什么是操作系统：

1. 计算机**核心程序（kernel program）**
2. 用户**接口程序（interface program）**
3. 资源分配与回收的**控制程序（control program）**

操作系统是位于Hardware与User之间的**程序集合**

基本目标：方便性与高效性

工作原理：中断驱动

存储结构：内存，二级存储，分成结构

系统保护：双模式（系统保护），特权指令，内存，IO，CPU（硬件保护）

服务：程序，IO，文件，进程，通信

**硬件（Hardware）：**

启动：执行引导程序，载入operation program kernel（ROM/EPROM中）

中断：告知操作系统执行中断处理程序/策略（interrupt/trap）

Interrupt driven：中断-挂起-中断处理-中断返回

通用中断功能：Interrupt service routine（由**中断向量**保存）、save address（程序恢复）、disable interrupt（进程同步）

Memory： 寄存器-main memory（内存）-electronic disk（U盘）-magnetic disk（磁盘）-optical disk

内存： CPU可以**直接访问**的唯一的大型存储介质

IO设备：控制器**local buffer**和**外设**交互的过程（I/O过程无需CPU干预，中断响应方式）

系统保护：Dual-Mode（用户，内核态）、CPU（进程时间控制）、内存（程序地址控制）、IO保护（全privilege）

**Privileged Instruction**特权指令：可能对**硬件**造成伤害的指令，在内核态执行

**发展(Develop)：**

单任务系统、批处理系统（批处理文件）、多道程序系统（提高CPU利用率，发挥程序**并发性**，内存，CPU，任务，IO资源控制决策）、分时系统（多用户共享，CPU调度，virtual memory）

并行：多CPU同时进行 并发：单CPU资源调配（时间段）

操作系统特征：**并发**（Concurrence），**共享**（Sharing），**虚拟**（Virtual），**异步**（Asynchronous）

CPU利用率：CPU**非空闲时间**占比

Desktop System：单处理器（Single-Processor），多处理器（Multiprocessor），紧耦合（Tightly coupled），分布式（network云端，shared storage集群系统（hot-standby非对称，对称）），实时（时间约束），手势终端（Pocket-PC，Cellular telephone，资源少）系统

优点：高吞吐量（Throughput），资源共享，可靠性高

**功能（Functions）：**

系统内部具有的能力

进程（运行中的程序），内存，磁盘（Secondary-Storage），文件（文件系统），IO管理

**服务（Services）：**

对外可以提供的服务

程序执行，IO操作，文件系统操作（File-system manipulation），进程通信，错误监控

系统执行：资源分配，审计（Accounting安全追踪，系统优化），保护

**接口（interface）：**

命令行（Command-Line），批处理（Batch 配置命令文件.bat），图形用户界面（Graphical User）

Command-line Shell命令行解释器：解读命令

**S**ystem **C**alls **I**nterface系统调用（基本功能实现）：参数放置（寄存器，内存，栈）

**A**pplication **P**rogramming **I**nterface（API）：应用程序接口，调用OS层功能（run-time support library接口使用说明）

**结构（Structure）：**

**简单**（MS-DOS，UNIX 结构区分不明显），**分层**（分为多层 hardware…功能模块化…user layer），**微内核**（Microkernels仅保留进程，内存管理和通信功能 易于扩展和移植，user和kernel转换产生额外开销），**模块化**（面向对象的方法，**灵活性高**）

Virtual machine：模拟硬件，为操作系统提供illusion hardware interface，虚拟机间资源不共享（JVM：解释Java，可移植性很高）

**设计（Design）：**

Policy（策略 做什么），Mechanism（机制 怎么做）

C，C++，汇编

***进程Process：***

按照顺序结构**执行中的程序，**程序决定进程执行顺序（非一一对应）

程序代码（Text section），程序计数器（Program Counter），指令寄存器（执行内容Processor Register），堆栈（堆：程序申请的空间 栈：操作系统分配Heap-stack），数据段（Data section）

**动态性**（Dynamic），**独立性**（Independency），**并发性**（Concurrence），**结构化**（Structure）

进程状态：New（初始）-Ready（准备，获得内存资源）-Running（获得CPU使用权）-Waiting（等待IO响应）-Terminated（exit终止）

CPU核心/用户态：执行kernel/user process

**P**rocess **C**ontrol **B**lock（**PCB进程控制块**）：链表方式连接进程，控制进程状态转换（program counter，pointer）（**管理进程**）

**Job queues所有进程队列**：Ready queue**就绪**（等待执行），Device queue**设备**（等待IO操作），其他队列（均在Main memory作业队列中）

**Scheduler调度器**：Long-term（作业队列 **程序**内存加载选择，控制多道程序度（内存中存放进程个数）），Short-term（就绪队列 CPU执行**进程**选择），Medium-term（**进程镜像保存，**交换swapping 腾出内存空间）

IO进程（以IO操作为主），CPU进程（以CPU计算为主）通过调度器控制

Context switch上下文切换：保存PCB，CPU，内存等信息值，用于区分进程状态（在进程切换时保存context状态）

进程创建：父进程创建子进程形成进程树，并由Process ID标识每个进程

子进程(PID标识子进程（PID<0创建失败），子进程执行PID=0的分支)：

**fork()：子进程复制父进程代码与状态，包含所有子线程**

**exec()**：**父进程执行exec()，覆盖子进程和其子线程代码段执行新功能**

子进程通过getID()获得自己的ID号

进程终止：结束，挂起，随父进程终止（级联终止）

**I**nter-**P**rocess **C**ommunicate进程通信：共享内存（临界区问题，需互斥访问），消息传递（producer consumer）

Bounded-Buffer：通过循环队列控制进程的内存数据读写（可用空间**BUFFER\_SIZE-1**）

Message-passing：通信连接，可供不同主机间通信（物理介质+逻辑链接）

Indirect communication间接通信：进程通过mailbox间接通信，仅当进程共享mailbox时才能建立链接

进程通信：Block同步传输（Synchronization），Non-block异步传输（asynchronous）

Buffering：zero（零容量，等待应答），bound（有限容量，滑动窗口），unbound（无限容量，Post）

CS通信方式：Socket（套接字，应用层访问网络层接口，IP& port，无结构字节流），**R**emote **P**rocedure **C**alls（通过Stubs调用远程**进程和函数**，函数ID与参数，高度结构化），**R**emote **M**ethod **I**nvocation（远程方法调用，调用远程**对象和方法**，**stub（C）-skeleton（S）**）

***线程Thread：***

进程中的**控制流**，CPU执行的最小单位，多个线程**共享进程地址空间**

线程：线程ID，程序计数器，寄存器组，堆栈（不共享），属于同一进程的线程共享code，data，OS recourse（响应性，资源共享，创建和删除开销低，多处理器架构利用率提升）

User thread用户线程：由**用户线程库**管理，内核不知道线程存在（无需状态切换，可能出现线程阻塞导致进程阻塞）

Kernel thread内核线程：由内核管理，不会导致线程阻塞

Multithreading models多线程模型：Many-One（同时只能有一个线程代表进程访问同一个内核），One-One（线程同时访问内核 并发性，费资源），Many-Many（多个用户线程对应多个内核线程），Two level模型（**多对多**，**一对一处理紧急线程**）

Thread states线程状态：join（等待子线程先执行），cancel（取消线程 EN/DIS 立即/延迟 设置取消点），kill（杀死）

API 线程创建：属性，大小，线程函数，是否立即执行（程序中最后的线程退出，程序退出）

Suspend and Resuming：通过栈挂起/恢复线程

并发访问：读写冲突（本地存储，临界区（对共享变量操作的代码段）控制）

Java 线程：extend线程类或采用runnable接口（JVM控制）

Signal：由特定事件产生，通知进程进行挂起（**无优先级** 进程可选择忽略）

Thread Pools线程池：事先创建**内核线程**并放在线程池中，进程需要使用时分配（多对多 分配并新建线程效率高 对**线程总数可控**）

Upcalls：内核对线程表的提醒

***CPU 调度Scheduling***

CPU调度：**CPU空闲时**从就绪队列中选择进程，取决于CPU，IO Burst代码（进行CPU，IO Burst cycle替换）

**CPU/IO**操作**交替执行**：**CPU**（进程少，周期长）/**IO**（进程多，周期短） bound进程

调度：Preemptive 抢占（控制策略）/Non- Preemptive 非抢占调度（进程可执行完）

Dispatcher分配器：CPU控制权调度

**策略评价**：CPU Utilization（CPU**非空时间占比**），Throughput（吞吐量 **单位时间执行的进程数**）**（系统）**，Turnaround（周转时间 提交-任务完成的时间（包含执行时间）），**Waiting time**（**就绪队列**中等待时间），Response time（用户请求-第一个响应）**（用户）**

调度策略Scheduling：

**F**irst **C**ome **F**irst **S**erved**先到先服务**：按照就绪队列顺序依次执行（**取队头** 非抢占式 convey effect护航效应CPU占用 短进程与分时系统不适用）

Gantt Chart：用队列表示调度过程（进程等待时间等于其前队列中进程的所有执行时间之和）

**S**hortest **J**ob **F**irst**最短最优先**：按照剩余CPU数（burst length）进行排序**等待时间：终止-burst-arrival 周转时间：终止-arrival**（理想参考模型）

Priority**优先级**：选择最高优先级执行，可按等待时间长短更改进程的优先级（抢占 进程饥饿（采用Aging老化方法解决）/非抢占）

**R**ound **R**obin**轮询**：响应时间快，适用于**分时系统**，设置进程最长时间片，进程执行完自动释放CPU（应使进程在时间片整数倍里完成 **大多进程能在一个时间片中完成**）

Multilevel Queue多级队列：根据进程需求而分别采用不同调度策略，队列间按照优先级（feedback **多级队列反馈**调度（老化））/时间片（轮询，不同队列设置不同时间片）调度（队列调度策略，进程队列间移动 提高吞吐量，防止CPU占用）

Multiple Processer多处理器调度：负载均衡（push/pull），

非平衡（系统数据仅运行一个处理器访问，降低数据共享需求），SMP（避免处理器选择同一进程）

软亲和（防止进程迁移策略），硬亲和（不允许执行中进程迁移）

Real Time实时调度：**资源预留**（resource reservations）确保进程按时完成（磁盘与虚拟内存无效），硬（确保执行完成）/软（优先级高）实时（**无法使用老化**来防止饥饿）

**Priority Inversion优先级退让**：改变执行中进程优先级防止抢占，执行完后恢复

线程调度：进程级/系统级

策略评价方式：决策（人为设定），队列（人为模拟），模拟（模拟真实系统），实现

***进程同步Synchronization：***

规定进程**执行顺序**，确保共享数据一致性

Race Condition竞争条件：多个进程同时访问同一共享数据时产生

**Critical Section临界区问题**：**更改共享数据**的进程**代码段**，设置互斥访问（同步的子问题）（临界资源：共享数据）

临界区协议：entry（进入 互斥访问），critical（临界区），exit（退出 进展性）部分

解决方法：**互斥**，**进展性**，**有限等待**（三个条件需**同时满足**，证明满足（反证））

让权等待：进程在等待进入临界区且区中进程执行IO时，不能占用CPU资源（多CPU时可以使用自旋锁）

**临界区中进程不一定运行**（可能为IO）**，等待进入临界区进程不一定处于等待状态**

**P**erterson‘s **S**olution： turn（权限控制），flag（进入预约）（**双进程**临界区问题）

**B**akery **A**lgorithm：想进入线程取号，(取号，线程号)先比较取得号大小再比较线程号

循环直到无人取号且进程序号最小，执行CS()，更改number[i]=0;

**硬件支持同步**：禁止中断（单CPU适用，影响系统功能），特殊指令

Test And Set：检测并更正lock为true，占用临界区（TAS需一次性执行，易产生饥饿）

Swap：本地变量key，全局锁lock，交换key与lock满足互斥访问（易产生饥饿）

满足临界区问题解决**TAS**：waiting[i]，lock，key

进入：waiting || key != true （waiting[i]=false）

退出：查找并使下一个进程跳出临界区（waiting[j]=false）

Semaphore信号量：**软件方式**，**分布式**同步机制，**仅由wait/single操作**（类型声明）

Counting：奇数信号量，可用资源个数 Binary：0-1互斥信号量

信号量改进：获取资源失败，阻塞在信号量队列，资源空闲时取出队头执行

双Semaphore：P：wait(Q);wait(S) Q：wait(S);wait(Q)（**相互等待，死锁**）

Starvation**饥饿**：一个进程一直得不到使用权 Deadlock**死锁**：多个进程相互等待

经典同步问题： **临界资源数**（互斥信号量**mutex**），**同步关系**（同步信号量**counter**）

**B**ound-**B**uffer：仅通过wait与single操作

Producer生产者：wait(empty);wait(mutex); single(mutex);single(full);

Consumer消费者：wait(full);wait(mutex); single(mutex);single(empty);

**R**eader-**W**riter：读者（仅读取），写者（可读写）

Reader first读者优先：counter记录读者数，临界区有读者时新读者直接进入，写者等待

**D**ining-**P**hilosophers哲学家就餐问题：5根筷子（5互斥信号量）

**pickup()**{设置hungry，调用test判断是否可以进食}，

**putdown()**{设置thinking，调用test观察左右是否需要进食}

**test()**{测试左右是否在进食，若不均在开始进食，否则阻塞}

独木桥问题：

Entry：获取更改信号量，若获取，释放

Exit：获取更改信号量，若释放，释放

Monitor管程：高级的**同步数据结构**(construct)，将分布的临界区集中管理，进程通过管程访问共享数据，确保访问互斥性

Condition Variable条件变量：进程**进入管程**等待，仅允许x. wait()（**挂起x**）与x. single()（**唤醒** 唤醒后等待/继续） 操作（不进行加减操作）

**管程（解决共享资源的公用数据结构），进程（占有数据资源实现并发性的私有数据结构）**（操作系统需确保进程按照管程规定调用管程操作）

***死锁Deadlocks：***

模型：资源，进程（请求，使用，释放资源）

死锁：进程集合内进程互相等待使集合处于死锁

**资源分配图**：V（进程，资源点），E（请求，分配边）

单实例资源分配图出现环路则死锁，无环则不会死锁

死锁条件：**互斥**（mutual exclusion），**占有等待**，**非抢占**，**循环等待**（不独立）

处理死锁：拒绝（预防（破坏死锁条件之一），避免（满足/延迟资源分配）），发现恢复，忽略死锁

**P**revention**预防**：增加可用资源实例数（互斥），不允许占有（占有等待），抢占（抢别人/被抢），仅申请更高级资源（循环等待）

**A**voidance**避免**：

资源分配图算法（单实例）：单实例（声明，请求，占有边 判断是否成环）

银行家算法（多实例）：假装更新，判断是否满足安全状态，满足则分配

**安全状态**Safe State：系统中包含所有进程的安全序列（，累加Available）

安全一定不死锁，非安全可能死锁

**D**etection**发现**：

等待图：将资源节点移除，边表示等待关系（单实例）

死锁发现：Available, Allocation, **Request**（Work：可用资源，Finish：true占有为0（不参与判断） 得到finish为false死锁队列）（**多实例** 进程请求资源，CPU负载轻运行）

死锁恢复：按一定策略终止/抢占相应进程（优先级，剩余时间片，进程占有，需要资源，交互/批处理）（终止进程后回退 考虑回退次数避免饥饿）

***内存管理Memory Management：***

内存管理：Main Memory，Register，Cache

地址分类：Symbolic符号，Relocatable重定位（相对），Absolute物理地址（绝对）

地址捆绑：**编译，装载，执行**

**逻辑地址**：由CPU管理的虚拟地址 **物理地址**：内存单元地址（编译装载同 执行不同）

重定位寄存器Relocation：**逻辑地址+重定位=物理地址**（便于编程与移植性）

执行时：动态加载（使用时加载程序，动态链接（程序中函数标识，执行时导入（API等）），交换Swapping（虚拟内存Baking Store，CPU直接访问磁盘）

连续分配：

固定分区（内存固定分为大小不同块，放入单个进程 空间浪费）

可变分区（Hole可用内存块，First-fit（**首次适配**，首个合适），Best-fit（**最佳适配**，最小可行 空间排序->首次），Worst-fit（最差适配，最大剩余））（空间利用率：首次=最佳>最差 时间效率：首次>最佳=最差）

Fragmentation碎片：未被使用且不能分配的空间（External外部碎片（分配片段间过小的空闲内存），Internal内部碎片（已分配但未使用））

解决外部碎片问题：Compaction紧缩，不连续分配（分页，

紧缩：将已分配内存上移，腾出可用块

不连续分配：

Paging分页：将物理内存分为大小相同的帧，逻辑地址分为大小相同的页（页表记录映射关系 未解决内部碎片）（页号p：页的基址（帧起始地址），页偏移量d：页中位置）

页表：索引-页号，内容-帧起始地址

页表存放：直接由寄存器存放，PTBR页表寄存器（找页表，访问 **两倍时间**），TLB快表寄存器（存储经常访问的页，页号+所属进程（ASDI））（先查TLB快表，未找到则通过PTBR找内存）

**内存保护**：页表中增加有效-无效位，页表长度寄存器（控制进程访问空间）

Shared Pages共享页：共享只读代码，维护独立代码与数据

分层页表Hierarchical：Two-level（外部页表：子页表索引p1+页表偏移量（p2子页表地址存放位置）+地址，子页表：物理地址）

帧大小=页大小，页表项储存于页表中，程序需要帧则增加页表项