判断题：10

填空题：10

选择题：10

程序题：1

简答题：4

计算题：4-5

**第一章**

**1.存储器——>运算器，控制器的过程：**取指-执行

**2.操作系统是一种管理硬件资源的软件**

**3.操作系统的作用：**为应用程序提供统一的运行环境，并通过进程调度、内存分配和文件系统实现多任务高效执行；同时，它为用户提供交互界面，并确保系统安全与稳定

**4.操作系统的基本特征：并发、资源共享**（cpu，内存，I/O设备，文件，数据）

**5.中断**：在计算机执行期间，系统内发生任何非寻常的或非预期的急需处理事件使得CPU暂时中断当前正在执行的程序而，转去执行相应的时间处理程序。待处理完毕后又返回原来被中断处继续执行或调度新的进程执行的过程

**6.外部中断、trap、exception 他们三者的区别**

| ****特性**** | ****外部中断（Interrupt）**** | ****陷阱（Trap）**** | ****异常（Exception）**** |
| --- | --- | --- | --- |
| ****来源**** | 外部硬件（异步） | 程序主动请求（同步） | CPU执行错误（同步） |
| ****是否可预期**** | 不可预期 | 可预期（如系统调用） | 可能意外（如除零） |
| ****处理目标**** | 响应外部事件 | 提供内核服务 | 处理错误或特殊情况 |

**7.直接内存访问（DMA）**：允许外围设备和主内存之间直接传输它们的I/O数据，而不需要系统处理器的参与

****8.中断与DMA的区别：中断****依赖CPU处理I/O请求，而****DMA****允许外设直接访问内存，无需CPU全程参与，从而减少中断开销。

**9.内存与Graphic和Disk通过DMA的方式连接**

**10.内存与USB外设不直接连接**

**11.多CPU（多个独立CPU芯片，贵，独立）**

**12.多核（单芯片内多个核心，便宜，共享）**

**13.多道程序设计系统：**在内存中同时存放多个程序，由操作系统调度它们交替执行。当某 个程序因I/O操作（如读写磁盘）而阻塞时，CPU立即切换到另一个就绪程序，避免空闲等待。

**第二章**

**1.操作系统提供给用户的界面：**GUI, 图形用户界面，批处理，命令行

**2.系统调用：**系统调用是操作系统提供给用户程序的接口，用于访问内核功能

**3.命令解释器**：shell

**4.系统调用与库函数的区别：**

系统调用：必须从用户态切换到内核态，性能慢

库函数：完全在用户态执行，无需权限切换，可能内部调用系统调用（如fopen()最终调用open()），但本身不直接触发内核模式。性能快

**5.系统调用的类型：**如文件操作、进程控制，设备管理，内存管理，进程间的通信，用户与权限管理，网络通信，系统信息与调试

**6.单层、多层、微内核vs宏内核vs模块化设计**

**单层**：所有功能都在单一内核空间运行，无模块化分层

**多层：**将内核功能划分为多个层次，每个层次仅调用下一层的服务，形成严格的层级依赖

**微内核：**内核态仅保留最核心的功能，其他功能作为用户态服务运行

**宏内核：**所有核心功能均在内核态运行

**模块化设计**：在宏内核的基础上，将功能拆分为可动态加载的模块，运行时按需加载

| ****设计结构**** | ****核心逻辑位置**** | ****性能**** | ****稳定性**** | ****扩展性**** |
| --- | --- | --- | --- | --- |
| ****单层**** | 全部在内核，无分层 | 最高 | 最低 | 最差 |
| ****多层**** | 分层内核，严格调用链 | 中 | 中 | 中 |
| ****微内核**** | 仅核心在内核，其余在用户态 | **最低** | **最高** | 最佳 |
| ****宏内核**** | 全部功能在内核态 | **高** | **较低** | 较差 |
| ****模块化**** | 内核动态加载模块 | 高 | 中 | 优 |

**7.虚拟机技术：**通过虚拟化软件在物理硬件上创建多个虚拟计算机，每个虚拟机（VM）拥有独立的操作系统和应用程序。

|  |  |  |
| --- | --- | --- |
| **printf()** 是系统调用 | ❌ | 它是库函数，内部可能调用系统调用 **write()**。 |
| 系统调用接口用C/C++描述 | ✅ | 因OS内核多用C开发，接口以C函数形式提供 |

**第三章**

**1.进程定义**：运行中的程序，资源分配的最小单位，CPU调度的一个单位

**2.线程定义：**有时称轻量级进程，cpu调度的最小单位，是进程中的一个运行实体

**3.进程和线程的不同之处**：

(1) 调度：线程作为调度的基本单位，同进程中线程切换不引起进程，当不同进程的线程切换才引起进程切换；进程作为拥有资源的基本单位。

(2) 并发性：一个进程间的多个线程可并发。

(3) 拥有资源：线程仅拥有隶属进程的资源；进程是拥有资源的独立单位。

(4) 系统开销：进程大；线程小

**4.PCB（进程控制块）**：是操作系统为每个进程维护的核心数据结构，存储了进程ID、状态、CPU上下文、内存映射、打开文件、调度优先级等所有关键信息，用于实现进程管理、资源分配和上下文切换

**5.长期调度**：批处理系统; **中期调度**：换入换出; **短期调度**：实时系统、分时系统

**6.进程创建：**fork函数：子进程与父进程。exec函数：替换进程映像

**7.进程的通信模型**：消息传递、共享内存

**8.并发与并行：**

| ****概念**** | ****定义**** | ****核心区别**** |
| --- | --- | --- |
| ****并发**** | 多个任务****交替执行****（看似同时，实则在时间片轮转下快速切换）。 | ****逻辑上的同时****（单核即可实现）。 |
| ****并行**** | 多个任务****真正同时执行****（需多核/多CPU硬件支持）。 | ****物理上的同时****（必须多核）。 |

**9.进程的实验：**

创建子进程：使用 fork() 系统调用创建子进程

父进程与子进程的关系：独立地址空间、共享文件描述符、执行顺序不确定。

**第四章**

**1.多线程优势**：****响应性：****单线程阻塞时（如I/O），其他线程可继续执行（如GUI界面保持响应）。

****资源共享：****线程直接共享进程内存，通信成本低

****经济性：****创建/切换线程比进程开销小

****可扩展性：****在多核CPU上真正并行

**2.一对一模型（核心级线程）：**每个用户线程映射到一个内核线程。

优点：真正的并行（内核直接调度）。

缺点：线程创建/切换需内核介入，开销大。

**3.多对多模型（用户加核心线程）：**多个用户线程复用到多个内核线程。

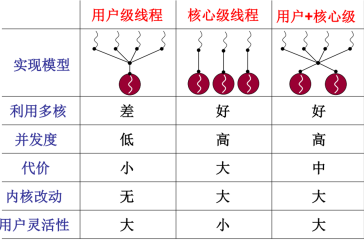
优点：平衡并行度和轻量性。

缺点：调度复杂（需用户态和内核态协作）。

**4.多对一模型（用户级线程）：**多个用户线程映射到一个内核线程。

优点：完全在用户态调度，极轻量。

缺点：无法利用多核，一个线程阻塞则整个进程阻塞。



**5.同一个进程下的线程共享与独立的资源：**

共享的资源：代码段，数据段，堆，进程ID和父进程ID。

独立的资源：线程ID，寄存器状态，栈

**6.线程的组成：**线程ID、程序计数器、寄存器集合和栈

**7. IO约束型任务：**每次使用CPU的时间短，但是频率高，相应的优先级就得高

**8.CPU约束型任务：**每次使用CPU的时间都比较长，切换次数少，这样的任务就优先级就相对低一些

**第五章**

**多级反馈队列：如果进程使用过多CPU时间，那么它会被转移到更低优先级队列。在较低优先级队列中等待时间过长的进程被转移到更高优先级队列**

**饥饿与老化：饥饿**是指某些进程或线程长期得不到所需资源，是资源分配不公平的结果，

**老化**是一种动态调整优先级的技术，通过逐渐提升长时间等待进程的优先级，防止饥饿。是解决饥饿的动态策略。

**多级队列：**将就绪队列分为多个子队列，每队列采用不同调度策略。

**第六章**

**进程的关系**：**同步：**临界区（Critical Section）的访问控制、**互斥**：进程间的时序依赖关系

**临界区**：是指进程之间的一段代码

**临界区访问原则：**每次只允许一个进程进入临界区，进入后不允许其他进程进入。

**解决临界区问题的三个条件**：**互斥，前进，有限等待**

**3个经典问题**：生产者-消费者、读者-写者、哲学家进餐，它们的本质就是：上锁 – 进入临界区 – 解锁

**第七章**

**死锁:**多个进程因循环等待资源而造成无法执行的现象

**发生死锁的必要条件**：互斥使用，不可抢占，占有并等待，循环等待

**死锁预防：**破坏发生死锁的四个条件之一

互斥：资源的固有属性，通常无法破除

占有并等待：需预知未来，编程困难

非抢占：对只能保存和恢复的资源有效，对打印机等外设不适用

循环等待：编程时就需要考虑用户觉得很变扭

**死锁避免：**保证系统始终处于安全状态（银行家算法（复杂度为 O(n² × m)：n进程数。m资源类型数。））

**资源分配图**：有环：可能死锁、无环：无死锁

**死锁检测：**是一种允许死锁发生，但通过定期检查来发现并解除死锁的策略。

**第八章**

**内存的根本目的**：把程序放在内存并让其执行

**重定位：**为执行程序而对其中出现的地址做出修改（逻辑地址->物理地址）

**重定位的时机，每个时机有什么特点：**

编译，效率高，但程序只能放在内存的固定位置

加载：慢但灵活，程序载入内存后不能移动

运行：在内存中任何地方

**静态链接：**在编译/链接阶段将所有依赖的库代码直接合并到最终的可执行文件中。

**动态链接：**在运行时通过动态链接库（如 .so、.dll）加载所需代码，多个程序可共享同一库。

**动态加载：**在程序运行期间**手动**加载/卸载库（如插件系统），通过 API 显式控制

**MMU**： 是 CPU 的硬件组件，负责**虚拟地址到物理地址的转换**，并管理内存访问权限。

**为什么要用MMU**：让每个进程“以为”自己独占内存，同时保证系统安全性和资源利用率。

**MMU的作用：**地址转换，内存保护，内存共享，分页/分段支持，缓存控制

**连续分配：**动态内存分配，分段（典型：首次适应、最佳适应、最差适应）

**外部碎片：**内存加起来满足一个要求，但不连续，无法用

**内部碎片**：已分配的内存内部未被利用的剩余空间

**问题**：碎片->紧缩

**紧缩** ：代价高，在紧缩过程中，cpu将内存中的数据复制到另一个区域，无法完成其他任务

**不连续分配：**分段、分页各自的优缺点

**分段：**支持用户视角的内存管理，管理复杂，外部碎片，内部碎片，内存利用率低

**分页：**管理简单，无外部碎片，页表开销大，内存利用率高，内部碎片，支持虚拟内存

**段页结合：**分段和分页技术的结合，通过逻辑上分段、物理上分页的方式，通过段号查段表，获取页表基址。通过页号查页表，获取物理页帧，最终生成物理地址

**页表很大怎么办：**

层次页表：将页表划分为多级结构，通过树状层次减少内存占用

哈希页表：用哈希表存储页表项，通过哈希函数快速定位映射，减少查找时间。

反向页表：全局唯一页表，按物理页帧索引（而非虚拟页号），大幅减少页表大小

**第九章**

**虚拟内存：**利于系统，又有利于用户，是分段分页的核心

优点：地址空间>物理内存，部分程序放入物理内存

**抖动：**频繁的页面置换导致系统效率急剧下降；

**抖动的原因：**进程分配的物理页不足，导致频繁缺页，cpu大部分时间用于页面置换而非执行任务

**Belady异常**：当分配的物理页面（帧）数量增加时，某些算法的缺页次数反而增多

**第十章**

**文件系统的主要目的**：实现对文件的按名存取

**Linux的文件系统和MS-Windows的文件系统有很大的不同**：它没有“分区”的概念，也即它不会将文件存储位置指定为磁盘驱动器C、D、E等

**Linux的文件系统：**从整体来看是一个树型结构，一般称这样的结构为目录树。整个文件系统是以一个树根“/”为起点的，所有的文件和外部设备都以文件的形式挂结在这个目录树上，包括硬盘、软盘、光驱、调制解调器等

阅读：实验1-linux基本操作，实验2-linux文件目录权限

磁盘的使用：

第一层映射：通过盘块号来访问硬盘

第二层映射：多个进程通过队列使用磁盘

第三层映射：文件字符流到盘块集合的映射

第四层映射：文件系统抽象整个磁盘