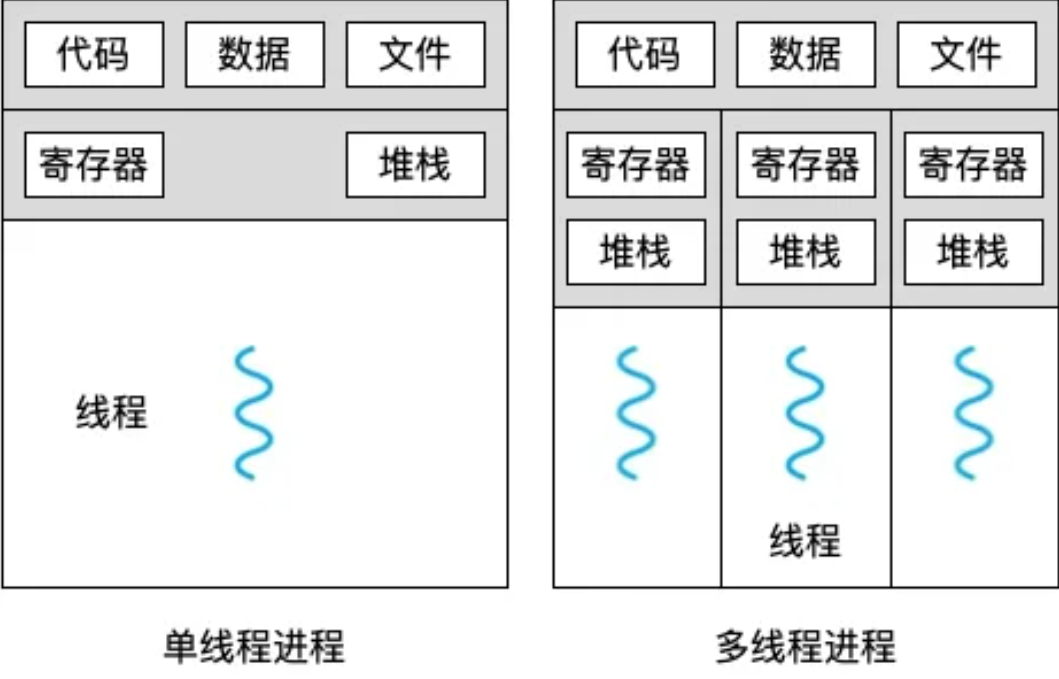
面试真题

【腾讯】进程和线程的区别



* **进程是操作系统资源分配的基本单位；**  
  线程是任务调度和执行的基本单位，是进程的一部分，也被称为轻量级进程。
* **进程有独立的上下文（代码、数据、文件），切换开销大；**  
  线程组内共享资源，但有独立的PC和堆栈
* **子进程崩溃不会影响其他进程；**但线程的崩溃可能导致整个进程崩溃。

【腾讯】为什么进程崩溃不会对其他进程产生很大影响

**进程运行独立、内存隔离**：进程间不会共享文件、网络连接等。进程崩溃时，其独立的内存空间会被操作系统回收。

面向简历

进程、线程、虚拟内存

1、进程

进程是操作系统中资源分配的基本单位，每个进程拥有独立的内存空间、文件句柄、I/O资源等系统资源。

由于进程之间独立，需要通过进程间通信（IPC）进行数据交互，如管道、共享内存、消息队列等。

2、线程

线程是操作系统中CPU调度的基本单位，一个进程可以包含多个线程，多个线程共享进程的资源，如内存地址空间、文件句柄等，每个线程都有自己的栈、程序计数器和寄存器。

可以通过共享内存、锁、信号量等方式实现线程通信。

线程安全是指多个线程同时访问共享资源时，程序的行为是可预期的，不会出现数据竞争问题。

如何解决线程安全问题？

1. 使用同步机制（如锁、信号量、原子操作等）来控制共享资源的访问。
2. 使用线程安全的集合类（如ConcurrentHashMap）。
3. 使用线程本地变量（ThreadLocal）。

3、虚拟内存

虚拟内存通过将内存划分为页和页表，将不常用的内存数据暂时存储到硬盘的交换空间中，当访问的虚拟地址不在物理内存中时，触发缺页中断，将数据从硬盘调入内存。

分页是按固定大小的内存块划分，分段是按逻辑功能划分。

进程间通信技术

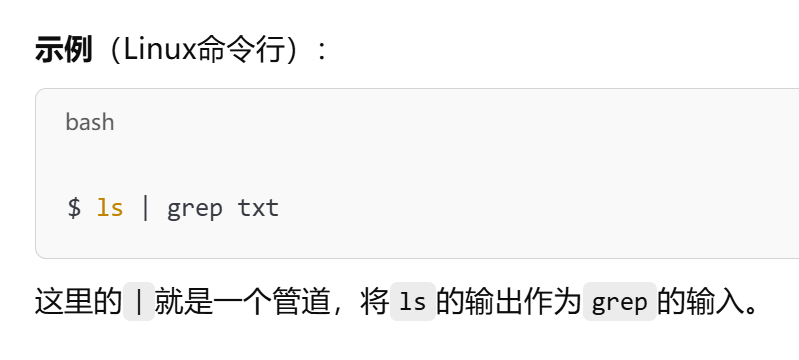
**进程间通信（Inter-Process Communication, IPC）**：由于每个进程都有自己独立的**内存空间**，因此需要**进程间通信**来实现数据的交互。

（1）管道（Pipe）

管道是一个**半双工**的通信机制，数据以**字节流**的形式**单向流动**。

简单快速，具有**阻塞**和**同步**的特性。

通常用于**父子进程之间的通信**。



1. 命名管道（FIFO）

命名管道是管道的扩展，**具有名字标识**。

支持**无亲缘关系**的进程之间通信，但仅限同一主机。

支持多对多通信。

可以通过**文件系统**访问。



（3）消息队列

允许多个进程通过**消息队列**发送和接收**结构化数据**。

支持**随机访问（不同于管道的FIFO顺序）、消息持久化**。

**适用于异步任务队列、日志系统**等。

（4）共享内存

共享内存是**速度最快**的进程间通信方式，不同进程可以**共享一段内存区域**，因为数据不需要在内核和用户空间之间来回拷贝。

需要**同步机制**（如信号量、互斥锁）来防止数据竞争。

**适用于高性能数据传输**的场景，如视频流、实时数据处理等。

（5）信号量（Semaphore）

信号量能实现**同步和互斥**，通常用于控制多个进程对**共享资源**的访问。

可以用于**解决死锁问题**。

**适用于**控制**共享资源的访问**，如数据库连接池、文件读写等。

（6）信号（Signal）

信号是用于**向进程发送异步通知**或**中断**的机制。

每个信号都有**固定的编号**，如SIGKILL、SIGSTOP等，数量有限、可能丢失。



（7）套接字（Socket）

套接字支持**分布式**通信，使用**网络协议**（如TCP、UDP）来传输数据，可以在**不同主机**进程之间进行通信。

**适用于网络通信**场景，如客户端-服务器模型、微服务之间的通信。

多路复用技术（Multiplexing）

1.核心概念

通过**单个线程或进程**同时监视多个I/O事件。

2.应用场景

1. 网络服务器监视多个客户端连接
2. 文件系统监听多个文件的读写状态
3. 同一线程中异步非阻塞处理多个I/O任务
4. 管理多个数据库连接

3.分类详解

1. select()   
   通过一个数组监视多个文件描述符，检查哪些描述符可读/可写   
   大小限制：数组大小受限于系统设置的FD\_SETSIZE。  
   效率低下：每次调用都需要扫描整个数组。
2. poll()  
   改进版的select()，使用链表监视文件描述符  
   解决了数组大小限制  
   同样需要**遍历整个链表**来判断事件状态
3. epoll()   
   Linux 提供  
   事件驱动：不需要扫描整个文件描述符列表，而是通过事件通知机制，在事件发生时自动触发回调。

* 边缘触发（ET）：仅在状态变化时触发事件 适用于高性能场景
* 水平触发（LT，默认）：只要状态未变化，每次查询都会触发事件

支持大规模文件描述符、适用于高并发的网络服务器场景

知识框架

线程和进程区别

线程（Thread）和进程（Process）是操作系统中进行多任务处理的两个基本概念，它们有一些关键的区别：

1. 定义：

线程： 线程是进程内的一个独立执行单元，共享进程的资源，包括内存空间和文件句柄等。多个线程可以在同一进程内并发执行。

进程： 进程是一个独立的执行环境，包括独立的内存空间、文件句柄等，可以包含多个线程。

2. 资源分配：

线程： 线程是进程内的轻量级执行单元，共享进程的资源。线程之间的切换开销较小。

进程： 进程是独立的执行环境，有独立的内存空间和资源。进程之间的切换开销较大。

3. 通信和同步：

线程： 线程之间共享进程的数据和资源，因此需要进行同步操作以避免数据竞争。

进程： 进程之间的通信较为复杂，通常需要使用进程间通信（IPC）机制，如管道、消息队列、共享内存等。

4. 创建和销毁开销：

线程： 创建和销毁线程的开销相对较小，因为线程共享进程的资源。

进程： 创建和销毁进程的开销相对较大，因为每个进程都有独立的资源空间。

5. 独立性：

线程： 线程是进程的一部分，对于同一进程内的线程，它们共享相同的地址空间和文件描述符等资源。

进程： 进程是相对独立的，它们有独立的地址空间和资源，通常需要通过IPC进行通信。

6. 安全性：

线程： 多个线程共享同一进程的地址空间，因此需要使用同步机制来保证数据的一致性。

进程： 进程拥有独立的地址空间，相对于线程更为安全，不容易出现数据竞争。

7. 适用场景：

线程： 适用于需要轻量级并发的场景，如GUI应用、网络服务器等。

进程： 适用于需要独立执行环境、数据隔离的场景，如操作系统、数据库系统等。

总体来说，线程适用于并发执行的场景，能够更高效地利用系统资源；而进程适用于需要隔离和独立执行环境的场景，但其开销相对较大。在实际应用中，线程和进程可以结合使用，根据具体需求选择合适的并发模型。

线程和协程的区别与联系

**区别：**

执行单元：

线程： 是操作系统调度的最小执行单元，由操作系统进行管理和调度。

协程： 是在用户空间进行调度的执行单元，由程序员显式控制。

调度方式：

线程： 由操作系统进行抢占式调度，可以被中断和切换。

协程： 由程序员控制，在需要时显式地挂起和恢复。

并发性：

线程： 线程可以并行执行，由操作系统管理多个线程的调度。

协程： 协程一般在单个线程内执行，通过非抢占式调度实现协作式并发。

状态保存：

线程： 线程的状态由操作系统负责保存和恢复。

协程： 协程的状态由程序员显式保存和恢复，通常保存在栈上。

**联系：**

并发编程：

无论是线程还是协程，都是为了实现并发编程，让程序能够有效地处理多个任务。

异步编程：

协程常用于异步编程，通过非阻塞的方式处理I/O等操作，提高程序的性能。

资源消耗：

协程通常消耗的资源较少，因为它在用户空间进行调度，减少了操作系统线程切换的开销。

可读性和控制：

协程通常具有更好的可读性，因为它在编写代码时更接近自然的同步写法，而不需要考虑锁和线程间通信的复杂性。

程序员对协程的控制更细粒度，可以手动控制协程的执行顺序。

**总体而言**

线程更适用于 CPU 密集型任务

协程更适用于 I/O 密集型任务，以及需要更好的可读性和控制的情况。