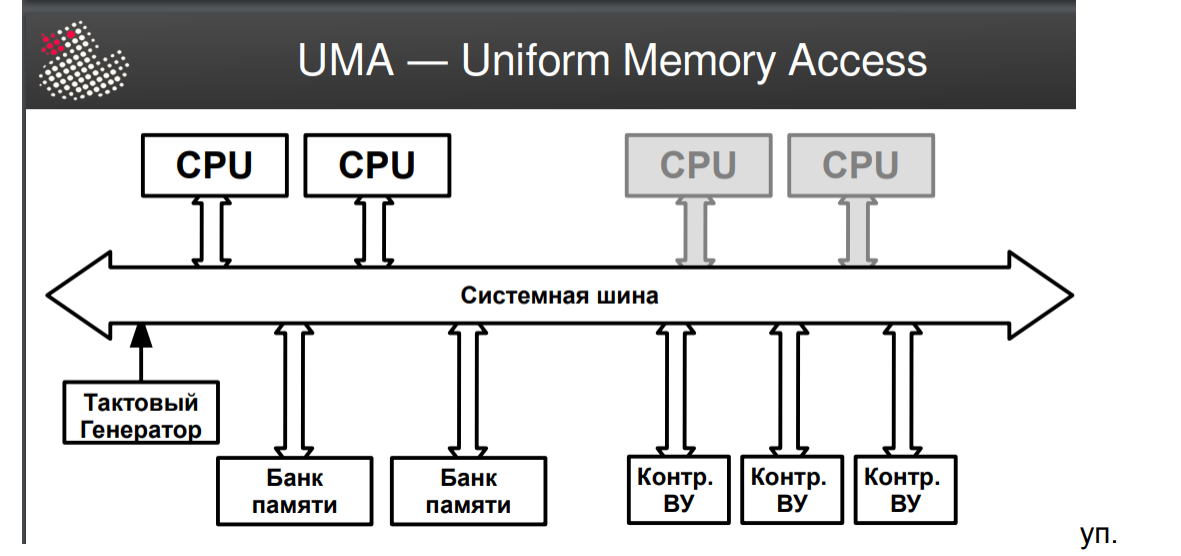
哈佛原理：

- 单独的数据，单独的指令

- 指令通道和数据通道也不同

根据处理器对内存的访问方式将共享存储器方式的计算机系统分为两大类，即UMA(Uniform Memory Access,统一内存访问)架构和NUMA(Non Uniform Memory Access,非统一内存访问)架构。

UMA（统一内存访问）：同一级别的所有设备以及所有设备对内存的访问权限相同，这意味着每个设备的访问时间相同。 延迟是相同的等等。



多个CPU对称工作，无主次或从属关系。各个CPU 共享相同的物理内存，每个CPU访问内存中的任何地址，所需要的时间是相同的

优点：

- 相等的内存访问时间

- 易于实施

缺点：

- 可扩展性差（随着进程数量的增加，会出现内存访问的冲突和竞争，从而降低系统速度）

- 仅最多两个处理器即可正常工作。

NUMA（非统一内存访问）：NUMA 模式是一种分布式存储器访问方式，处理器可以同时访问不同存储器地址，大幅度提高并行性，其访问时间随存储字的位置不同而变化。

NUMA模式下，处理器被划分成多个”节点”（node）， 每个节点被分配有的本地存储器空间。 所有节点中的处理器都可以访问全部的系统物理存储器，但是访问本节点内的存储器所需要的时间，比访问某些远程节点内的存储器所花的时间要少得多。

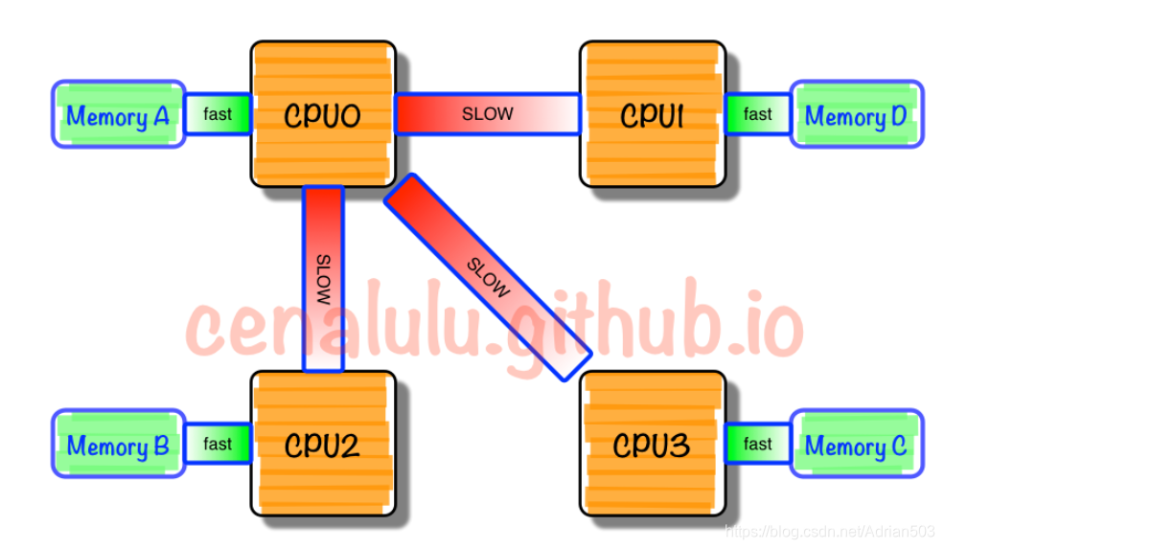
- 所有系统资源都位于单独的主板上

- 有本地内存（距离处理器“更近”）

- 共享内存和 I/O 设备可通过协调组件交互的开关进行访问

- 用于服务器

- 地址空间对所有进程都是通用的！！！



优点：

- 能够在系统运行时更换几乎所有组件

- 良好的可扩展性

- 可靠性和容错性

缺点：

- 对操作系统要求较高（必须支持所有这些功能）

- 考虑访问时间和数据在内存中的位置很重要（因为访问时间取决于数据所在的位置）

1. 结构处理器、存储器、指令流水线。（Общая организация процессора, памяти, организация вычислений.）

结构处理器：

二级缓存（L2$）：所有核心共享的缓存

一级Cache（L1$）：分为Data Cache（L1 D$）和Instruction Cache（L1 I$），都有TLB缓冲区，这有助于并行读取数据和指令

转换关联缓冲区（TLB）：存储有助于快速查找高速缓存内部的条目或了解它们是否在本级高速缓存中的条目（加快虚拟内存地址到物理内存地址的转换）

超线程（两个寄存器块），超线程技术的本质：处理器有两组寄存器，这可以让你快速切换两个线程的上下文。 对于操作系统来说，这看起来就像两个逻辑处理器。

分支预测单元：当满足条件分支中的条件时，加载的命令结果是不必要的，这意味着管道将停止。为了避免这种情况，创建了分支预测器，这使得清楚哪些命令加载。

指令流水线（讲座中提到错误，最好阅读wiki或这里）

- 由于多个 ALU 的并行执行和取指令周期，允许操作队列的并行化

- 输送机

- 从命令获取设备接收命令，对其进行解码并将其发送到操作调度程序 - 调度程序

- 决定是否并行运行传入命令以及是否完全运行命令，具体取决于数据的可用性

-分支前置单元

- 转换预测器，优化指令的加载（使传送带不会停止）

需要写缓冲区，因为写入内存的速度可能跟不上命令执行的进度

有普通的 alu，也有特定的（用于浮点、用于向量运算等）。

内存：

- 虚拟内存是一种组织内存的方式，每个进程都有自己的地址空间，认为有自己独立的内存。

- 使用转换表将虚拟内存地址转换为物理内存地址。

- 访问信息的单位是字节。 数据被组合成页（大小通常为 4 KB）。

- 大页面 – 包含大量数据的页面（例如 1 MB）

内存金字塔：



计算机内部的内存是分层组织的。 最快和最昂贵的内存位于顶部，最大容量位于底部。 以便减少对较慢内存的访问。

Тд（访问时间）——第一个内存访问请求和第一个数据之间的间隔

控制类型：

- 编译器：使用编译开关（优化级别）控制

- 硬件：使用关联存储器（TLB）中的电路进行搜索

- 软件

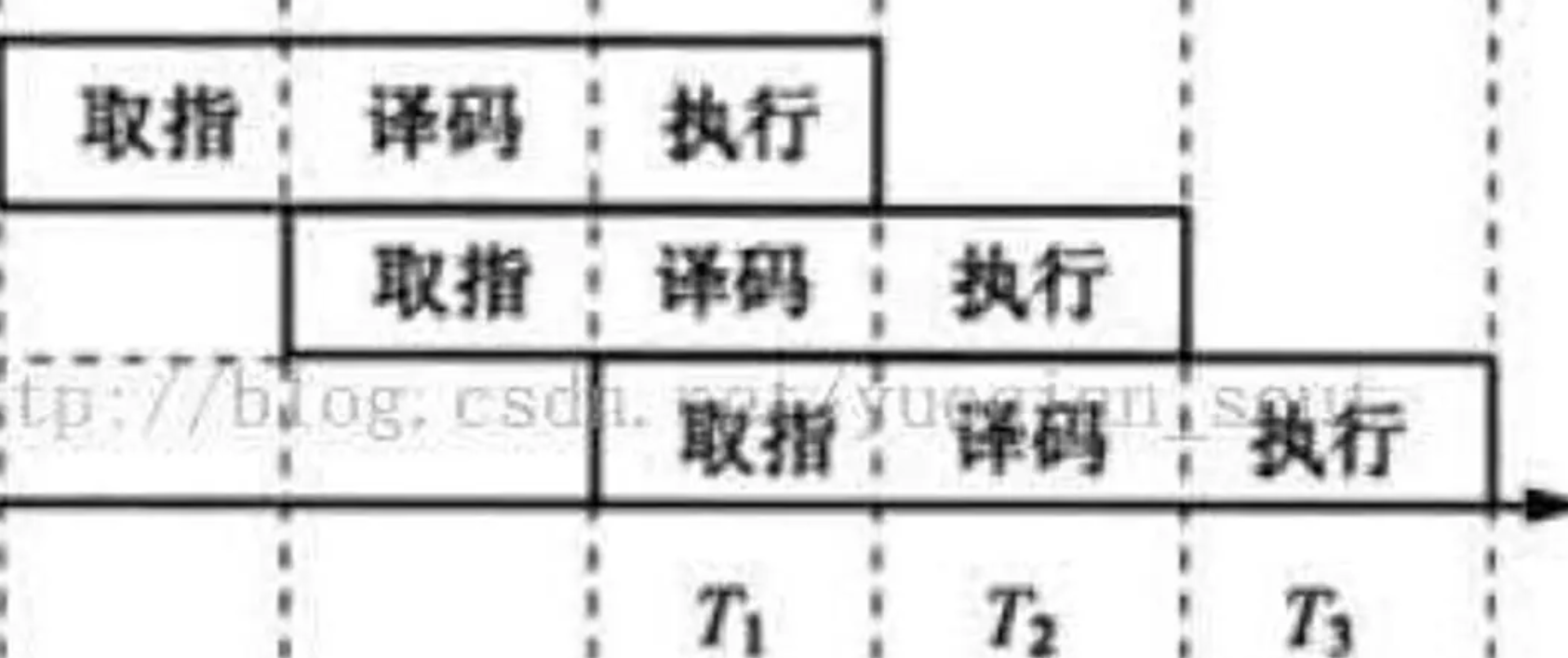
TLB（translation lookaside buffer）原理：

当cpu要访问一个虚拟地址/线性地址时，CPU会首先在TLB中查找。如果是表中没有相应的表项，称为TLB miss，需要通过访问慢速RAM中的页表计算出相应的物理地址。同时，物理地址被存放在一个TLB表项中，以后对同一线性地址的访问，直接从TLB表项中获取物理地址即可，称为TLB hit。

想像一下x86\_32架构下没有TLB的存在时的情况，对线性地址的访问需要3次RAM的访问。如果有TLB存在，并且TLB hit，那么只需要一次RAM访问即可。

指令流水线организация вычислений（英语：Instruction pipeline）：是为了让计算机和其它数字电子设备能够加速指令的通过速度（单位时间内被执行的指令数量）而设计的技术。即在前一个命令完成执行之前，下一个命令开始执行。

指令流水线是为提高处理器执行指令的效率，把一条指令的操作分成多个细小的步骤，每个步骤由专门的电路完成的方式。



分为五个阶段：

1. 读取指令
2. 指令解码与读取[寄存器](https://zh.wikipedia.org/wiki/%E6%9A%AB%E5%AD%98%E5%99%A8)
3. 执行
4. 存储器访问
5. 写回寄存器
6. 中断的结构、中断的类型、中断控制器。（Организация прерываний, типы прерываний, контроллер прерываний.）

中断：（英语：Interrupt），又称插断，在计算机科学中是指处理器接收到来自硬件或软件的信号，通知处理器发生需要立即注意的事件。 中断通知处理器发生了高优先级事件，需要中断处理器正在执行的当前代码。

为什么需中断：

CPU 如果要从其他地方获取数据，不可能时时盯着啥时候会有数据。当收到数据时，给CPU 发送一个中断请求信号，请求占用CPU，进行数据读取的操作。

如此，CPU 就没必要傻傻的一直等着数据准备就绪，在数据到来之前，CPU可以着手处理其他事情。

中断是指CPU对系统发生某个时间做出的一种反应。（是外部设备向处理器发起的请求事件）中断发生后，CPU 不会立即执行响应，而是在当前指令执行完以后转入“ 中断处理流程 ”。此时 CPU 会跳转到中断处理程序去处理这个中断，处理完毕后再回到原来位置的下一个位置

● 可能是由程序本身或来自IO 设备的信号引起的。

● 较高优先级的中断可以被较低优先级的中断中断。

● 在命令周期结束时执行

中断控制器（Programmable Interrupt Controller，PIC）——负责顺序处理来自不同设备的中断请求的能力的芯片或内置处理器单元。

中断控制器相当于一个代理，我们外部设备产生的中断事件不会直接通过INTR总线进入CPU，而是先发送给中断控制器，中断控制器再转交给CPU。

为什么需要中断控制器：减少CPU的负载，让CPU更加专注于计算。

中断类型

中断可分为同步（synchronous）中断和异步（asynchronous）中断：

同步（内部）中断是当指令执行时由 CPU 控制单元产生，之所以称为同步，是因为只有在一条指令执行完毕后 CPU 才会发出中断，而不是发生在代码指令执行期间，比如系统调用。

异步（外部）中断是指由其他硬件设备依照 CPU 时钟信号随机产生，即意味着中断能够在指令之间发生，例如键盘中断。

同步中断又称为异常（exception），异步中断则被称为中断（interrupt）。

我们通常讲的中断指的都是异步中断。

1.中断可分为可屏蔽中断（Maskable interrupt）和非屏蔽中断（Nomaskable interrupt）。  
2.异常可分为故障（fault）、陷阱（trap）、终止（abort）三类。

1. 典型操作系统功能。 操作系统接口。 操作系统替代计算机操作员的操作（Типичные функции операционной системы. Интерфейсы ОС. Работа ОС как замена оператора ЭВМ）

操作系统功能：

主要功能（简单操作系统）：

* 将应用程序加载到 RAM 中并执行它们；
* 外围设备（输入/输出设备）的标准化访问；
* RAM管理（进程间分配、虚拟内存）；
* 控制对非易失性介质（如硬盘、CD等）上数据的访问，通常使用文件系统；
* 用户界面;
* 网络操作、协议栈支持

附加功能：

* 并行或伪并行执行任务（多任务）；
* 进程间交互：数据交换、相互同步；
* 保护系统本身以及用户数据和程序免受用户或应用程序的恶意操作；
* 区分访问权限和多用户操作模式（身份验证、授权）。

简而言之，这就是程序对资源访问的管理，以及程序开发的简化

RAM：随机存取存储器（英语：Random Access Memory，缩写：RAM），也叫主存，是与CPU直接交换数据的内部存储器。它可以随时读写（刷新时除外），而且速度很快，通常作为操作系统或其他正在运行中的程序的临时数据存储介质。RAM工作时可以随时从任何一个指定的地址写入（存入）或读出（取出）信息。它与ROM的最大区别是数据的易失性，即一旦断电所存储的数据将随之丢失。RAM在计算机和数字系统中用来暂时存储程序、数据和中间结果。

操作系统接口：包括指令系统结构（Instruction Set Architecture，简称ISA）、应用程序编程接口（Application Programming Interface，简称API）、应用程序二进制接口（Application Binary Interface，简称ABI）

API：

API（Application Programming Interface，简称API）是应用程序的高级语言编程接口，在编写程序的源代码时使用。常见的API包括C语言、Fortran语言、Java语言、Javascript语言、OpenGL图形编程接口等。使用一种API编写的应用程序经重新编译后可以在支持该API的不同计算机上运行。API是一组定义的规则，使不同的应用程序能够相互通信。

API是对接使用者和后台服务的程序接口

ISA：

指令集架构（Instruction Set Architecture，ISA）也称指令集或者指令集体系，是计算机体系结构中与程序设计有关的部分，包含基本数据类型，指令集，寄存器，寻址模式，存储体系，中断，异常处理以及外部IO。

ABI：

ABI（Application Binary Interface） 应用程序二进制接口，描述了应用程序和操作系统之间，一个应用和它的库之间，或者应用的组成部分之间的低接口。ABI 不同于 API，

ABI 允许编译好的目标代码在使用兼容 ABI 的系统中无需改动就能运行

当我们的应用程序引用了一个以二进制形式发布的库时，在源代码层面，我们使用了这个库的 API；而在编译、链接之后，在运行时，我们的应用程序通过 ABI 在与这个库通信。从这个角度看来，ABI 只是 API 的底层实现，所以多数时候我们才不需要去关心这个问题。

在操作员之前，科学家=程序员自己输入程序，启动它并离开一段时间，直到它完成它的工作。 有一个时间表，用户可以按照时间表来携带他们的程序。

然后计算机变得更快，维持这样的时间表变得困难，出现了操作员，向他们提供程序，并且随着计算机变得免费，他们自己启动了新程序

1. 批量处理。 系统监视器。（Пакетная обработка. Системный монитор）

批处理任务（英語：batch processing），又称批次处理任务，是指在计算机上无须人工干预而执行系列程序的作业。计算机用来周期性地完成大量重复数据作业的方法

批处理任务无须人工交互，所有的输入数据预先设置于程序或命令行参数中。这是不同于需要用户输入数据的交互程序的概念。

计算机用来周期性地完成大量重复数据作业的方法。某些数据处理任务（如备份、筛选和排序）可能需要大量计算，而且在单个数据事务上运行效率很低。相反，数据系统通常可以在计算资源更普遍可用的非高峰时间批量处理这些任务，例如一天结束时或夜间。例如，考虑一个全天接收订单的电子商务系统。系统可能会在每天结束时收集所有订单，并用一个批处理与订单履行团队共享，而不是在发生时处理每个订单。

优点

批处理有如下优点：

* 允许多用户共享计算机资源
* 可以把作业处理转移到计算机资源不太繁忙的时段
* 避免计算资源闲置，而且无须时刻有人工监视和干预
* 在昂贵的高端计算机上，使昂贵的资源保持高使用率，以减低平均开销

比如：在公共交通中，公共小巴时常都是用批处理的方法运输乘客的。

系统监视器（system monitor）是一种用于监视计算机系统中资源与性能情况的硬件或软件组件。 它们通常用于监控系统资源的使用情况，最基本的信息如CPU使用率与频率、空闲内存量。有时它们还用来显示一个或多个硬盘驱动器的可用空间，CPU和其他重要组件的温度，以及系统的IP地址、当前的网络传输速度等信息。

1. 分析全系统效率作为多任务处理的前提。 多任务处理是提高系统效率的一种方式。 分时系统

通过评估不同任务和功能在系统内如何相互作用，分析全系统效率在多任务处理中发挥着重要作用。 系统范围的效率衡量系统在优化利用资源的同时实现其目标的情况。

多任务处理又是指系统或组织同时或按顺序执行多项任务而不损失质量或效率的能力。 成功的多任务处理要求系统中的每项任务都以最高效率完成，而不干扰其他任务的完成。

因此，分析整个系统的效率可以让您识别系统中可能对其多任务处理能力产生负面影响的瓶颈或问题区域。 纠正这些问题可以提高系统的整体效率及其同时成功执行多个任务的能力。

多任务处理是系统同时或顺序处理多个任务的能力。 它是提高系统效率的关键机制，原因如下：

* 最大化资源利用率：多任务系统可以有效地利用计算资源，例如CPU时间、内存和网络资源。 当一个任务正在等待某些操作完成时，处理器可以切换到执行另一个任务，从而充分利用可用资源。
* 提高性能：系统可以继续处理其他任务，而不是等待一项任务完成。
* 提高响应能力：多任务系统可以并行处理用户请求或其他外部事件，从而与用户或其他系统进行更快、响应更灵敏的交互。
* 更好地利用时间：多任务处理可以有效地利用时间，使系统即使面对波动的负载或意外事件也能连续运行。
* 提高可靠性：在多任务系统中，错误隔离是可能的，因为一项任务的失败并不一定会使系统完全停止。

因此，多任务处理通过更有效地利用资源、提高性能和响应能力以及提高系统可靠性，在提高系统效率方面发挥着关键作用。

有很多用户，每个人都需要一台计算机，他们注意到处理器闲置了很多时间，因为程序需要获取数据，例如从磁盘获取数据。 我们决定确保此时处理器上正在运行另一个程序，该程序不会等待任何事情。 从概念上讲，操作系统当时已经以分时系统的形式出现，但它本质上只是一个骨架

1. 流程、现代流程的问题。 流程执行计划和资源管理。（Процессы, проблемы современных процессов. Планирование выполнения процессов и управление ресурсами.）

进程是程序执行期间的一个实例。

进程是操作系统活动的一个单元，其中有顺序操作、当前状态和一组关联的资源。 这个定义是从操作系统的角度来看的。

流程结构

·可执行程序（进程就是程序）

· 一组执行线程（程序内包含许多执行线程，这本质上是处理器消耗的一个单位）

内核相关的结构（当在操作系统内部创建进程时，内核必须构建结构，即包含进程消耗的资源描述的内存区域）

· 地址空间是任何计算机系统对象（内存单元、磁盘扇区、网络节点等）所有有效地址的集合，可用于在特定进程中访问它们

· 执行上下文（例如寄存器）（当我们从另一个进程返回到这个进程时需要恢复进程的状态）

· 安全上下文（linux中各种suid/sgid）

· 资源（文件等）+结构体、关联结构体

· 动态库现代流程的问题

进程内存保护（非确定性程序行为，即不可预测）

· 互锁（死锁、饥饿、活锁）

· 同步问题

·资源访问互斥

规划流程执行和资源管理

● 平等（用户应平等地获得资源）

● 响应差异化（需要减少响应时间）

● 全系统效率

● 进程、磁盘等的调度程序（不同类型的调度（分时、交互式、实时、系统、公平共享（例如每个与优先级〜共享数量成比例）、固定..）） I/O 调度

1. 内存管理，虚拟内存。 信息保护和操作系统安全。（Управление памятью, виртуальная память. Защита информации и безопасность ОС.）

内存管理

1.进程隔离（监控没有进程能够改变别人的内存）

2.管理内存的分配和释放（程序必须动态连接到内存——也就是说，它们还必须能够将控制权转移给它）

3. 模块支持（定义、创建、销毁和调整大小的能力）

4.安全和访问控制（操作系统必须监控不同用户如何访问不同的内存区域）

5. 长期存储——应用程序需要一种在计算机关闭后仍能保留信息的方法。

6. 分页（英语 paging；术语“swapping”来自“swap”，有时使用/swɔp/）是虚拟内存机制之一，其中各个内存片段从 RAM 移动到辅助存储（硬盘驱动器或其他外部驱动器），从而释放内存RAM 用于加载其他活动内存片段（现代计算机中的此类片段是内存页）。 在 Windows 中：页面文件 sys.

虚拟内存由MMU和TLB维护，为每个进程和内核提供单独的地址空间。

虚拟内存是一种管理计算机内存的方法，它允许您通过在主内存和辅助存储之间自动移动部分程序来运行需要比计算机上可用内存更多的 RAM 的程序 + 存在非分页页面。

信息保护和操作系统安全。 关于安全的一切：法规。 事实并非它会起作用，但有一些程序（协议）如果遵循，可以提供帮助。

● 系统访问（防止未经授权的访问）

● 保密性（未经授权不得访问）

● 数据完整性（保护数据免遭未经授权和不完整的更改）

● 身份验证和授权（身份验证是一种身份验证过程，例如通过将用户输入的密码与数据库中存储的密码进行比较来验证用户的真实性。授权是向特定的人或一组人授予权限，以执行特定的操作。执行某些操作。）

1. 操作系统内核的结构。 单片内核、动态加载内核和微内核的体系结构。（Структура ядра операционной системы. Архитектуры монолитного ядра, ядра динамически загружаемыми модулями и микроядра.）



有一个用户模式，有一个内核模式，有一个调用接口，数据在用户进程和系统本身之间传输。

1) 单片核心

单片核心（监视器的模拟）； 有一个实用程序，信息被加载到内存中并在那里工作； 内核无法更改。 现在，除了嵌入式系统之外，它的纯粹形式并未在任何地方使用。

2) 模块化内核是对单片操作系统内核体系结构的现代改进修改。 与“经典”单片内核不同，模块化内核通常在计算机硬件的组成发生变化时不需要完全重新编译内核。 相反，模块化内核提供了一些机制来加载支持特定硬件（例如驱动程序）的内核模块。 在这种情况下，模块的加载可以是动态的（在运行的系统中“即时”执行，无需重新启动操作系统）或静态的（在重新配置系统以加载某些模块后重新启动操作系统时执行）。 Linux中加载和卸载的命令：rsmod、modprob、rmmod。 开关机方便，不会使整机超载。

3）微内核 内核执行基本功能，其他功能以服务的形式实现，并有各自的安全级别。 由于频繁的上下文切换，它运行缓慢。 （例如：Windows NT）

1. 执行线程、多线程、多线程模型。（Потоки исполнения, многопоточность, модели многопоточности.）

从操作系统角度看流程：

1）进程（资源）的所有权

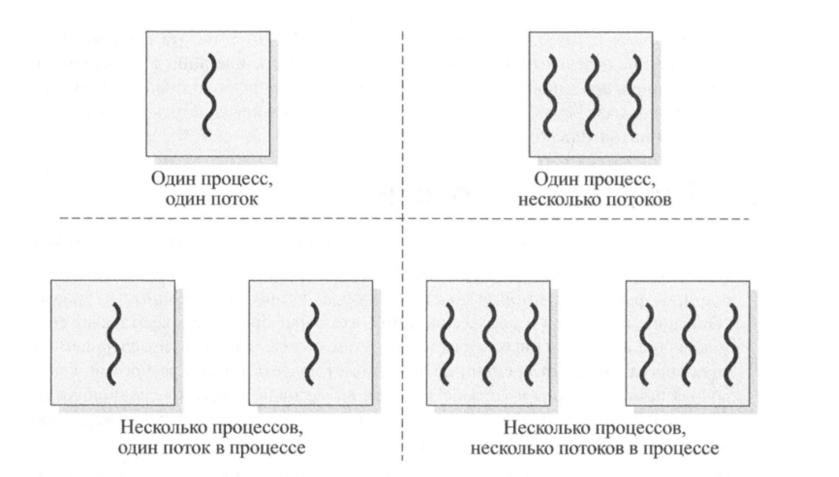
2）计划/执行（只是我们的线程）

线程（thread）是一组独立的寄存器，是操作系统的调度和执行的单位（实际上，它是进程执行的一个元素）。

多线程（英语：multithreading），是指从软件或者硬件上实现多个线程并发执行的技术。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。

单个进程中执行的多个线程称为线程，是 CPU 使用的基本单位，由线程 ID、计数器、寄存器和堆栈组成。

多线程模型：



1. 对称和非对称多处理。（Симметричная и асимметричная многопроцессорная обработка. ）

对称多处理（Symmetric multiprocessing，SMP）是一种多处理器计算机的体系结构，其中两个或多个性能相当的相同处理器统一连接到共享内存（和外围设备）并执行相同的功能（这就是为什么，事实上，该系统称为对称系统）。 很简单，因为一切都是一样的，但我们必须对抗同一区域的各种锁定和占领。 每个进程都自行管理。而非对称多处理（Asymmetric multiprocessing，ASMP）则包括多个处理器，每个处理器都有自己的本地内存，但可以通过特定的通信方式进行数据共享。

在SMP体系结构中，每个处理器可以同时运行不同的进程或线程，并且在共享内存中进行通信和同步，因此对于每个处理器来说，所有任务具有相同的处理优先级，且处理器间具有相同的系统异常和中断响应时间。SMP可以更好地发挥多处理器技术的优势，例如增加系统处理性能、提高系统可靠性和提高吞吐量。

相比之下，AMP系统的处理器通常采用不同的操作系统和应用程序，并且没有共享的内存区域，因此处理器之间的通信和同步需要使用特定的通信方式（例如消息传递）。AMP适用于一些特殊应用领域，例如多功能嵌入式系统或高可靠性系统。

总的来说，SMP是面向通用计算机应用的多处理器架构，而AMP则更专注于特定应用领域的多处理器架构。

1. 虚拟化。 虚拟化的类型（Виртуализация. Типы виртуализации）

虚拟化是在一台计算机上创建某些资源的虚拟（逻辑）版本。 那些。 这是一种在另一个程序控制下运行的解释器。

一般来说，虚拟机是模拟真实机器行为的资源集合。 同时，宿主机和来宾平台运行的进程是隔离的

应用程序容器 - 更改根目录，以便从那里执行所有内容，您可以只放置必要的库，而不要放置多余的东西。

（隔离应用程序的目的是使其“不知道”与它同时在同一操作系统上运行的其他程序，以便它“认为”它是唯一正在运行的程序。）示例：在沙箱中运行

对于每个应用程序，在其启动期间，都会创建一个所谓的运行时环境。

粗略地说，虚拟化是对硬件环境（一台机器上的多个操作系统）的模拟，容器化是在一个操作系统内分配一个隔离的环境（容器）。 所有容器都使用一个操作系统内核；在虚拟化中，每个环境都有自己的内核。

云技术允许您从您的设备使用远程设备的资源。

1. 操作系统故障和容错。 操作系统失败的原因及处理方法。（Сбои и отказоустойчивость ОС. Причины появления отказов в ОС и способы борьбы с ними..）

操作系统（OS）的故障和容错是重要的方面，特别是在确保计算机系统的可靠性和稳定性方面。 以下是几个关键点：

1. \*\*崩溃及其原因：\*\* 操作系统崩溃可能由多种原因引起，包括软件错误、硬件故障、用户错误或断电等外部因素。 重要的是能够识别和分析故障原因以防止将来再次发生故障。

2. \*\*防止故障的措施：\*\* 使用各种方法和技术来提高操作系统的容错能力。 这包括开发可靠的软件、使用故障检测和恢复机制以及实施电源备份和硬件冗余等硬件保护。

3. \*\*检测故障并从故障中恢复的方法：\*\* 操作系统通常包括检测故障并自动从故障中恢复的机制。 这可能包括数据备份机制、文件完整性监控系统、系统健康监控以及关键故障的自动恢复。

4. \*\*备份与恢复：\*\* 数据备份是保证操作系统容错的重要方面。 这使您可以从重大故障或攻击中恢复系统，从而最大限度地减少数据丢失和恢复时间。

5. \*\*测试和更新：\*\* 定期测试和更新操作系统可以让您识别和修复漏洞、提高性能并提高容错能力。 更新可能包括错误修复、安全改进和新功能。

一般来说，操作系统的故障和恢复需要一种全面的方法，包括软件和硬件措施以及数据备份和恢复计划。 这使您即使在不利的条件或突然出现故障的情况下也能确保计算机系统稳定可靠的运行。

失败原因：

● 由于组件故障而导致硬件或软件状态不正确（可能是由于月波，也可能是永久性的，则程序已在设备中）

● 操作员错误（输入错误：）

● 物理环境干扰（电缆影响电缆）

● 设计、编程、数据结构等方面的错误。

可以是：永久的、临时的（一次性或周期性）

处理操作系统故障的方法有以下几种：

1. \*\*定期更新和补丁：\*\* 操作系统及其组件的定期更新允许您修复漏洞和错误，从而降低发生故障的可能性。

2. \*\*监控和资源管理：\*\* 监控操作系统资源（内存、处理器、磁盘空间）的使用情况可以让您发现问题并防止系统过载。

3. \*\*备份和恢复：\*\* 定期创建数据和系统的备份副本可以让您在出现故障后快速恢复系统。

4. \*\*安装防病毒软件：\*\* 防范恶意软件和恶意攻击可降低恶意软件导致故障的风险。

5. \*\*测试和调试：\*\* 定期测试操作系统是否有错误并调试代码有助于在实际运行中防止故障发生。

6. \*\*硬件组件的冗余：\*\* 使用重复或冗余的硬件组件（例如，硬盘驱动器的 RAID）可以让您在其中一个组件出现故障时维持系统功能。

这些方法的结合有助于降低操作系统出现故障的风险，保证操作系统稳定可靠的运行。

1. 可靠性。 平均恢复时间。 可用性和停机时间。（Надежность. Среднее время восстановления. Коэффициент доступности и время простоя.）

操作系统 (OS) 的可靠性决定了其在一定时间内无故障或最少故障次数运行的能力。 可靠性通常通过平均恢复时间 (MTTR)、可用性比率和停机时间等指标来衡量。

1. \*\*平均恢复时间 (MTTR)：\*\* 这是发生故障后恢复系统所需的平均时间。 MTTR包括故障检测时间、诊断时间、恢复时间和系统健康检查时间。 MTTR越低，系统在发生故障后恢复的速度就越快，这有助于提高其可靠性。

2. \*\*可用性比率：\*\* 可用性比率决定系统可供使用的时间百分比。 它的计算方法是系统的运行时间与系统必须可用的总时间的比率。 以百分比表示。 可用性系数高表明系统可靠性高。

3. \*\*停机时间：\*\* 停机时间是指系统因崩溃或其他问题而无法使用的总时间。 这包括系统完全不可用的时间，以及系统由于故障而以降低的容量或性能运行的时间。 目标是最大限度地减少停机时间，以确保系统不间断运行。

为了提高系统可靠性并减少停机时间，采用了多种方法和技术，例如数据备份、硬件冗余、故障检测和恢复机制以及冗余设备和通信的使用。 测试系统故障并定期更新也是保持其可靠性的重要步骤。

1. 冗余和容错。（Резервирование и отказоустойчивость.）

冗余方法

● 物理冗余（组件、服务器）- 对同一事物使用两个设备/控制器。

● 时间冗余（重复计算）

● 信息冗余（ECC、RAID - 校验和）

容错是指系统或组件在存在硬件或软件错误的情况下继续正常运行的能力。 通常，容错涉及一定程度的冗余。 容错旨在提高系统的可靠性。通常，提高容错能力（从而提高可靠性）会产生一定的成本，要么是经济成本，要么是性能下降（或两者兼而有之）。 因此，确定所需的容错程度必须考虑到底什么是关键资源。

提高操作系统容错能力的方法

● 进程隔离（虚拟空间现在很方便，但是内核里还是有一个空间）。

● 允许并发锁。

● 虚拟化- 完全隔离执行（您自己的内核）。

● 还原点（关键文件的副本）和回滚。

1. GNU/Linux 操作系统的历史和发展。 单一 UNIX 规范和 POSIX。（История и развитие ОС GNU/Linux. Single UNIX Specification и POSIX）

GNU/Linux 操作系统的历史和发展与自由开源软件的复杂历史交织在一起。 以下是要点的快速概述：

1. \*\*开始：\*\* GNU/Linux 操作系统是由 Linux 内核开发者 Linus Torvalds 和 GNU 项目创始人 Richard Stallman 在 20 世纪 90 年代初创建的，Richard Stallman 为用户实用程序和应用程序提供了基础设施。 Linux 内核与 GNU 实用程序的结合产生了 GNU/Linux 操作系统。

2. \*\*发行版：\*\* GNU/Linux因其开放性、灵活性和自由性而迅速流行起来。 世界各地的开发者社区为该操作系统的开发和改进做出了重大贡献。

3. \*\*发行版：\*\* 随着时间的推移，出现了许多 GNU/Linux 发行版，每个发行版都提供自己独特的功能、软件包和专用工具。

4. \*\*单一 UNIX 规范 (SUS)：\*\* 单一 UNIX 规范是一组定义 UNIX 兼容操作系统的接口和功能的标准。 该标准提供了不同类UNIX系统之间的兼容性，并确保接口和功能的一致性。

5. \*\*POSIX：\*\* POSIX（便携式操作系统接口）是由 IEEE（电气和电子工程师协会）开发的一组标准，定义了 UNIX 兼容操作系统之间的互操作性接口。 POSIX 促进了不同类 UNIX 系统之间的应用程序可移植性，并确保 API 和功能的一致性。

GNU/Linux 非常适合单一 UNIX 规范和 POSIX，因为它受到 UNIX 的启发并努力与这些标准兼容。 这为类 UNIX 系统开发的应用程序和工具提供了高度的兼容性和可移植性。

SUS 是设定 UNIX 系统要求的规范。 使用此文档，您可以确定操作系统是否属于 Unix POSIX - 负责 UNIX 操作系统接口的标准（ISA - 指令集架构、ABI - 应用程序二进制接口、API - 应用程序接口）。

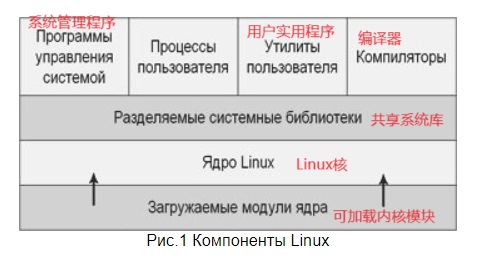
类 UNIX - 不符合标准的操作系统

1. 发行套件的概念，Linux发行套件（Понятие дистрибутива, дистрибутивы Linux.）

发行版是一个内核+一组用于应用任务的软件；每个发行版都可以有自己的一套。 该套件包括“系统”软件：包管理器、文件管理器、图形外壳和一组应用程序（文本编辑器、文字处理器、浏览器、音频/视频播放器等）（但并非所有这些都一定是包括，例如，图形 shell 并不是到处都使用，有些发行版几乎是纯内核，例如 Arch）有些发行版是由公司创建的（RedHat、Ubuntu、Suse）：虽然这些是开源项目，但公司出售提供支持（更新、故障排除）的发行版许多发行版都受到使用它们并对它们的开发感兴趣的社区的支持（Arch、Debian 及其许多后代）由于代码是开放的，许多发行版以旧发行版为基础并进行一些更改在那里，创建一个新的 - 这解释了 Linux 发行版的庞大树状结构，但其中大多数都是 Debian 和 RedHat 的后代。 Android 使用 Linux 内核，但不使用 GNU 内核，因此通常不被归类为 Linux 发行版。

在 Linux 中，内核通常取自 Linus Torvalds（某些版本），并包含环境（库、实用程序）。 发行版有不同的包和更新管理器。 此外，还有一个图形子系统，有一些应用程序在 Linux 上工作/不工作:)。 分发支持可以用金钱购买。

1. Linux的体系结构和主要子系统。 Linux 内核图。（Архитектура и основные подсистемы Linux. Linux Kernel Map.）



主要 Unix/Linux 子系统：

● 进程和调度程序。 – 创建、管理和规划流程。

● 虚拟内存。 – 为进程分配和管理虚拟内存。

● 物理内存。 – 管理页框池并为虚拟内存分配页面。 （物理内存和虚拟内存通过 MMU 管理器内存单元连接；内核级别有许多函数控制这种交互）。

● 文件系统（物理和虚拟）— 为文件、目录以及与文件和文件系统功能关联的其他对象提供全局、分层命名空间。 在 Unix 中，一切都是文件，因此我们使用的一切都是文件系统。 任何系统都会访问vfs（虚拟文件系统）驱动程序。

● 字符设备驱动程序。 （直接将数据传输到设备） – 控制需要内核一次发送或接收一个字节数据的设备，例如终端、打印机或调制解调器。 （ls -l：面向链）

● 块设备驱动程序。 (缓冲区优先) – 控制以块的形式读写数据的设备，例如各种类型的辅助存储器（磁盘、CD-ROM 等）。 （ls -l：面向块）

● 网络协议。 TCP/IP。 – 支持一组协议的用户接口套接字（例如，银行业中的 x25）。 他们非常 很多。

● 网络设备驱动程序。 – 管理连接到网桥或路由器等网络设备的网络接口卡和通信端口。

● 陷阱和失败。 – 处理处理器生成的中断，例如内存故障。 （+反应：恐慌，正在加载页面...）

● 中断。 – 处理来自外围设备的中断。

● 信号和IPC – 控制进程间通信。 发送信号的实用程序：kill，有哪些信号：kill -l。 IPC 是一个进程间工具。 互动。 包括：共享内存、信号量、消息。

1. Windows的历史和发展（История и развитие Windows）

Windows操作系统的历史和发展始于20世纪80年代，有几个关键阶段：

1. \*\*MS-DOS：\*\* 1981年，微软发布了用于IBM PC个人电脑的MS-DOS（微软磁盘操作系统）。 MS-DOS 提供了命令接口以及基本的文件和设备管理功能。

2. \*\*Windows 1.0：\*\* 1985 年，Microsoft 发布了 MS-DOS 图形 shell 的第一个版本，称为 Windows 1.0。 它提供了图形用户界面，能够运行图形应用程序，但需要 MS-DOS 才能运行。

3. \*\*Windows 3.0：\*\* 1990年，微软发布了Windows 3.0，成为Windows的第一个流行版本。 Windows 3.0 对用户界面进行了重大改进，包括多任务处理功能和对广泛应用程序的支持。

4. \*\*Windows 95：\*\* 1995年，微软发布了Windows 95，这是第一个完全移植到32位架构的Windows版本。 Windows 95 引入了新的用户界面，包括任务栏、“开始”菜单以及对设备的即插即用支持。

5. \*\*Windows NT：\*\* 1993年，微软发布了Windows NT，它引入了MS-DOS以外的新操作系统架构。 Windows NT 面向商业领域，提供高度的可靠性、可扩展性和安全性。

6. \*\*Windows XP：\*\* 2001年，微软发布了Windows XP，它成为最流行的Windows版本之一。 Windows XP 在用户界面、性能和功能方面进行了重大改进。

7. \*\*Windows Vista、7、8、10：\*\* 在接下来的几年中，Microsoft 发布了多个 Windows 版本，包括 Windows Vista、Windows 7、Windows 8 和 Windows 10。每个新版本都在用户方面进行了改进界面、功能和安全性。

8. \*\*Windows 11：\*\* 2021年，微软发布了Windows操作系统的最新版本Windows 11。 Windows 11 引入了更新的用户界面、改进的对游戏和应用程序的支持以及新的生产力功能。

因此，Windows 操作系统有着悠久的发展历史，从 MS-DOS 和 Windows 1.0 开始，一直到现代版本的 Windows 10 和 Windows 11，为不同类型的用户和设备提供了广泛的功能和支持。

1. 通用Windows体系结构。 视窗应用程序接口（Общая архитектура Windows. Windows API）

Windows 的整体架构由几个关键组件组成，这些组件使操作系统能够运行并与应用程序和硬件交互。 以下是 Windows 体系结构的主要元素：

1. \*\*操作系统内核（Windows Kernel）：\*\* Windows内核提供基本的计算机资源管理功能，包括内存管理、进程管理、数据输入/输出管理、文件系统、网络支持等。

2. \*\*子系统：\*\* Windows 支持多种子系统，如 Win32 子系统、POSIX 子系统、OS/2 子系统等。每个子系统为特定类型的应用程序提供运行时环境，并提供与相关 API 和应用程序的兼容性。功能。

3. \*\*Windows API（应用程序编程接口）：\*\* Windows API是操作系统提供的一组用于与应用程序交互的函数和接口。 它提供对各种 Windows 功能和资源的访问，例如窗口、对话框、文件、注册表、网络等。

4. \*\*图形用户界面（GUI）：\*\* Windows GUI提供了操作系统的用户界面，包括窗口管理器、图形控件（按钮、输入字段、列表等）、动画、效果以及其他使操作更方便的功能。界面更加方便直观。

5. \*\*进程和线程管理：\*\* Windows提供了管理进程和线程的机制，包括它们之间的创建、管理、调度和交互。 这包括内存管理、同步、进程间通信和安全机制。

6. \*\*设备和驱动程序：\*\* Windows 支持多种设备和驱动程序，提供与硬件交互的统一界面。 这包括键盘、鼠标、驱动器、网卡等的驱动程序。

Windows应用程序接口（Windows API），或俗称的WinAPI，是微软Windows操作系统中的一套核心应用程序接口。Windows API将与Windows系统有关的复杂操作封装在简单的函数当中，编译成动态链接库，再随Windows一同发布。开发者可以简单地借助这些函数来完成复杂的操作，有效降低了编写Windows程序的难度。

1. Windows 的服务、功能和重要组件。（Сервисы, функции и важные компоненты Windows.）

Windows 包含许多服务、功能和组件，为用户和应用程序提供广泛的功能。 这里是其中的一些：

1. \*\*资源管理器（Windows资源管理器）：\*\*资源管理器是用户与Windows文件系统交互的主要手段。 它允许您打开和管理文件和文件夹、启动应用程序、创建快捷方式、移动和复制文件以及管理窗口和进程。

2. \*\*控制面板：\*\* 控制面板提供对操作系统的各种设置和功能的访问，例如网络设置、用户帐户、硬件、程序等等。 您可以在此处更改安全设置、管理已安装的软件以及执行其他管理任务。

3. \*\*用户帐户控制 (UAC)：\*\* UAC 通过限制用户和程序的访问权限来提供安全性。 它提供了在执行某些可能影响系统安全的操作时要求管理员确认的机制。

4. \*\*服务：\*\* Windows 服务是在操作系统中执行各种任务的后台进程。 一些服务负责管理网络、打印、安全、任务调度、音频、视频和系统的其他方面。

5. \*\*注册表：\*\* Windows 注册表是操作系统和已安装应用程序使用的配置信息的集中存储。 这是存储确定系统行为和配置的设置、参数和注册表项的位置。

6. \*\*域服务和域控制器：\*\* 在企业环境中，Windows 提供通过域服务和域控制器管理网络设备和用户帐户的功能。

7. \*\*DirectX：\*\* DirectX是Windows中处理多媒体数据的一组API。 它提供对硬件图形和音频加速器的访问，以运行和控制游戏和多媒体应用程序。

8. \*\*Windows Update：\*\* Windows Update 提供了一种自动更新操作系统和已安装软件的机制。 这允许用户获得最新的安全补丁、性能改进和新功能。

这只是 Windows 服务、功能和组件的一个小概述。 Windows 操作系统提供了广泛的工具和功能来满足用户和组织的不同需求。

1. 进程，进程执行时的特征。 进程状态。 资源共享。（Процесс, характеристики процесса в момент выполнения. Состояние процесса. Разделение ресурсов.）

操作系统中的进程是在运行时执行的程序。 以下是该进程执行时的主要特征：

1. \*\*进程标识符（PID）：\*\* 每个进程都有一个唯一的标识符，允许操作系统跟踪和管理它。

2. \*\*进程状态：\*\* 进程可以根据其当前的执行状态处于不同的状态。 一些主要的进程状态包括：

- 正在运行：进程正在处理器上主动运行。

- 就绪：进程正在等待 CPU 时间并准备运行。

- 输入/输出 (I/O)：进程等待 I/O 操作完成。

- 等待状态：进程正在等待某个事件或条件。

3. \*\*进程资源：\*\*进程在执行过程中使用各种计算机资源，包括：

- 处理器时间：分配给进程在处理器上执行的时间。

- 内存：进程使用 RAM 来存储可执行代码、数据和调用堆栈。

- 输入/输出设备：该进程可以与输入/输出设备交互，例如硬盘驱动器、网络接口、键盘、鼠标等。

4. \*\*资源共享：\*\* 在多任务操作系统中，进程共享计算机资源，例如CPU时间、内存和I/O设备。 这允许多个进程同时运行并在它们之间提供隔离以防止干扰和冲突。

5. \*\*进程上下文：\*\*进程上下文包括有关进程当前执行状态的信息，例如处理器寄存器值、指向执行代码的指针、内存页表和其他相关数据。 操作系统使用进程上下文在进程之间切换时保存和恢复进程状态。

这些进程特征有助于操作系统有效地管理进程的执行并分配计算机资源，以保证系统的性能和可靠性。

1. 具有五个状态的过程模型，状态分配。（Модель процесса с пятью состояниями, назначение состояний.）