syst八股文

Monday, September 30, 2024 10:14 PM

Day10

进程和线程的区别

掌握程度：

- 理解进程和线程在资源拥有、执行、上下文切换、创建和销毁开销方面的差异， 能够清晰地描述进程和线程的定义、区别和联系。
- 知道进程间通信和线程间通信的不同方式及其特点。
- 如果有实际的多线程或多进程编程经验，能够结合实例讨论进程和线程的使用和最佳实践。

参考资料：

【操作系统】进程和线程的区别
进程和线程的比较

进程是资源分配和调度的基本单位。
线程是程序执行的最小单位，线程是进程的子任务，是进程内的执行单元。 一个进程至少有一个线程，一个进程可以运行多个线程，这些线程共享同一块内存。

资源开销：

- 进程：由于每个进程都有独立的内存空间，创建和销毁进程的开销较大。进程间切换需要保存和恢复整个进程的状态，因此上下文切换的开销较高。
- 线程：线程共享相同的内存空间，创建和销毁线程的开销较小。线程间切换只需要保存和恢复少量的线程上下文，因此上下文切换的开销较小。

通信与同步：

- 进程：由于进程间相互隔离，进程之间的通信需要使用一些特殊机制，如管道、消息队列、共享内存等。
- 线程：由于线程共享相同的内存空间，它们之间可以直接访问共享数据，线程间通信更加方便。

安全性：

- 进程：由于进程间相互隔离，一个进程的崩溃不会直接影响其他进程的稳定性。
- 线程：由于线程共享相同的内存空间，一个线程的错误可能会影响整个进程的稳定性。

并行和并发有什么区别

掌握程度：

- 能够清晰地描述并发和并行的定义、区别和联系。
- 理解并发和并行在实现、性能、应用场景和复杂性方面的差异。

参考资料：
【一个视频告诉你“并发、并行、异步、同步”的区别】

- 并行是在同一时刻执行多个任务。
- 并发是在相同的时间段内执行多个任务，任务可能交替执行，通过调度实现。
  并行是指在同一时刻执行多个任务，这些任务可以同时进行，每个任务都在不同的处理单元（如多个CPU核心）上执行。在并行系统中，多个处理单元可以同时处理独立的子任务，从而加速整体任务的完成。
  并发是指在相同的时间段内执行多个任务，这些任务可能不是同时发生的，而是交替执行，通过时间片轮转或者事件驱动的方式。并发通常与任务之间的交替执行和任务调度有关。

解释一下用户态和核心态

掌握程度：

- 能够清晰地描述用户态和核心态的定义、区别和它们在操作系统中的角色。
- 理解用户态和核心态在特权级别、系统调用、安全性和性能方面的差异。

1. 用户态和内核态的区别
   用户态和内核态是操作系统为了保护系统资源和实现权限控制而设计的两种不同的CPU运行级别，可以**控制进程或程序对计算机硬件资源的访问权限和操作范围。**
- 用户态：在用户态下，进程或程序只能访问受限的资源和执行受限的指令集，不能直接访问操作系统的核心部分，也不能直接访问硬件资源。
- 核心态：核心态是操作系统的特权级别，允许进程或程序执行特权指令和访问操作系统的核心部分。在核心态下，进程可以直接访问硬件资源，执行系统调用，管理内存、文件系统等操作。

2. 在什么场景下，会发生内核态和用户态的切换

- 系统调用：当用户程序需要请求操作系统提供的服务时，会通过系统调用进入内核态。
- 异常：当程序执行过程中出现错误或异常情况时，CPU会自动切换到内核态，以便操作系统能够处理这些异常。
- 中断：外部设备（如键盘、鼠标、磁盘等）产生的中断信号会使CPU从用户态切换到内核态。操作系统会处理这些中断，执行相应的中断处理程序，然后再将CPU切换回用户态。

Day11

进程调度算法你了解多少

掌握程度 ：

- 能够清晰地描述至少几种常见的进程调度算法及其特点。
- 知道在不同应用场景下如何选择适合的调度算法。
  参考资料：
  进程调度算法
  【操作系统】CPU调度算法】

- 先来先服务：按照请求的顺序进行调度。 这种调度方式简单，但是能导致较长作业阻塞较短作业。

- 最短作业优先：非抢占式的调度算法，按估计运行时间最短的顺序进行调度。 但是如果一直有短作业到来，那么长作业永远得不到调度，造成长作业 饥饿 现象。
- 最短剩余时间优先：基于最短作业优先改进，按剩余运行时间的顺序进行调度。当一个新的作业到达时，其整个运行时间与当前进程的剩余时间作比较。如果新的进程需要的时间更少，则挂起当前进程，运行新的进程。否则新的进程等待。
- 优先级调度：为每个进程分配一个优先级，按优先级进行调度。为了防止低优先级的进程永远等不到调度，可以随着时间的推移增加等待进程的优先级。
- 时间片轮转：为每个进程分配一个时间片，进程轮流执行，时间片用完后切换到下一个进程。
- 多级队列：时间片轮转调度算法和优先级调度算法的结合。 将进程分为不同的优先级队列，每个队列有自己的调度算法。

进程间有哪些通信方式

面试中的掌握程度：

- 能够清晰地描述至少几种常见的进程间通信方式及其特点。
- 知道进程间通信在多任务操作系统中的应用，理解不同IPC机制的使用场景和限制。
- 了解进程间通信可能带来的安全问题，如竞态条件、死锁等，并讨论如何避免这些问题。

参考资料：

小林coding: 进程通信方式

【大厂面试笔试题31丨进程间通信方式】

『面试问答』：进程间通信的方式有哪些？

3. 管道：是一种半双工的通信方式，数据只能单向流动而且只能在具有父子进程关系的进程间使用。
4. 命名管道： 类似管道，也是半双工的通信方式，但是它允许在不相关的进程间通信。
5. 消息队列：允许进程发送和接收消息，而消息队列是消息的链表，可以设置消息优先级。
6. 信号：用于发送通知到进程，告知其发生了某种事件或条件。
7. 信号量：是一个计数器，可以用来控制多个进程对共享资源的访问，常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此主要作为进程间以及同一进程内不同线程之间的同步手段。
8. 共享内存：就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的进程通信方式，
9. Socket套接字：是支持TCP/IP 的网络通信的基本操作单元，主要用于在客户端和服务器之间通过网络进行通信。

解释一下进程同步和互斥，以及如何实现进程同步和互斥

掌握程度：

- 能够清晰地描述进程同步和互斥的定义、目的和区别。
- **互斥锁**：在访问共享资源前，进程必须先获取互斥锁，访问后再释放锁。
- **信号量**：通过P（等待）和V（信号）操作来控制对共享资源的访问。
- 理解不同同步和互斥机制的工作原理和适用场景。
- 知道如何使用常见的同步和互斥工具（如互斥锁、信号量、条件变量等）来实现进程同步和互斥。

参考资料：

小林coding: 进程同步与互斥

进程同步是指**多个并发执行的进程之间协调和管理它们的执行顺序，以确保它们按照一定的顺序或时间间隔执行。**

互斥指的是在某一时刻只允许一个进程访问某个共享资源。当一个进程正在使用共享资源时，其他进程不能同时访问该资源。

解决进程同步和互斥的问题有很多种方法，其中一种常见的方法是使用**信号量和 PV 操作**。信号量是一种特殊的变量，它表示系统中某种资源的数量或者状态。PV 操作是一种对信号量进行增加或者减少

的操作，它们可以用来控制进程之间的同步或者互斥。

- **P操作**：相当于"检查"信号量，如果资源可用，就减少计数，然后使用资源。
- **V操作**：相当于"归还"资源，增加信号量的计数，并可能唤醒等待的进程。

除此之外，下面的方法也可以解决进程同步和互斥问题：

- **临界区**：将可能引发互斥问题的代码段称为临界区，里面包含了需要互斥访问的资源。进入这个区域前需要先获取锁，退出临界区后释放该锁。这确保同一时间只有一个进程可以进入临界区。
- **互斥锁（Mutex）**：互斥锁是一种同步机制，用于实现互斥。每个共享资源都关联一个互斥锁，进程在访问该资源前需要先获取互斥锁，使用完后释放锁。只有获得锁的进程才能访问共享资源。
- **条件变量**：条件变量用于在进程之间传递信息，以便它们在特定条件下等待或唤醒。通常与互斥锁一起使用，以确保等待和唤醒的操作在正确的时机执行。

Day12

什么是死锁，如何预防死锁？

掌握程度：

- 理解死锁的危害，能够清晰地描述死锁的定义和四个必要条件。
- 知道并能够解释常见的死锁预防策略。
- 能够讨论死锁检测和恢复的机制。
- 结合案例和并发编程讨论死锁的预防和处理。

参考资料：

【面试问答：什么是死锁？，如何预防死锁】
小林coding: 如何避免死锁

死锁是系统中两个或多个进程在执行过程中，因争夺资源而造成的一种僵局。当每个进程都持有一定的资源并等待其他进程释放它们所需的资源时，如果这些资源都被其他进程占有且不释放，就导致了死锁。

死锁只有同时满足以下四个条件才会发生：

- 互斥条件：一个进程占用了某个资源时，其他进程无法同时占用该资源。
- 请求保持条件：一个进程因为请求资源而阻塞的时候，不会释放自己的资源。
- 不可剥夺条件：资源不能被强制性地从一个进程中剥夺，只能由持有者自愿释放。
- 循环等待条件：多个进程之间形成一个循环等待资源的链，每个进程都在等待下一个进程所占有的资源。

避免死锁：**通过破坏死锁的四个必要条件之一来预防死锁。比如破坏循环等待条件，让所有进程按照相同的顺序请求资源。** 检测死锁：通过检测系统中的资源分配情况来判断是否存在死锁。例如，可以使用资源分配图或银行家算法进行检测。 解除死锁：一旦检测到死锁存在，可以采取一些措施来解除死锁。例如，可以通过抢占资源、终止某些进程或进行资源回收等方式来解除死锁

介绍一下几种典型的锁

掌握程度：

- 够清晰地描述每种锁的基本概念和用途。
- 理解不同锁机制的适用场景和优缺点。
- 知道如何根据实际需求选择合适的锁机制。
- 了解不同锁机制对性能的影响，并能够讨论如何优化锁的使用。
- 结合并发编程讨论锁的应用。

参考资料：
小林coding: 什么是悲观锁，乐观锁

- **互斥锁**：互斥锁是一种最常见的锁类型，用于实现互斥访问共享资源。在任何时刻，只有一个线程可以持有互斥锁，其他线程必须等待直到锁被释放。这确保了同一时间只有一个线程能够访问被保护的资源。
- 自旋锁：自旋锁是一种基于忙等待的锁，即线程在尝试获取锁时会不断轮询，直到锁被释放。
  其他的锁都是基于这两个锁的
- 读写锁：允许多个线程同时读共享资源，只允许一个线程进行写操作。分为读（共享）和写（排他）两种状态。
- 悲观锁：认为多线程同时修改共享资源的概率比较高，所以访问共享资源时候要上锁
- 乐观锁：先不管，修改了共享资源再说，如果出现同时修改的情况，再放弃本次操作。

讲一讲你理解的虚拟内存

掌握程度：

- 能够清晰地描述虚拟内存的定义和工作原理。
- 理解虚拟内存如何通过页置换算法来管理内存。
- 知道虚拟内存的优点和可能带来的性能问题。

参考资料：
【操作系统】内存管理——虚拟内存】
小林coding: 虚拟内存

虚拟内存是指在每一个进程创建加载的过程中，会分配一个连续虚拟地址空间，**它不是真实存在的，而是通过映射与实际物理地址空间**对应，这样就可以使每个进程看起来都有自己独立的连续地址空间，并允许程序访问比物理内存RAM更大的地址空间，每个程序都可以认为它拥有足够的内存来运行。
需要虚拟内存的原因：

- 内存扩展： 虚拟内存使得每个程序都可以使用比实际可用内存更多的内存，从而允许运行更大的程序或处理更多的数据。
- 内存隔离：虚拟内存还提供了进程之间的内存隔离。每个进程都有自己的虚拟地址空间，因此一个进程无法直接访问另一个进程的内存。
- 物理内存管理：虚拟内存允许操作系统动态地将数据和程序的部分加载到物理内存中，以满足当前正在运行的进程的需求。当物理内存不足时，操作系统可以将不常用的数据或程序暂时移到硬盘上，从而释放内存，以便其他进程使用。
- 页面交换：当物理内存不足时，操作系统可以将一部分数据从物理内存写入到硬盘的虚拟内存中，这个过程被称为页面交换。当需要时，数据可以再次从虚拟内存中加载到物理内存中。这样可以保证系统可以继续运行，尽管物理内存有限。
- 内存映射文件：虚拟内存还可以用于将文件映射到内存中，这使得文件的读取和写入可以像访问内存一样高效。

Day13

你知道的线程同步的方式有哪些？

掌握程度：

- 清晰地描述每种线程同步机制的基本概念和用途。
- 互斥锁

- 自旋锁
- 读写锁
- 条件变量
- 信号量
- 理解不同同步机制的适用场景和优缺点。
- 知道如何根据实际需求选择合适的线程同步机制。

参考资料：

【多线程编程：一次性搞懂线程同步机制】

【『面试问答』：线程间同步方式有哪些？】

线程同步机制是指在多线程编程中，为了保证线程之间的互不干扰，而采用的一种机制。常见的线程同步机制有以下几种：

1. 互斥锁：互斥锁是最常见的线程同步机制。它允许只有一个线程同时访问被保护的临界区（共享资源）
2. 条件变量：条件变量用于线程间通信，允许一个线程等待某个条件满足，而其他线程可以发出信号通知等待线程。通常与互斥锁一起使用。
3. 读写锁： 读写锁允许多个线程同时读取共享资源，但只允许一个线程写入资源。
4. 信号量：用于控制多个线程对共享资源进行访问的工具。

有哪些页面置换算法

掌握程度：

- 能够清晰地描述至少几种常见的页面置换算法及其特点。
- 理解不同页面置换算法的工作原理和适用场景。

参考资料：

小林coding: 页面置换算法

【操作系统】10分钟-速解页面置换算法】

常见页面置换算法有最佳置换算法（OPT）、先进先出（FIFO）、最近最久未使用算法（LRU）、时钟算法（Clock）等。

1. 最近最久未使用算法LRU：LRU算法**基于页面的使用历史，通过选择最长时间未被使用的页面进行置换。**
2. 先进先出FIFO算法：也就是**最先进入内存的页面最先被置换出去**。
3. 最不经常使用LFU：淘汰访问次数最少的页面，考虑页面的访问频率。
4. 时钟算法CLOCK：Clock算法的核心思想是通过使用一个指针(称为时钟指针)在环形链表上遍历，检查页面是否被访问过，当需要进行页面置换时，Clock算法从时钟指针的位置开始遍历环形链表。 如果当前页面的访问位为0，表示该页面最久未被访问，可以选择进行置换。将访问位设置为1，继续遍历下一个页面。 如果当前页面的访问位为1，表示该页面最近被访问过，它仍然处于活跃状态。将访问位设置为0，并继续遍历下一个页面如果遍历过程中找到一个访问位为0的页面，那么选择该页面进行置换。
5. 最佳置换算法: 该**算法根据未来的页面访问情况，选择最长时间内不会被访问到的页面进行置换**。那么就有一个问题了，未来要访问什么页面，操作系统怎么知道的呢?操作系统当然不会知道，所以这种算法只是一种理想情况下的置换算法，通常是无法实现的。

Day15

熟悉那些 Linux 命令

掌握程度：

- 掌握关于文件操作，文件内容查看、权限管理、网络管理、进程管理、磁盘管理、软件包管理的一些常用命令，能够清晰地描述每个命令的基本用途和用法。
- 理解不同命令在系统管理、文件操作、网络管理等方面的应用。
  10分钟视频：【常用的Linux命令介绍：13个基本命令和Shell脚本编程】

10. 文件操作：
    - ls：列出目录内容。
    - cd：改变当前目录。
    - pwd：显示当前工作目录。
    - cp：复制文件或目录。
    - mv：移动或重命名文件。
    - rm：删除文件或目录。
    - touch：创建空文件或更新文件时间戳。
11. **文件内容查看**：
    - cat：查看文件内容。
    - head：查看文件的前几行。
    - tail：查看文件的后几行，常用于查看日志文件。
12. **文件编辑**：
    - vi 或 vim：强大的文本编辑器。
13. **权限管理**：
    - chmod：更改文件或目录的访问权限。
    - chown：更改文件或目录的所有者和/或所属组。
14. **磁盘管理**：
    - df：查看磁盘空间使用情况。
15. **网络管理**：
    - ifconfig 或 ip addr：查看和配置网络接口。
    - ping：测试网络连接。
    - netstat：查看网络状态和统计信息。
    - ssh：安全远程登录。
16. **进程管理**：
    - ps：查看当前运行的进程。
    - kill：发送信号给进程。
17. **软件包管理**（根据Linux发行版不同，命令可能有所不同）：
    - apt-get（Debian/Ubuntu）：安装、更新和删除软件包。

如何查看某个端口有没有被占用

掌握程度：

- 能够清晰地描述至少一种方法来检查端口是否被占用。
- 理解不同工具的输出结果，知道如何根据输出判断端口的使用情况。
- 如果有实际的网络问题排查经验，能够结合案例讨论如何使用命令。

18. 查看进程： 用 ps 命令查看当前运行的进程，比如 ps aux 可以列出所有进程及其详细信息。
19. 杀死进程： 首先用 ps 或 top 命令找到进程的PID（进程ID）。 然后用 kill 命令加上进程ID来结束进程，例如 kill -9 PID。"-9" 是强制杀死进程的信号。
20. 查看端口占用： 使用 lsof -i:端口号 可以查看占用特定端口的进程。 或者用 netstat -tulnp | grep 端口号，这会显示监听在该端口的服务及其进程ID。

说一下 select、poll 和 epoll 【重点】

掌握程度：

- 能够清晰地描述select、poll和epoll的定义和工作原理。
- 理解它们在I/O多路复用中的应用和区别。
- 知道它们各自的限制和适用场景。

[Linux IO模式及 select、poll、epoll详解]
[推荐：深入学习IO多路复用 select/poll/epoll 实现原理]

I/O多路复用通常通过select、poll、epoll等系统调用来实现。

- **select：** select是一个最古老的I/O多路复用机制，它可以监视多个文件描述符的可读、可写和错误状态。然而，但是它的效率可能随着监视的文件描述符数量的增加而降低。
- **poll：** poll是select的一种改进，它使用**轮询方式**来检查多个文件描述符的状态，避免了select中文件描述符数量有限的问题。但对于大量的文件描述符，poll的性能也可能变得不足够高效。
- **epoll：** epoll是Linux特有的I/O多路复用机制，相较于select和poll，它在处理大量文件描述符时更加高效。epoll使用事件通知的方式，只有在文件描述符就绪时才会通知应用程序，而不需要应用程序轮询。

总结：select是最早的 I/O 多路复用技术，但受到文件描述符数量和效率方面的限制。poll克服了文件描述符数量的限制，但仍然存在一定的效率问题。epoll是一种高效的I/O多路复用技术，尤其适用于高并发场景，但它仅在 Linux 平台上可用。一般来说，epoll 的效率是要比 select 和 poll 高的，但是对于活动连接较多的时候，由于回调函数触发的很频繁，其效率不一定比 select 和 poll 高。所以 epoll 在连接数量很多，但活动连接较小的情况性能体现的比较明显。