简答题整理

Please describe the difference between a process and a program.

- 答：

  程序：

  1. 静态概念
  2. 是指令的集合
  3. 永久存在

  进程：

  1. 动态概念
  2. 描述并发性
  3. 临时存在，包含了数据，代码和进程控制块（PCB）
  4. 一个程序可以实现为多个进程，一个进程也可以调用多个程序
  5. 进程可以创建其他的子进程

Describe the concept of the critical resource and critical region, and give an example for each.

- 答：

  临界资源：一次仅允许一个进程访问的资源。如：硬件资源（输入机、打印机等）；软件资源（共享变量、表格、队列、文件等）。

  临界区：**访问临界资源的程序段。**假设 `a` 为共享变量，则访问 `a` 的那段程序就是临界区。如：`a := a + 1; print(a);`

Will Resource Allocation Graph with a cycle lead to deadlock? Why?

- 答：不一定。如果每个资源只有一个资源实例，则有环路的资源分配图会导致死锁；如果每个资源有多个资源实例，则有环路的资源分配图不一定会导致死锁。

How many disk operations are needed to fetch the i-node for the file `/usr/ast/workspace/mp1.tar`? Why? Assume that the i-node for the root directory is in memory, but nothing else along the path is in memory. Also assume that all directories fit in one disk block.

- 答：

  1. Directory for `/`

2. i-node for `/usr`
3. Directory for `/usr`
4. i-node for `/usr/ast`
5. Directory for `/usr/ast`
6. i-node for `/usr/ast/workspace`
7. Directory for `/usr/ast/workspace`
8. i-node for `/usr/ast/workspace/mp1.tar`

- In total, **8 disk reads** are required.

In a virtual memory system, does a TLB miss imply a disk operation will follow? Why or why not?

答案：不是。TLB 缺失仅意味着需要访问完整的页表（页表通常存储在内存中）。只有在页表也缺失（更具体地说，发生缺页异常）时，才需要进行磁盘操作。

What is the purpose of the open system call in UNIX? What would the consequences be of not having it?

答案：

1. 目的：让系统**将文件属性和磁盘地址列表读入主存，以便后续调用时快速访问。**
2. 如果没有`open`，每次`read`时都必须**指定要打开的文件名。系统随后需要获取该文件的i-node（尽管可以缓存）**。但很快会出现一个问题：何时将i-node写回磁盘？虽然可以设置超时机制，但这种方式会比较笨拙，勉强可行。

A system has `p` processes each needing a maximum of `m` resources and a total of `r` resources available. What condition must hold to make the system deadlock free?

答案：如果一个进程已获得`m`个资源，它就能完成任务而不会陷入死锁。因此，最坏情况是所有进程都持有`m-1`个资源且各自需要再获取一个。此时只要系统剩余至少一个资源，就有一个进程能完成任务并释放其所有资源，从而使其他进程继续执行。
因此，避免死锁的条件是：

$$\text{资源总数} r \geq \text{进程数} p \times (\text{每个进程最大需求} m - 1) + 1$$

即：

$$r \geq p \times (m - 1) + 1$$

3. UNIX 中 `open` 系统调用的作用是什么？如果没有它会导致什么后果?

答:

- **作用**:
  `open()` 用于打开或创建文件，返回文件描述符（File Descriptor），后续的 `read`、`write` 等操作均基于该描述符。主要功能包括:
  - 检查文件是否存在及权限。
  - 建立进程与文件的访问通道。
  - 指定访问模式（如只读 `O_RDONLY`、写入 `O_WRONLY` 等）。
- **没有 `open` 的后果**:
  - 进程无法直接访问文件，必须通过其他复杂机制（如内存映射）间接操作，增加编程复杂度。
  - 无法灵活控制文件访问权限和模式（如并发写入时的冲突问题）。
  - 系统安全性下降，因为缺少权限校验环节。

总结: `open` 是文件操作的入口，缺失会导致文件 I/O 无法高效、安全地进行。

---

4. 系统有 p 个进程，每个进程最多需要 m 个资源，系统共有 r 个资源。如何确保系统无死锁?

答:
根据 **银行家算法（Banker's Algorithm）** 的理论，系统无死锁的条件是:
所有进程的最大需求总量不超过系统资源总数与进程数的某种关系。

具体数学条件为:

$$p \times (m - 1) + 1 \leq r$$

解释:

- 最坏情况下，每个进程都只差 1 个资源即可完成（即已持有 $m - 1$ 个），此时系统至少剩余 1 个资源供某个进程完成，从而避免循环等待。
- 若不满足该条件，则可能所有进程都因资源不足而阻塞，导致死锁。

示例:
若 $p = 3, m = 4, r = 9$，则 $3 \times (4 - 1) + 1 = 10 \nleq 9$，系统可能死锁;
若 $r = 10$，则满足条件，系统无死锁。

What is the biggest advantage of implementing threads in user space? What is the biggest disadvantage?

答案:

优点：

1. 封装性好：完全在用户空间中实现，内核不知道线程的存在
2. 线程管理由用户级线程库完成
3. 线程的切换不需要内核的参与，创建、管理和删除线程方便快捷

缺点：如果内核是单线程的，任何用户级线程发生了阻塞，就会阻塞整个进程。

How does MS-DOS implement random access to files?

答案：通过使用 FAT 的链表分配实现 MS-DOS 的随机访问，FAT 维护一个索引链表，每个 FAT 条目中包含了下一个块的地址，且 FAT 表位于内存中，通过搜索 FAT 表，可以实现快速随机访问，而不需要通过访问块指针实现随机访问。

List the advantages and disadvantages of using small pages in paging systems.

答案

优点：**减少内部碎片，提高了内存的利用率**，避免页面过大导致内存的浪费，使得内存分配更加灵活

缺点：**增加了页表的大小，增加了内存管理的开销和复杂性**

What is a process? What is a thread? How are they similar/different?

答案

进程是正在执行的程序，线程是进程内的一个控制流

相似之处：

- 都是活跃的实体，具有相似的属性，并且会消耗系统资源
- **生命周期**：都有创建、执行和消亡的生命周期
- **创建**：进程可以动态创建进程，被进程创建的线程也可以创建其他线程

不同之处：

- **资源分配**：进程是资源分配的基本单位，拥有独立的虚拟地址空间，而线程不拥有资源，而是共享所属进程的资源
- **地址空间**：不同进程拥有不同的地址空间，而位于同一进程的不同线程共享同一地址空间
- **切换开销**：进程切换涉及到资源指针保存和地址空间的转换，上下文切换开销大，而线程切换不涉及，上下文切换开销较小
- **调度**：进程的调度由操作系统内核完成，而线程的调度可以由内核/用户程序完成

What are the advantages and disadvantages of using FAT (File Allocation Table) in implementing files? And how can we deal with these shortcomings?

答案

优点:

- 整个块都可以用于数据存储，不需要额外的指针指向下一个块
- 支持随机访问
- 目录项只需要一个数字

缺点:

- 整个表需要一次性加载到内存中，表可能非常大

解决方法：可以使用索引分配方法，**将所有指针集中到一个位置，并且只在打开相应文件**时，将索引节点加载到内存中。

List at least three key differences between user-level threads and kernel-level threads.

答案:
用户级线程

- 由应用程序管理
- 内核无法感知线程存在
- 不需要内核权限，创建和管理速度快
- 上下文切换成本低
- 可创建任意数量线程
- 需要谨慎使用：**内核是单线程的，任何用户级线程执行阻塞系统调用时，整个进程都会阻塞。**

内核级线程

- 由操作系统内核负责创建、调度和管理
- 消耗内核资源，开销大
- 切换需要内核支持，上下文切换成本高
- 支持多处理器架构，数量受内核资源限制
- 使用更简单

How many disk operations are needed to open the file `/usr/student/lab/test.doc`? Why? (Assume that nothing else along the path is in memory. Also assume that all directories fit in one disk block.)

答案:

(1) i-node for /

(2) directory for /

(3) i-node for /usr

(4) directory for /usr

(5) i-node for /usr/student

(6) directory for /usr/student

(7) i-node for /usr/student/lab

(8) directory for /usr/student/lab

(9) i-node for /usr/student/lab/test.doc

In total, 9 disk reads are required.

进程与线程的区别是什么？使用线程有哪些好处？

- **进程**
  - 资源所有权的单位，与程序的执行相关。
  - 可以包含多个执行线程。
  - 进程之间相对独立。
  - 创建成本高。
  - 上下文切换成本高。
- **线程**
  - 执行的单位。
  - 属于某个进程。
  - 线程是同一“作业”的一部分，紧密合作。
  - 创建成本低。
  - 同一进程内的上下文切换成本低。
- **优点**
  - 创建线程比创建进程便宜（约10-20倍）。
  - 在同一进程中切换到不同线程的成本低（5-50倍）。
  - 同一进程内的线程**可以更方便、高效地共享数据和其他资源**（无需复制或消息传递）。

如果磁盘采用双倍交错，是否还需要柱面偏移以避免在进行磁道到磁道寻道时丢失数据？简要解释你的答案。

也许需要，也许不需要。双重交错实际上相当于两个扇区的圆柱柱面偏移。**如果磁头能在少于两个扇区时间内完成磁道间寻道，那么就不需要额外的圆柱面偏移。如果不能，那么就需要额外的柱面偏移，以避免寻道后丢失一个扇区。**

在分页系统中，页表可能非常大，需要大量内存空间。给出两种可能的解决方案，并简要解释。

（1）多级页表（Hierarchical Page Table）

- **原理**：将页表分层（如二级、三级页表），每级页表仅存储下一级页表的地址。例如，二级页表中，一级页表的每个条目指向一个二级页表，仅当需要访问某部分地址空间时，才加载对应的下级页表到内存。
- **优势**：避免一次性加载完整页表，减少内存占用。

（2）倒置页表（Invert Page Table）

- **原理**：**每个物理页帧一个条目（PTE）**，条目由存储在该真实内存位置的页面的**虚拟地址以及拥有该页面的进程的相关信息**组成。（进程、虚拟页面）**物理页码用作表中的索引**
- **优势**：物理地址的空间范围较小，对应实现的页表也更新，减少内存占用。

（3）哈希页表（Hash Page Table）

- **原理**：通过哈希函数将虚拟页号映射到页表项，页表存储哈希表条目（含虚拟页号、物理页帧号及冲突链表指针）。访问时，先计算虚拟页号的哈希值，再查找哈希表获取物理地址。
- **优势**：适用于稀疏地址空间（如大地址空间中仅部分地址被使用），无需为未使用的地址分配页表项，节省内存。

硬链接和符号链接的区别是什么？

3. 硬链接与符号链接的区别

维度	硬链接（Hard Link）	符号链接（Symbolic Link）
本质	同一文件的多个目录项，共享同一个 inode（文件索引节点）。	独立文件，存储目标文件的路径字符串（类似快捷方式）。
存储内容	不存储文件数据，仅指向 inode 编号。	存储目标文件的绝对或相对路径。
跨文件系统	不能跨文件系统（因 inode 编号仅在本文件系统内唯一）。	可以跨文件系统（路径指向不受文件系统限制）。

维度	硬链接（Hard Link）	符号链接（Symbolic Link）
删除影响	删除硬链接不影响文件，仅当所有硬链接和原文件均被删除时，文件才被删除。	删除目标文件后，符号链接失效（变为“broken link”）。删除符号链接不会影响文件
权限	与原文件共享权限，无法独立设置。	作为独立文件，可单独设置权限，但访问时受目标文件权限限制。
示例	`ln source_file link_file`（创建硬链接）。	`ln -s target_file symlink_file`（创建符号

什么是操作系统

- **操作系统是一台做拓展机器**：隐藏了实现的诸多细节，向用户提供了易于使用的虚拟机
- 操作系统是一台资源管理器：允许多个程序同时允许，且支持管理和保护内存，I/O设备和其他资源

OS 概念

- 进程：正在进行的程序，系统为每个进程定义了一个数据结构，即进程控制块PCB，用于对该进程进行控制和管理，一个进程可以创建一个或者多个其他的进程，这些进程又可以创建他们的子进程，形成了树形结构。
- 地址空间；可以分配的内存大小，从 0 到 max，进程可以根据其进行读写，对内存实现管理
- 管道（pipe）：是一种特殊的伪文件，用于连接两个进程，将一个进程的输出作为另外一个进程的输入
- I/O 子系统：用于管理 I/O 设备，包含了 I/O 软件。设备驱动程序（隐藏了硬件设备的实现细节）将标准调用映射到特定的操作。目的是为了实现应用程序的源代码在多个操作系统之间的可移植性
- 保护：必须防止未授权的资源访问，防止一个用户对另外一个用户造成干扰。包含了3位（读写和执行），一共三个字段：拥有者、组和其他。

什么是系统调用？作用是什么

- 提供了运行程序和操作程序之间的接口，目的是为了应用程序的源代码在多个OS之间的可移植性。

什么是进程？

- 进程是正在执行的程序，是程序在某个数据集合上的一次活动，是系统资源分配和管理的最小单元。

进程的状态有哪些？他们之间是如何进行转换的？

- 三种状态：Blocked，Ready，Running
- 转换：

- Blocked to Ready：I/O操作完成
- Ready to Running：CPU调度执行
- Running to Blocked：请求 I/O 读取操作
- Running to Ready：CPU 时间片使用完

什么是死锁？需要满足的四个条件是什么？

- 一组进程中的每个程序都在等待一个事件，而该事件只能由该组中的其他进程引起，通常该事件是资源的释放。此时没有一个进程可以：运行，释放资源，被唤醒。
- 四条件：
  - 互斥：每个资源只能分配给一个进程，或者处于可用状态
  - 占用和等待：进程在持有资源的同时，会请求额外的资源
  - 不可抢占：已经分配给进程的资源不能被强制夺走，只能由其本身释放
  - 循环等待：必须存在一个由多个进程组成的循环练，即每个进程都在等待下一个进程所持有的资源

资源分配图

- **如果 图 无环，则无死锁。**
- 如果包含环：
  - **如果每种资源类型只有一个实例，则死锁。**
  - 如果每个资源类型**有多个实例，则可能会发生死锁。**

处理死锁的方法的方法有哪些？

- 鸵鸟算法：

  Windows 和 Unix 采用鸵鸟算法，假装死锁不存在（原因：如果死锁很少发生，且预防死锁的成本很高，这种方法是合理的）在便利性和正确性之间的权衡
- 恢复：
  - 通过抢占进行恢复：从其他进程中获取资源，取决于资源的性质
  - 通过回滚进行恢复：定期保存进程状态，如果出现死锁，则重启进程
  - 通过杀死进程进行恢复：杀死死锁循环中的一个进程，然后获取其资源解决，最简单最暴力
- 预防：攻击四个死锁条件其中之一
  - 针对互斥：通过资源共享来避免互斥，只在必要情况下分配资源，实际分配资源的进程尽可能少
  - 针对保持和等待条件：要求进程在开始运行前请求所有的资源，不需要等待它所需要的资源
  - 针对不可抢占条件：通过强制抢占资源实现，但实际上很难实现
  - 循环等待条件：通过全局资源排序或者限制进程请求资源的顺序来避免循环等待

- 避免：动态检查资源的分配，通过银行家算法确保系统永远不会进入死锁（不安全）状态，即要求进程在运行前声明对每种资源的最大需求

内存管理的方法有哪些？各自的优缺点是什么？

- 位图：每个分配单元对应位图中的一个位，0表示空闲，1表示已经分配
  - 优点：简单直观
  - 缺点：查询速度慢，如果要为一个新到达的进程分配存储空间时，需要找到练习的0位，且单元越小，位图越大，浪费空间
- 链表：链表中每个条目指定一个空闲块或者已分配块的起始地址和长度，此外还有指向下一个条目的指针，一般按照地址或者大小排序（推荐地址，管理方便，容易增删查改）
  - 优点：动态管理内存高效
  - 缺点：更新操作复杂

什么是虚拟内存？优点是什么？如何实现

- 定义：将用户逻辑内存和实际的物理内存分离开来，提供比物理内存更大的逻辑地址空间
- 实现：通过将部分程序存储在磁盘上，只加载部分程序在内存中运行实现
- 优势：
  - 多程序运行
  - 提供内存利用率
  - 运行进程共享地址空间
  - 提高进程的创建效率
- 具体实现方法：分页、分段、分页和分段混合

页表是什么？其简单实现为register数组和单级别页表，有什么问题？有什么改进方法？

- 页表是将 VPN（虚拟页码）映射到 PFN 的表格。大多数 OS 为每个进程分配一个页表。
- register
  - 优势：实现简单
  - 缺点：页表过大成本过高，在每次上下文切换时加载整个页表会影响性能开销
- 单级别页表：
  - 优势：上下文切换成本低
  - 缺点：读取页表时需要多次内存访问
- 可以通过 TLB 进行改进：TLB 是一个用于存储最近使用的页表条目的高速缓存
  - 优点：快速查找虚拟地址到物理地址的映射，提高内存的访问速度
  - 缺点：TLB未命中时需要访问页表，增加延迟时间，此外还需要考虑TLB的替换策略

- 可以通过多级页表实现或者反转页表实现：
  - 多级页表：将页面表划分为多个层次，减少页表的大小，如将页面号分为多个部分，作为不同层次页面表的索引
    - 优点：减少页表的大小，不需要加载整个页表
    - 缺点：可能需要对内存的多次访问
  - 反转页表：每个物理页帧一个条目，条目由存储在该真实内存位置的页面的**虚拟地址以及拥有该页面的进程的相关信息**组成。物理页码用作表中的索引
    - 优点：适用于大地址空间的小页表
    - 缺点：查找困难，需要哈希链等开销

页面大小过大或者过小有什么优缺点?

优势

- **减少内部碎片**：小页面可以减少内存中未使用的程序部分，从而降低内部碎片。
- **提高内存利用率**：小页面使得内存分配更加灵活，减少了因页面过大而导致的内存浪费。

劣势

- **增加页表大小**：程序需要更多页面，导致页表变大，增加了内存管理的复杂性。
- **增加管理开销**：页表的大小和管理开销会随着页面数量的增加而增加。

文件实现的方法有哪些？各自的优缺点是什么?

- 连续分配：将每个文件存储为连续的数据块
  - 优点：实现简单，读取性能优秀（适合顺序访问和直接访问）
  - 缺点：存在外部碎片，文件大小必须在创建时被确定（文件大小无法增长，只适合 CD-ROM、DVD 和其他一次性写入的光介质）
- 链表分配：将文件存储为磁盘块的链表，块可以分散在磁盘上。每个块的首个字用作指向下一个块的指针。
  - 优点：没有外部碎片，文件可以动态增长，目录项简单，适合顺序访问
  - 缺点：随机访问速度慢
- 使用 FAT 的链表分配：从每个数据块中取出**表指针字，并将它们放入索引表**，即文件分配表（FAT）。每个分区开头都保留一段磁盘空间来存放 FAT，通过查询FAT表寻找目标块
  - 优点：
    - 整个块用于存储数据，提高内存利用率。
    - 实现随机访问（搜索在内存中的整个FAT链表）
    - 目录项只需要一个字（起始块号）
  - 缺点：
    - 整个表都在内存中，FAT 表可能非常庞大，损耗性能

- 索引分配：每个文件都有自己的 i-Node，其中包含了文件属性和磁盘块地址（所有指针汇集在一个位置（索引块或者索引节点））打开文件时，加载相应的 i-node 到内存中即可
  - 优点：
    - 支持快速查找和随机访问
    - 不会受到外部碎片影响
    - 只有在打开文件时 i-node 才加载到内存中，提高资源利用率
  - 缺点：使用 i-node 需要额外的空间开销

对于文件存储，磁盘空间管理的方法有哪些？优缺点是什么？

- 连续分配：为文件分配连续的地址空间
  - 优点：简单容易实现，适合顺序访问
  - 缺点：文件增长时需要移动，可能会导致外部碎片
- 分页分配：将文件分配为大小固定的块
  - 优点：避免外部碎片
  - 缺点：可能会产生内部碎片，最后一个块利用率低

磁盘块大小过大或者过小对文件存储的影响？

- 大块：
  - 产生内部碎片，平均浪费 二分之一 块空间
  - 性能更好，但磁盘利用率低
- 小块：
  - 需要花费更多的寻道时间，文件访问速度慢
  - 磁盘利用率高，性能更差
- 总结：
  - 更大的块大小：更高的数据速率，更低的存储空间利用率
  - 更小的块大小：更低的数据速率，更高的存储空间利用率

如何管理磁盘的空闲块？哪种方法更优？

- 链表：每个块尽可能存储多的空闲块，此外还需要一个指针指向下一个管理块
  - 优点：简单易实现
  - 缺点：需要更多的块管理空闲块
- 位图：一个包含 n 个块的磁盘需要一个包含 n 个位的位图，每个块用一个位表示（1表示空闲，0表示已分配）
  - 优点：如果能完全保存在内存中，效率更高
  - 缺点：需要更高的空间存储位图

- 比较：需要具体的分析：假设存在一个 16GB 的磁盘，每个块大小为 1KB，那么需要对应的位图大小为 2^{24}bits，即对应 2048 个 1KB 块

  使用链表的话：假设一个存储管理条目为4bytes，一个块最多管理255个块，一个作为指针指向下一个块，那么需要 $2^{24}/255 = 65793$ 个块
- 此外还有计数法：记录每个连续的空闲的首地址和对应的数量
  - 优点：空闲块越多，存储需求空间越少
  - 缺点：每个存储条目需要更多空间：一个磁盘地址，一个计数

I/O 软件的目标是什么?

- 设备独立性：程序无需提前指定设备，可以访问任意的I/O设备
- 统一命名：设备名称是一个字符串或者数字，不依赖于具体的机器
- 错误处理尽可能靠近底层解决
- 实现同步和异步传输：同步传输会阻塞，而异步传输不会
- 缓存数据
- 实现设备的共享和专用

设备驱动程序是什么？功能是什么?

- 设备驱动程序是管理设备控制器和操作系统之间进行交互的模块，负责将设备独立请求转换为设备依赖请求，一般处理一种类型的设备，可适用于多种操作系统。
- 功能：
  - 接收来自其他部分的读写请求
  - 将抽象术语转换为具体术语
  - 初始化设备
  - 检查设备是否在被其他请求使用、排队请求
  - 向设备发出一系列命令
  - 检查错误类型等

设备独立的 I/O 软件功能是什么?

- 为设备驱动程序提供统一接口。
- 缓冲。
- 错误报告。
- 分配和释放专用设备。
- 提供设备独立的块大小。