

第 3 章：进程描述与控制 (Process Description and Control)

参考来源：

一、Key Terms (名词解释)

1. **Blocked State (阻塞状态)**: 进程在等待某个事件（如 I/O 完成）发生而无法执行的状态。
2. **Child Process (子进程)**: 由另一个进程（父进程）创建的进程。
3. **Dispatcher (分派器)**: 操作系统调度器的一部分，负责将 CPU 控制权交给被选中的进程（包括上下文切换）。
4. **Exit State (退出状态)**: 进程已终止执行，资源已被释放，但其 PCB 暂时保留以供父进程收集信息的状态。
5. **Interrupt (中断)**: 外部硬件（如 I/O 设备、时钟）产生的信号，打断 CPU 当前指令的执行，转而去处理特定事件。
6. **Kernel Mode / System Mode / Privileged Mode (内核模式/系统模式/特权模式)**: CPU 的一种执行模式，允许执行所有指令（包括特权指令）和访问所有内存区域，通常用于操作系统内核代码的执行。
7. **Mode Switch (模式切换)**: CPU 执行状态在用户模式和内核模式之间的切换（通常由中断或系统调用触发）。
8. **New State (新建状态)**: 进程刚刚被创建，还未加载到主存中的状态。
9. **Parent Process (父进程)**: 创建了当前进程的那个进程。
10. **Preempt (抢占)**: 操作系统强制中断当前运行的进程，收回 CPU 并分配给其他进程。
11. **Process / Task (进程/任务)**: 一个正在执行的程序实例，包含程序代码、数据、栈和 PCB。
12. **Process Control Block (PCB, 进程控制块)**: 操作系统用于存储进程信息（状态、寄存器、优先级等）的数据结构，是进程存在的唯一标志。
13. **Process Control Information (进程控制信息)**: PCB 中用于操作系统管理进程所需的额外信息（如调度信息、状态信息）。
14. **Process Image (进程映像)**: 进程在内存中的完整分布，包括程序代码、数据、栈和 PCB。
15. **Process Spawning (进程派生)**: 操作系统应某个进程请求而创建一个新进程的过程。
16. **Process Switch (进程切换)**: 保存当前进程的上下文并恢复另一个进程上下文的过程（比模式切换开销大）。
17. **Program Status Word (PSW, 程序状态字)**: 一个寄存器或寄存器组，包含条件码、执行模式（用户/内核）等状态信息。
18. **Ready State (就绪状态)**: 进程已具备运行条件，只等待 CPU 分配的状态。
19. **Round-Robin (轮转调度)**: 一种调度算法，每个进程被分配一个固定的时间片，时间到后被抢占并放入就绪队列末尾。
20. **Running State (运行状态)**: 进程当前正在 CPU 上执行指令的状态。
21. **Suspend State (挂起状态)**: 进程被换出到辅存（磁盘）中，不占用主存空间的状态。
22. **Swapping (交换)**: 将进程的一部分或全部从主存移动到辅存（或反之）的技术，用于释放内存。
23. **Time Slice (时间片)**: 在轮转调度中，分给每个进程连续使用 CPU 的最大时间段。
24. **Trace (轨迹)**: 单个进程执行的指令序列。
25. **Trap (陷阱)**: 由当前执行指令产生的异常或同步中断（如除零、系统调用）。

26. **User Mode (用户模式):** CPU 的一种非特权模式，禁止执行特权指令，用于运行用户应用程序。

二、Review Questions (复习题答案)

- **3.1 什么是指令轨迹 (Instruction Trace)?**
 - 答：指令轨迹是进程执行过程中，指令被 CPU 执行的顺序序列。
- **3.2 解释进程的概念，并说明它与程序的区别。**
 - 答：进程是“正在运行的程序”。程序是静态的代码文件，而进程是动态的执行实体，拥有独立的内存空间、状态和系统资源。
- **3.3 简述图 3.6 进程模型中各状态的定义。**
 - 答：(1) **运行**: 正在占用 CPU；(2) **就绪**: 准备好运行，等待 CPU；(3) **阻塞**: 等待事件；(4) **新建**: 刚创建；(5) **退出**: 执行结束。
- **3.4 抢占 (Preempt) 是什么意思?**
 - 答：指操作系统因时间片用完或高优先级进程到来，强制暂停当前运行进程并剥夺其 CPU 使用权。
- **3.5 什么是进程派生 (Process Spawning)?**
 - 答：当操作系统根据一个现有进程的显式请求创建一个新进程时，称为进程派生。
- **3.6 为什么图 3.9b 有两个阻塞状态?**
 - 答：为了区分**在内存中的阻塞** (Blocked) 和**被换出到磁盘的阻塞** (Blocked/Suspend)。这有助于操作系统决定是否需要将进程换入/换出内存。
- **3.7 列出挂起进程 (Suspended Process) 的四个特征。**
 - 答：(1) 进程不在主存中；(2) 进程在等待某些事件（但也可能没在等待）；(3) 除非被显式命令（如从父进程），否则不能被调度执行；(4) 通常是由于系统为了释放内存或排错而将其挂起。进程不能立刻执行。为阻止该进程执行，课通过代理使其置于挂起态，代理可以是进程本身也可以是父进程OS
- **3.8 操作系统维护哪些类型的实体信息表?**
 - 答：内存表、I/O 表、文件表、进程表。
- **3.9 进程映像 (Process Image) 的要素是什么?**
 - 答：用户数据、用户程序、系统栈、进程控制块 (PCB)。
- **3.10 为什么需要两种模式 (用户和内核)?**
 - 答：为了保护操作系统和关键系统表不受用户程序的干扰或破坏，确保系统稳定性。
- **3.11 操作系统创建一个新进程的步骤是什么?**
 - 答：(1) 分配唯一进程标识符；(2) 分配空间；(3) 初始化 PCB；(4) 设置链接（如放入队列）；(5) 创建或扩充其他数据结构。
- **3.12 中断 (Interrupt) 和陷阱 (Trap) 的区别是什么?**
 - 答：中断由外部事件（硬件）触发，与当前指令无关（异步）；陷阱由当前执行的指令（软件）触发，如错误或系统调用（同步）。
- **3.13 举出三个中断的例子。**
 - 答：时钟中断、I/O 完成中断、内存缺页中断。
- **3.14 模式切换 (Mode Switch) 和进程切换 (Process Switch) 的区别是什么?**

- 答：**模式切换**仅改变 CPU 的执行权限（用户态<->内核态），不需要保存/恢复完整的进程环境；**进程切换**涉及从一个进程换到另一个进程，需要保存和恢复整个 PCB 上下文，开销大得多。

第 4 章：线程 (Threads)

Running，ready 是调度决定的，派生，阻塞等是操作决定的

挂起是进程，线程共享用户空间，所以不谈论挂起一个线程

参考来源：

一、Key Terms (名词解释)

1. **Application (应用)**: 完成特定任务的软件程序，可包含一个或多个进程。
2. **Fiber (纤程)**: 一种比线程更轻量级的执行单元，通常由应用程序（而非内核）手动调度。
3. **coroutine (协程)**: 指一种程序组件，能在执行过程中暂停自身、保存当前状态，之后再从暂停处恢复执行，常用于异步编程、任务调度等场景
4. **Jacketing (包衣/封装)**: 一种技术，将阻塞式系统调用转化为非阻塞式调用，以避免用户级线程阻塞整个进程。
5. **Job Object (作业对象)**: (Windows概念) 允许将多个进程作为一个组来管理和控制的内核对象。
6. **Kernel-Level Thread (KLT, 内核级线程)**: 由操作系统内核管理的线程。内核维护线程上下文，调度以线程为单位。
7. **Lightweight Process (LWP, 轻量级进程)**: 在某些系统中（如 Solaris），作为用户级线程和内核级线程之间的桥梁，通常指内核支持的线程。
8. **Message (消息)**: 线程或进程间通信的数据单元。
9. **Multithreading (多线程)**: 操作系统支持在单个进程内并发执行多个线程的能力。
10. **Namespaces (命名空间)**: (Linux概念) 用于隔离系统资源（如 PID、网络），是容器技术的基础。
11. **Port (端口)**: 消息传递的端点或通道。
12. **Thread (线程)**: 进程内的执行单元（调度单位），拥有独立的程序计数器、栈和寄存器，但共享进程内存。
13. **Thread Pool (线程池)**: 预先创建一组线程等待分配任务，避免了频繁创建/销毁线程的开销。
14. **User-Level Thread (ULT, 用户级线程)**: 完全由用户空间库管理的线程，内核不可见。切换快，但一个阻塞会导致整个进程阻塞。
15. **User-Mode Scheduling (UMS, 用户模式调度)**: 允许应用程序直接调度线程而无需切换到内核模式，提高效率。
16. **Process / Task**: (同第3章，略)

二、Review Questions (复习题答案)

- 4.1 表 3.5 列出了 PCB 的元素。其中哪些属于线程控制块 (TCB)? 哪些属于进程控制块 (PCB)?
 - 答：**属于 TCB (独有)**: 寄存器状态、程序计数器、栈指针、处理器状态字、线程状态。**属于 PCB (共享)**: 内存地址空间、打开的文件、进程 ID、所有者信息。

多线程共享进程的代码段和数据段(全局变量)，每个线程有自己的程序计数值pc和局部变量stack
- 4.2 列出线程间的模式切换比进程间切换便宜的原因。

- 答：线程共享内存和资源，切换时不需要切换内存地址空间（TLB 刷新等），只需要保存/恢复少量寄存器状态。创建快结束快切换快通信快

- 缺点 (Disadvantages)

主要体现在**安全性（隔离性）*初*同步复杂性**上：

1. 健壮性/隔离性差 (Lack of Isolation):

- 这是线程最大的弱点。因为共享地址空间，一个线程的崩溃（例如非法内存访问）通常会导致**整个进程**（包括该进程内的所有其他线程）崩溃。
- 相比之下，进程之间是隔离的，一个进程崩溃通常不会影响到其他进程。

2. 同步困难 (Synchronization Issues):

- 因为共享数据，多个线程并发访问同一块内存时容易发生**竞争条件 (Race Condition)**。
- 程序员必须小心地使用锁（Mutex）、信号量（Semaphore）来保护临界区，这增加了编程难度和死锁的风险。

3. 阻塞问题 (Blocking Issue) —— 特指用户级线程 (ULT):

- 如果使用纯用户级线程（User-Level Threads），当一个线程执行阻塞式系统调用（如读磁盘）时，操作系统会阻塞整个进程，导致该进程内的其他就绪线程也无法运行。

• 4.3 进程的两个基本且独立的概念是什么？

- 答：(1) **资源所有权** (Resource Ownership); (2) **调度/执行** (Scheduling/Execution)。

• 4.4 举出四个在单用户多处理系统中使用线程的例子。

- 答：(1) 前台与后台处理（如电子表格计算的同时响应输入）；(2) 异步处理（如定期备份）；(3) 执行速度加快（并行计算）；(4) 模块化程序结构。

• 4.5 线程与进程有何不同？

- 答：进程是资源分配单位，线程是 CPU 调度单位。多线程共享同一进程的资源。

• 4.6 使用多线程代替多进程有哪些优点？

- 答：创建速度快、终止速度快、切换速度快、线程间通信效率高（直接读写共享内存）。

• 4.7 列出用户级线程 (ULT) 相对于内核级线程 (KLT) 的优缺点。

- **优点:** 切换不需要内核模式权限（快），调度算法可由应用定制，跨平台性好（可以在任何操作系统上执行）。
- **缺点:** 一个线程阻塞会导致整个进程阻塞，无法利用多处理器并行运行同一进程的线程。
• Multiple processes 用多进程代替多线程 • Jacketing 套管 针对阻塞问题
- 具体速度取决于应用程序性质如果需要频繁的切换到内核进程的时候ULT不见得速度比KLT更加快

• 4.8 以 Clouds 操作系统为例解释线程概念。

- 答：(此题较偏，通常指对象导向系统) 在 Clouds 中，线程可以跨越对象边界移动，执行路径穿过不同的对象和地址空间。
-

第 5 章：并发：互斥与同步

参考来源：，

一、Key Terms (名词解释)

1. **Atomic (原子性)**: 操作不可分割，要么全做要么全不做，中间状态不可见。
2. **Binary Semaphore (二元信号量)**: 只能取值 0 或 1 的信号量，类似互斥锁。
3. **Blocking (阻塞)**: 进程等待某种条件。
4. **Busy Waiting / Spin Waiting (忙等/自旋)**: 进程循环检查条件是否满足，消耗 CPU。
5. **Concurrency (并发)**: 多个进程在时间上重叠执行（交替或并行）。
6. **Concurrent Processes (并发进程)**: 能够并发执行的进程。
7. **Condition Variable (条件变量)**: 用于 Monitor 中，允许线程等待特定条件满足的数据类型。
8. **Coroutine (协程)**: 能够挂起和恢复执行的子程序，通常用于协作式多任务。
9. **Counting Semaphore (计数信号量)**: 取值可以是任意非负整数的信号量，常用于资源计数。
10. **Critical Resource (临界资源)**: 一次仅允许一个进程使用的资源。
11. **Critical Section (临界区)**: 访问临界资源的代码段。
12. **Deadlock (死锁)**: (见第6章)
13. **Direct/Indirect Addressing (直接/间接寻址)**: 在消息传递中，直接指定接收者进程 ID，或通过邮箱（间接）发送消息。
14. **Livelock (活锁)**: 进程在不断改变状态以响应对方，但无法做任何实质性进展。
15. **Message Passing (消息传递)**: 进程间通过发送和接收消息来交换信息。
16. **Monitor (管程)**: 一种高级同步机制，封装了共享数据和操作过程，自动实现互斥。
17. **Mutual Exclusion (互斥)**: 确保同一时间只有一个进程在临界区执行。
18. **Mutex (互斥锁)**: 用于实现互斥的变量，通常包含所有权（谁加锁谁解锁）。
19. **Nonblocking (非阻塞)**: 操作立即返回，不等待。
20. **Race Condition (竞争条件)**: 多个进程并发访问共享数据，最终结果取决于执行顺序。
21. **Semaphore (信号量)**: 用于同步的整数变量，仅通过 P/V (wait/signal) 操作访问。
22. **Starvation (饥饿)**: 进程长期得不到调度或资源。
23. **Strong/Weak Semaphore (强/弱信号量)**: **强**: 等待队列按 FIFO 顺序唤醒；**弱**: 唤醒顺序不确定。

二、Review Questions (复习题答案)

- **5.1 列出与并发相关的四个设计问题。**
 - 答：(1) 操作系统必须跟踪所有进程；(2) 分配和释放资源；(3) 保护数据和物理资源；(4) 进程执行结果必须独立于速度（避免竞争条件）。
- **5.2 产生并发的三种上下文是什么？**
 - 答：(1) 多应用程序；(2) 结构化应用程序；(3) 操作系统结构。
- **5.3 什么是竞争条件 (Race Condition)?**
 - 答：当多个进程读写共享数据，且最终结果取决于进程运行的精确时序时发生的情况。
- **5.4 进程对彼此感知的三个程度及定义？**

- 答: (1) **互不感知**: 竞争关系; (2) **间接感知**: 通过共享对象 (如文件) 合作; (3) **直接感知**: 通过通信 (如消息) 合作。
 - **5.5 竞争进程和合作进程的区别?**
 - 答: 竞争进程争夺资源, 互不知道对方; 合作进程共享数据或通过通信协同工作。
 - **5.6 列出与竞争进程相关的三个控制问题。**
 - 答: 互斥、死锁、饥饿。
 - **5.7 什么是饥饿 (Starvation)?**
 - 答: 进程虽然可运行, 但因调度策略导致无限期被忽视, 无法获得资源。
 - **5.8 二元信号量与信号量的操作有何不同?**
 - 答: 二元信号量的值只能是 0 或 1; 普通计数信号量可以是任意整数。
 - **5.9 二元信号量与互斥锁 (Mutex) 的主要区别?**
 - 答: Mutex 有所有权 (必须由加锁者解锁), 信号量没有所有权 (可以由其他进程解锁/signal)。
 - **5.10 管程 (Monitor) 与信号量的主要区别?**
 - 答: 管程是编程语言级别的结构, 自动实现互斥, 更易用且不易出错; 信号量是低级原语, 需程序员手动管理。
 - **5.11 将管程标记为高级同步工具的特征是什么?**
 - 答: 它是一个包含数据和过程的模块, 同一时间只允许一个进程进入管程内部。
 - **5.12 比较直接寻址和间接寻址的消息传递。**
 - 答: **直接**: Send(ProcessID, msg), 明确指定目标; **间接**: Send(Mailbox, msg), 发送到中间实体 (邮箱), 解耦发送者和接收者。
 - **5.13 读者/写者问题通常与什么条件相关?**
 - 答: 允许多个读者同时读, 但写者必须独占访问 (互斥所有其他读者和写者)。
-

第 6 章: 并发: 死锁与饥饿

参考来源:

一、Key Terms (名词解释)

1. **Banker's Algorithm (银行家算法)**: 一种死锁避免算法, 通过模拟资源分配来检查系统是否处于安全状态。
2. **Circular Wait (循环等待)**: 死锁条件之一, 存在一个封闭的进程链, 每个进程都在等待下一个进程持有的资源。
3. **Consumable Resource (消耗性资源)**: 可以被创建和销毁的资源 (如中断、信号、消息)。
4. **Deadlock (死锁)**: 一组进程因互相等待对方持有的资源而全部阻塞, 无法继续执行。
5. **Deadlock Avoidance (死锁避免)**: 动态允许更多资源请求, 但通过检查确保系统不进入不安全状态。
6. **Deadlock Detection (死锁检测)**: 允许死锁发生, 但周期性检查并采取措施恢复。
7. **Deadlock Prevention (死锁预防)**: 静态地破坏死锁产生的四个必要条件之一。
8. **Fatal Region (致命区域)**: (在联合进度图中) 进程进入该区域后, 必然会导致死锁。
9. **Hold and Wait (占有且等待)**: 进程持有一个资源的同时等待其他资源。

10. **Joint Progress Diagram (联合进度图)**: 用于可视化两个进程对资源需求和死锁状态的图表。
11. **Memory Barrier (内存屏障)**: 强制 CPU 对内存操作排序的指令，防止乱序执行导致并发问题。
12. **Pipe (管道)**: 循环缓冲区，用于进程间流式通信。
13. **Reusable Resource (可重用资源)**: 使用后不消失，可被释放给其他进程使用的资源（如 CPU、内存、I/O 通道）。
14. **Safe State (安全状态)**: 系统能找到一个进程执行序列，使所有进程都能顺利完成。
15. **Spinlock (自旋锁)**: 线程在等待锁时执行忙循环（自旋），不放弃 CPU。
16. **Unsafe State (不安全状态)**: 系统无法保证所有进程都能完成，可能（但不一定）发生死锁。

二、Review Questions (复习题答案)

- **6.1 举例说明可重用资源和消耗性资源。**
 - 答: **可重用**: 处理器、I/O 通道、主存、文件; **消耗性**: 中断、信号、消息、缓冲区数据。
- **6.2 死锁发生的三个必要条件是什么?**
 - 答: (1) 互斥 (Mutual Exclusion); (2) 占有且等待 (Hold and Wait); (3) 不可抢占 (No Preemption)。(注: 这也是死锁存在的条件, 加上循环等待即为充要条件)。
- **6.3 产生死锁的四个条件是什么?**
 - 答: 上述三个条件 + (4) 循环等待 (Circular Wait)。
- **6.4 为了预防死锁, 如何禁止“占有且等待”条件?**
 - 答: 要求进程在开始执行前一次性申请并获得所有需要的资源。
- **6.5 为了预防死锁, 为何不能禁止“互斥”条件?**
 - 答: 因为某些资源（如打印机、写文件）本质上就是不可共享的, 必须互斥访问。
- **6.6 为了预防死锁, 如何防止“循环等待”?**
 - 答: 定义资源的线性顺序, 要求进程必须按序号递增的顺序申请资源。
- **6.7 列出一些从死锁中恢复的方法。**
 - 答: (1) 终止所有死锁进程; (2) 回滚每个死锁进程到检查点; (3) 逐个终止死锁进程直到死锁消失; (4) 逐个抢占资源直到死锁消失。

第 7 章: 内存管理 (Memory Management)

参考来源:

一、Key Terms (名词解释)

1. **Absolute Loading (绝对装入)**: 编译时确定程序在内存中的物理地址, 装入时直接放入指定位置, 不可移动。
2. **Buddy System (伙伴系统)**: 一种动态分区算法, 内存块的大小均为 2^k 。分配时将大块平分, 释放时检查相邻的“伙伴”块是否空闲, 若是则合并。
3. **Compaction (紧凑/压缩)**: 移动内存中的进程, 将分散的小空闲区合并成一个大的连续空闲区, 以解决外部碎片问题。
4. **Dynamic Linking (动态链接)**: 在程序运行时才进行模块链接, 而不是在编译或装入时。
5. **Dynamic Partitioning (动态分区)**: 进程装入时, 根据其所需大小动态建立一个分区, 分区大小等于进程大小。

6. **Dynamic Run-time Loading (动态运行时装入):** 地址转换推迟到程序真正执行时才进行 (通常需要硬件重定位寄存器支持)。
7. **External Fragmentation (外部碎片):** 内存中存在许多未被使用的小空闲块, 总和虽大但因不连续而无法容纳新进程。
8. **Fixed Partitioning (固定分区):** 系统启动时将内存划分为若干固定大小的分区 (可以相等也可以不等), 每个分区只能装入一个进程。
9. **Frame (页帧/帧):** 主存中固定长度的块, 用于存放页面。
10. **Internal Fragmentation (内部碎片):** 分配给进程的内存空间大于进程实际所需, 导致分区内部有未被利用的空间。
11. **Linkage Editor (链接编辑器):** 将目标模块链接成一个完整的加载模块的程序。
12. **Logical Address (逻辑地址):** 程序中生成的地址 (相对于 0 开始), 与物理硬件地址无关。
13. **Logical Organization (逻辑组织):** 内存按用户可见的模块 (如函数、数据段) 进行组织, 即分段。
14. **Memory Management (内存管理):** 操作系统负责动态分配、回收内存, 并提供地址转换和保护的功能。
15. **Page (页):** 进程逻辑地址空间中划分出的固定大小的块。
16. **Page Table (页表):** 记录进程的页与主存中页帧对应关系的数据结构。
17. **Paging (分页):** 将进程划分为固定大小的页, 内存划分为同大小的帧, 以离散方式装入内存的技术。
18. **Partitioning (分区):** 将内存划分为不同区域以容纳多个进程。
19. **Physical Address (物理地址):** 内存中实际的绝对地址。
20. **Physical Organization (物理组织):** 内存与辅存之间的数据流组织。
21. **Protection (保护):** 防止一个进程非法访问其他进程或操作系统的内存区域。
22. **Relative Address (相对地址):** 相对于某个基址 (如程序起始点) 的地址。
23. **Relocatable Loading (可重定位装入):** 装入时根据内存情况确定物理地址, 通常加上一个基址偏移量。
24. **Relocation (重定位):** 将逻辑地址转换为物理地址的过程。
25. **Segment (段):** 逻辑地址空间中根据逻辑含义 (如代码、栈、数据) 划分的变长块。
26. **Segmentation (分段):** 将程序和数据划分为不同长度的段进行管理。
27. **Sharing (共享):** 多个进程访问内存中的同一个副本 (如共享代码库)。

二、Review Questions (复习题答案)

- **7.1 内存管理旨在满足哪些需求?**
 - 答: 重定位、保护、共享、逻辑组织 (分段)、物理组织 (主存-辅存移动)。
- **7.2 什么是程序的重定位 (Relocation)?**
 - 答: 由于不知程序将被加载到内存何处, 处理器需要将程序中的相对地址动态映射到实际物理内存地址的能力。
- **7.3 将程序和数据组织成模块有什么优点?**
 - 答: 模块可以独立编写和编译; 可以提供不同程度的保护 (只读/执行); 便于共享。
- **7.4 允许两个或多个进程访问特定内存区域的原因是什么?**
 - 答: 为了允许合作进程共享数据结构或共享同一份程序代码 (如编辑器), 节省内存。

- **7.5 在固定分区方案中，使用不等大分区的优点是什么？**
 - 答：可以容纳非常大的进程（在大分区中），同时减少小进程在过大分区中造成的内部碎片。
 - **7.6 内部碎片和外部碎片的区别是什么？**
 - 答：**内部碎片**在分区内部（已分配但未用完）；**外部碎片**在分区之间（未分配但太小无法使用）。
 - **7.7 什么是地址绑定 (Address Binding)？**
 - 答：将符号地址（代码中的变量名）或逻辑地址映射到物理地址的过程。
 - **7.8 页 (Page) 和帧 (Frame) 的区别是什么？**
 - 答：**页**是逻辑地址空间（进程）的块；**帧**是物理内存的块。它们大小相同。
 - **7.9 页 (Page) 和段 (Segment) 的区别是什么？**
 - 答：**页**大小固定，对用户透明，用于物理内存管理（消除外部碎片）；**段**大小可变，由用户/编译器定义，用于逻辑组织和保护。
-

第 8 章：虚拟内存 (Virtual Memory)

参考来源：

一、Key Terms (名词解释)

1. **Associative Mapping (全相联映射)**: TLB 中的一种查找方式，可以同时比较所有条目以找到匹配的页号。
2. **Demand Paging (请求调页)**: 只有当页面被访问（需要）时，才将其从磁盘调入内存。
3. **Fetch Policy (读取策略)**: 决定何时将页面调入内存（如请求调页 vs 预调页）。
4. **Locality (局部性)**: 程序执行时倾向于在短时间内访问局部的内存区域（时间局部性和空间局部性）。
5. **Page Fault (缺页中断)**: 访问的页面不在主存中时产生的异常，通知 OS 调入该页。
6. **Page Replacement Policy (页面置换策略)**: 当内存满时，决定淘汰哪一页的算法（如 LRU, FIFO）。
7. **Prepaging (预调页)**: 在页面被访问前就预测并调入内存（通常用于进程启动时）。
8. **Resident Set (驻留集)**: 进程当前在主存中实际拥有的页面集合。
9. **Resident Set Management (驻留集管理)**: 决定给每个进程分配多少个页帧以及如何调整大小的策略。
10. **Thrashing (抖动/颠簸)**: 进程频繁进行缺页中断，导致 CPU 主要忙于换页而非执行指令，系统吞吐量急剧下降。
11. **Translation Lookaside Buffer (TLB, 快表)**: 内存管理单元 (MMU) 中的高速缓存，存储最近使用的页表项以加速地址转换。
12. **Virtual Memory (虚拟内存)**: 允许程序使用的内存空间大于物理内存，通过将部分程序放在磁盘上实现。
13. **Working Set (工作集)**: 进程在某段时间内频繁访问的页面集合。确保工作集在内存中可防止抖动。

二、Review Questions (复习题答案)

- **8.1 虚拟内存的使用如何提高系统利用率?**
 - 答: 允许内存中驻留更多进程 (因为每个进程只需部分装入), 从而增加 CPU 忙碌的概率; 允许运行比物理内存更大的进程。
 - **8.2 解释抖动 (Thrashing)。**
 - 答: 当系统过度承诺内存 (多道程序度过高), 导致频繁缺页, 大部分时间花在 I/O 换页上, CPU 利用率崩溃。
 - **8.3 为什么局部性原理对虚拟内存至关重要?**
 - 答: 如果没有局部性, 程序会随机访问所有页面, 导致持续的缺页中断, 虚拟内存将无法高效工作。
 - **8.4 哪些因素决定了页面的大小?**
 - 答: 页表的大小 (页越小页表越大)、内部碎片 (页越小碎片越小)、磁盘 I/O 效率 (大页传输更有效)。
 - **8.5 TLB 的目的是什么?**
 - 答: 缓存最近的逻辑-物理地址映射, 避免每次地址转换都要访问两次主存 (一次查页表, 一次取数据)。
 - **8.6 请求调页的目的是什么?**
 - 答: 仅在需要时加载页面, 节省内存, 启动快。
 - **8.7 单纯使用请求调页或预调页的缺点是什么?**
 - 答: **纯请求调页:** 进程启动初期会有大量缺页; **纯预调页:** 可能调入不使用的页面, 浪费 I/O。
 - **8.8 FIFO 和时钟 (Clock) 置换算法的关系?**
 - 答: Clock 算法是 FIFO 的改进版。它像 FIFO 一样循环扫描, 但通过“使用位”给页面第二次机会 (类似 LRU 的近似), 避免置换最近常用的页面。
 - **8.9 缺页中断是如何处理的?**
 - 答: 陷入内核 -> 保存状态 -> 查找磁盘地址 -> 读入页面 -> (若满则置换) -> 更新页表 -> 恢复进程。
 - **8.10 为什么不能将全局置换策略与固定分配策略结合使用?**
 - 答: 固定分配意味着进程的帧数固定, 而全局置换会 from 其他进程抢夺帧, 这与“固定”帧数定义冲突。
 - **8.11 驻留集 (Resident Set) 和工作集 (Working Set) 的区别?**
 - 答: **驻留集**是当前实际在内存中的页面; **工作集**是理论上进程在当前阶段为了高效运行“需要”的页面集合。
 - **8.12 请求清理 (Demand Cleaning) 和预清理 (Precleaning) 的区别?**
 - 答: **请求清理:** 仅在页面被置换且被修改过时才写回磁盘 (会拖慢置换过程); **预清理:** 批量异步写回脏页, 使置换时页面是干净的 (可直接覆盖)。
-

第 9 章：单处理器调度 (Uniprocessor Scheduling)

参考来源： ,

一、Key Terms (名词解释)

1. **Arrival Rate (到达率)**: 单位时间内创建或进入系统的进程数量。
2. **Dispatcher (分派器)**: (同第3章) 负责上下文切换的模块。
3. **Fair-Share Scheduling (公平共享调度)**: 根据用户组或所有者而非单个进程来分配 CPU 时间, 确保特定用户组获得指定比例的 CPU。
4. **FCFS (先来先服务)**: 非抢占策略, 按进程请求 CPU 的顺序调度。
5. **Long-Term Scheduler (长程调度/作业调度)**: 决定将哪些程序调入系统成为进程 (控制多道程序度) 。
6. **Medium-Term Scheduler (中程调度)**: 决定将哪些进程换入/换出内存 (交换功能) 。
7. **Multilevel Feedback (多级反馈队列)**: 进程在不同优先级的队列间移动。通常时间片用完降级, 用于兼顾响应时间和吞吐量。
8. **Response Time (响应时间)**: 从提交请求到产生第一个响应的时间 (对交互式系统重要) 。
9. **Round Robin (轮转)**: 抢占式, 每个进程分一个时间片, 轮流执行。
10. **Short-Term Scheduler (短程调度/CPU 调度)**: 决定下一个在 CPU 上执行哪个就绪进程 (执行频率最高) 。
11. **Throughput (吞吐量)**: 单位时间内完成的进程数量。
12. **Turnaround Time (TAT, 周转时间)**: 进程从提交到完成的总时间。
13. **Waiting Time (等待时间)**: 进程在就绪队列中等待 CPU 的总时间。

二、Review Questions (复习题答案)

- **9.1 批处理系统的处理器调度是如何完成的?**
 - 答: 通常使用长程调度 (作业进入) 配合短程调度 (FCFS 或 SJF) 来最大化吞吐量。
- **9.2 分派器 (Dispatcher) 的主要功能? 举例说明触发事件。**
 - 答: 功能: 将 CPU 控制权交给调度程序选出的进程。触发事件: 时钟中断、I/O 中断、系统调用。
- **9.3 调度标准如何影响系统性能?**
 - 答: 优化一个标准 (如响应时间) 通常会牺牲另一个标准 (如吞吐量或公平性) 。
- **9.4 如果纯粹使用基于优先级的调度, 会面临什么问题?**
 - 答: **饥饿 (Starvation)**。低优先级进程可能永远得不到执行。
- **9.5 抢占式调度的优缺点?**
 - 答: **优**: 防止一个进程独占 CPU, 响应快; **缺**: 上下文切换开销大, 可能导致共享数据竞争问题。
- **9.6 简述 FCFS。**
 - 答: 按到达顺序执行, 非抢占。对短进程不利 (护航效应) 。
- **9.7 简述轮转调度 (Round-Robin)。**
 - 答: 每个进程分配时间片, 轮流执行。对交互式系统友好, 但上下文切换多。
- **9.8 简述最短进程优先 (SPN/SJF)。**

- 答：选择预计执行时间最短的进程。平均周转时间最优，但长进程可能饥饿。
 - 9.9 简述最短剩余时间优先 (SRT)。
 - 答：SJF 的抢占版。新进程更短则抢占当前进程。
 - 9.10 简述最高响应比优先 (HRRN)。
 - 答：优先级 = (等待时间+服务时间)/服务时间。既照顾短进程，又随等待时间增加照顾长进程（防饥饿）。
 - 9.11 简述反馈调度 (Feedback)。
 - 答：基于动态优先级的多队列。不知道进程长度时，先给高优先级短时间片，用完则降级。
-

第 11 章：I/O 管理与磁盘调度 (I/O Management and Disk Scheduling)

参考来源：

一、Key Terms (名词解释)

1. **Block-Oriented Device (块设备)**: 以固定大小的数据块进行传输和寻址的设备（如磁盘）。
2. **Circular Buffer (循环缓冲)**: 内存中的环形区域，用于平滑 I/O 生产和消费速度的差异。
3. **DMA (直接内存存取)**: I/O 模块直接与内存交换数据，无需 CPU 介入每字节的传输。
4. **Double Buffering (双缓冲)**: 使用两个缓冲区，一个设备在写，CPU 处理另一个，交替进行，提高并行性。
5. **Disk Access Time (磁盘访问时间)**: 寻道时间 + 旋转延迟 + 传输时间。
6. **Interrupt-driven I/O (中断驱动 I/O)**: CPU 发出 I/O 命令后继续做其他事，I/O 完成后发中断通知 CPU。
7. **Programmed I/O (程序控制 I/O / 轮询)**: CPU 不断查询 I/O 设备状态，直到完成。浪费 CPU。
8. **RAID (磁盘冗余阵列)**: 使用多个磁盘通过并行和冗余来提高性能和可靠性。
9. **Rotational Delay (旋转延迟)**: 磁盘旋转到指定扇区位于磁头下方所需的时间。
10. **Seek Time (寻道时间)**: 磁头移动到指定磁道（柱面）所需的时间（通常最耗时）。
11. **Stream-Oriented Device (字符/流设备)**: 以字节流方式传输数据的设备，无块结构（如键盘、鼠标、串口）。
12. **Transfer Time (传输时间)**: 数据从磁盘读出并写入内存所需的时间。

二、Review Questions (复习题答案)

- 11.1 简述三种 I/O 技术。
 - 答：(1) **程序控制 I/O**: 忙等轮询；(2) **中断驱动 I/O**: CPU 发命令后异步等待中断；(3) **DMA**: 专用模块负责内存传输，CPU 仅在开始和结束参与。
- 11.2 块设备与流设备的区别？
 - 答：**块设备**: 存取固定块，可寻址（磁盘）；**流设备**: 存取字节流，不可寻址（键盘）。
- 11.3 为什么要对 I/O 使用双缓冲？
 - 答：允许数据传输（设备写缓冲区）和数据处理（CPU 读缓冲区）并行进行，平滑速度差异。
- 11.4 相比单缓冲，双缓冲有什么性能提升？
 - 答：它可以解耦生产者和消费者，使得只要平均速度匹配，就能消除等待时间。

- **11.5 缓冲的一些用途？**
 - 答：平滑速度差异、协调数据传输大小不一致（如网络包拆分）、支持应用程序语义（如拷贝语义）。
 - **11.6 简述图 11.7 (书) 中的磁盘调度策略。**
 - 答：(通常指 SSTF, SCAN, C-SCAN) **SSTF**: 找最近的磁道（可能饥饿）；**SCAN (电梯)**: 单向移动扫到底再回头；**C-SCAN**: 单向移动，回此时直接复位不服务。
 - **11.7 硬件 RAID 和软件 RAID 的区别？**
 - 答：**硬件 RAID**: 专用控制卡处理，不占 CPU，性能高；**软件 RAID**: OS 处理，占 CPU，成本低。
 - **11.8 什么是 Linux 电梯调度？**
 - 答：Linux 的磁盘调度程序（现多用 Deadline 或 CFQ），维护一个有序队列，类似电梯算法合并请求以减少寻道。
-

第 12 章：文件管理 (File Management)

参考来源：，

一、Key Terms (名词解释)

1. **Chained File Allocation (链接分配)**: 文件块以链表形式存储，每个块指向下一个块。无外部碎片，但不支持随机访问。
2. **Contiguous File Allocation (连续分配)**: 文件占用磁盘上连续的块。读写极快，但有外部碎片且文件难以扩展。
3. **Directory (目录)**: 包含文件信息（文件名、属性、位置）的特殊文件。
4. **File Allocation Table (FAT, 文件分配表)**: 将链接指针集中存放在内存中的一张表中（而非块中），用于加速链接分配的查找。
5. **Indexed File Allocation (索引分配)**: 为每个文件分配一个索引块，其中列出所有数据块的地址。支持直接访问且无外部碎片。
6. **Inode (索引节点)**: UNIX/Linux 中的文件控制结构，包含文件属性和指向数据块的指针索引。
7. **Pathname (路径名)**: 从根目录或当前目录到文件的路径字符串。
8. **Record (记录)**: 文件内部的一组相关数据字段（逻辑单位）。
9. **Sequential File (顺序文件)**: 记录按物理顺序排列，必须从头读到尾。
10. **Virtual File System (VFS, 虚拟文件系统)**: 操作系统中的抽象层，为不同文件系统提供统一接口，使用户操作透明化。

二、Review Questions (复习题答案)

- **12.1 文件系统的理想属性？**
 - 答：长期存储、进程间共享、结构化管理、命名方便。
- **12.2 文件和数据库的区别？**
 - 答：**文件**: 基础存储，OS 管理字节/记录；**数据库**: 高级应用，管理数据关系、查询和事务。
- **12.3 文件管理系统有哪些主要标准？**
 - 答：访问速度、易用性、存储效率、数据完整性。
- **12.4 选择文件组织方式的优缺点？**

- 答：**顺序**: 适合批处理，不适合随机查；**索引**: 随机查快，但空间开销大。
- 12.5 顺序文件 vs 索引顺序文件查找时间的差异?
 - 答：顺序文件是 $O(N)$ ；索引顺序文件是 $O(\sqrt{N})$ 或 $O(\log N)$ （取决于索引层级）。
- 12.6 路径名 (Pathname) 是什么? 两种指定方式?
 - 答：**绝对路径**: 从根目录 (/) 开始；**相对路径**: 从当前工作目录 (.) 开始。
- 12.7 路径名与工作目录的关系?
 - 答：相对路径名是基于工作目录解析的。
- 12.9 (见图9269) 一般给特定文件的访问权限有哪些?
 - 答：读 (Read)、写 (Write)、执行 (Execute)、删除 (Delete)、追加 (Append)。
- 12.10 UNIX 中的 inode 是什么?
 - 答：是 UNIX 文件系统的核心结构，存储文件的元数据（权限、所有者、时间戳）和数据块指针（直接、间接指针）。
- 12.11 简述三种文件分配方法。
 - 答：(1) **连续分配**: 连续块，快但有碎片；(2) **链接分配**: 链表，无碎片但随机慢；(3) **索引分配**: 索引块存指针，支持随机且无碎片。