提问的问题：

1. 什么是生产者消费者模型？

是一种解决并发编程中资源共享与同步的模型，生产者线程负责生产任务并将其放入共享的任务队列中。消费者线程负责从任务队列中取出任务并进行处理。任务队列用于存储生产者生产的任务。如果任务队列已满，在等待期间，生产者线程会被阻塞，直到其他线程通过条件变量唤醒它。被唤醒的生产者线程将重新尝试互斥锁，然后继续向任务队列中添加任务。

1. 这个模型如何解决并发中的资源共享和同步问题（同步事件）？

生产者消费者共享一个任务队列，也就是缓冲区；

使用**互斥锁**来保护对缓冲区的访问，确保同一时刻只有一个线程可以对缓冲区进行操作;

使用**条件变量**使线程可以进入等待或者唤醒状态；

通过**信号量**的值可以表示资源是否被占用

1. 什么是粘包问题？如何解决粘包问题？

粘包问题指的是，因为TCP是流式协议，在数据传输过程中，多个数据包会被粘在一起，导致接收方无法正确解析单个数据包的情况，造成解析错误。

使用了一个自定义的传输协议--小火车协议，这个协议是先发送文件名，再发送文件的数据。其中火车的结构体包含火车头和车身，火车头代表车身存放了多少数据，车身代表数据

发送方将文件名的长度数据打包成小火车的格式发送，接收方接收到可以传输的文件名，然后发送方再将数据打包发送给接收方，接收方接收到数据后再解析出协议火车的车身，从而获取到发送方发送的数据。接收端先读取消息长度字段，然后再根据消息长度读取相应长度的数据，从而确保每个数据包都被正确地解析。

1. 什么是Linux下的socket编程？

程序首先调用socket()函数创建一个Socket，指定通信协议为TCP；

服务器端需要通过bind()将socket和本地地址绑定；

服务器端使用listen()监听连接请求；

(a)客户端发起请求，服务器端调用accept；

(b)服务器端使用epoll将socket加入到epoll实例中，有连接到达时调用epoll\_wait；

客户端调用connect与服务器建立连接；

发送和接收数据，客户端用户端使用recv，send

关闭socket，使用close()

1. 我的项目是长连接的，有什么好处？

用长连接可以减少频繁的连接建立和断开，节省客户端服务器端开销，提高响应速度和通信效率，适用于需要频繁通信或实时交互的场景。缺点是占用服务器资源，高并发下可能会导致网络拥塞

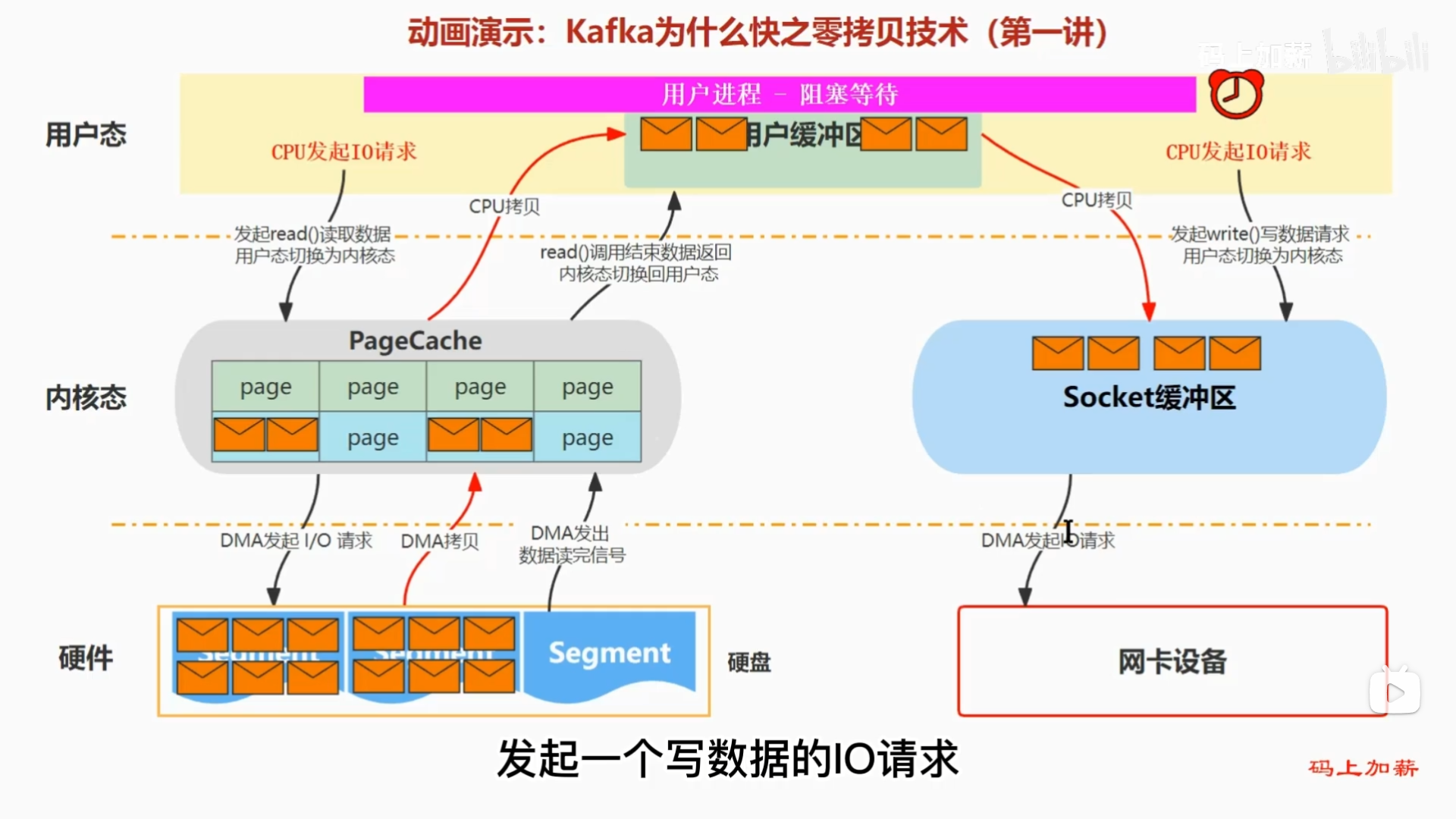
1. 数据库是怎么设计的？

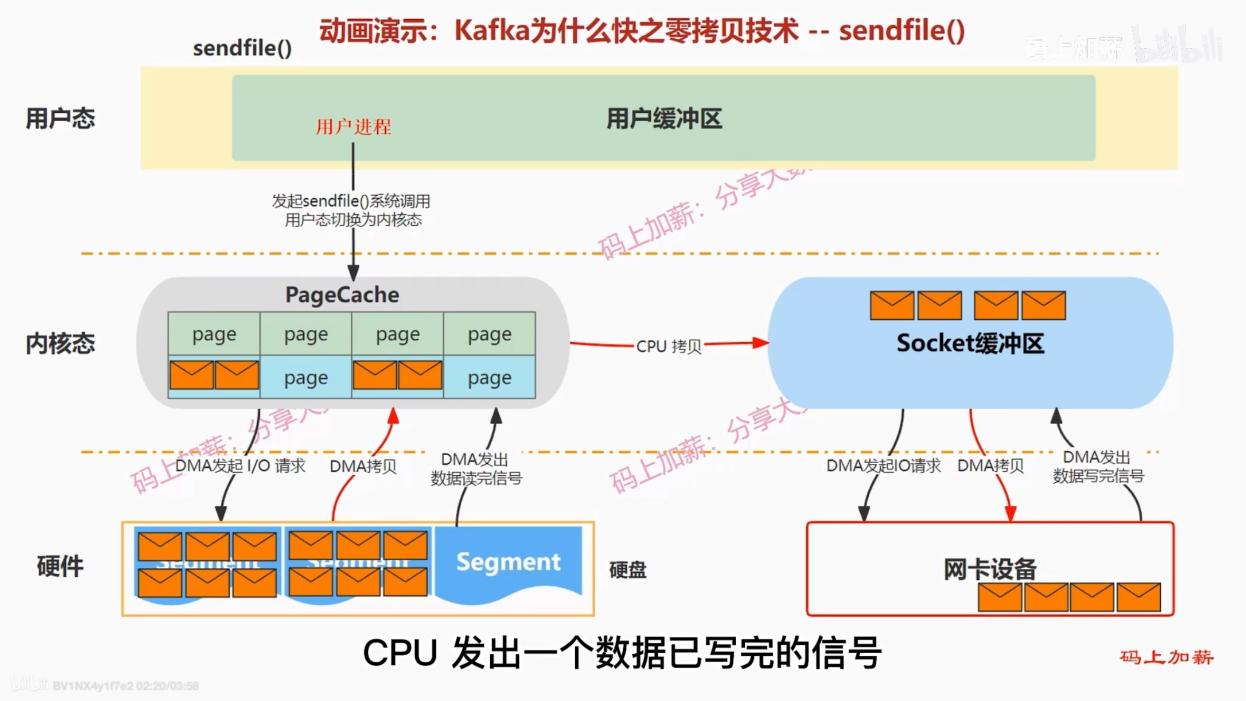
**MYSQL\_RES** 用于表示整个查询结果集，提供了获取和处理整个结果集的功能，而 **MYSQL\_ROW** 则用于表示单独的一行数据，提供了逐行获取和处理数据的功能。调用 **mysql\_fetch\_row** 函数，返回的 **MYSQL\_ROW** 类型的指针

1. 介绍一下零拷贝技术的sendfile？

sendfile可以高效地传输文件内容，能**直接将文件数据从文件描述符传输到另一个文件描述符。**

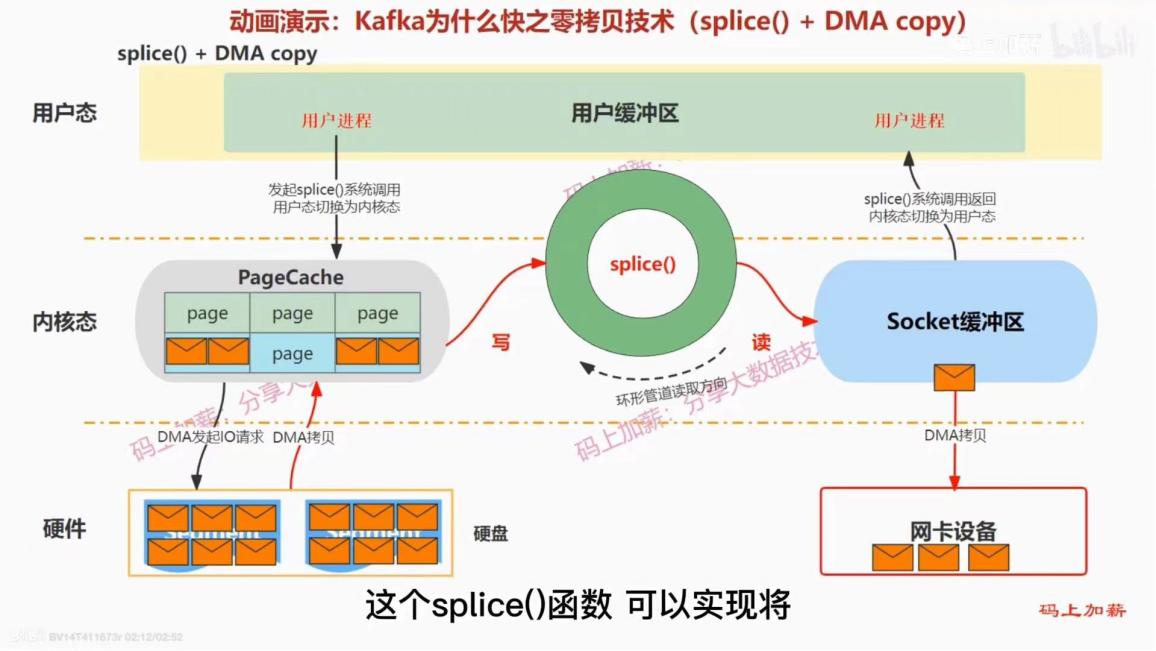
它会将数据**从磁盘**读取到**内核缓冲区**中，经CPU拷贝将数据**从缓冲区拷贝到socket缓冲区，再**传输到**目标位置**，它允许直接将文件数据从内核空间传输到网络套接字中，避免了用户态到内核态的数据拷贝，提高了传输效率。





1. 介绍一下splice？

splice是零拷贝技术的一种，splice函数同样是将数据从硬盘拷贝到内核缓冲区中，并将其写入到管道的写入端，并会从管道的读取端读取数据，将其写入到网络套接字或者目标文件描述符，通过从管道实现文件描述符之间的数据传输，而不经过用户空间。



1. 介绍一下什么是IO多路复用？

Process blocks in call to function(),waiting for one of possibly many sockets to become readable.

IO多路复用通过一个系统调用能同时监视多个文件描述符。一旦其中任何一个描述符有事件就绪时（如可读、可写或发生异常），IO多路复用通知事件就绪，应用程序就可以立即进行相应的操作，而无需创建额外的线程。这样可以有效减少线程的创建和销毁，减少系统调用开销，并提高系统性能和并发能力。

select：将要监听的描述符传给select函数，他会阻塞进程直到有事件就绪，它要不断遍历整个文件描述符集合检查是否就绪。使用了位图，文件描述符上限有限制。时间复杂度O(n)效率低。

poll：select的改进版本，用数组代替位图，解决了文件描述符的上限限制，但仍需要遍历整个集合。

epoll：采用了事件通知的方式，只需要在文件描述符就绪时就可以立即进行事件处理，而遍历文件描述符数组的操作会更快。因为epoll使用了红黑树存储文件描述符。

1. 请你介绍一下什么是水平触发，什么是边缘触发？

项目是水平触发epollin,而不是边缘触发epollet。因为长连接通常需要持续不断地监视文件描述符上的事件，以及持续处理这些事件。水平触发会在文件描述符上有就绪事件发生时立即通知应用程序，确保即时响应

水平触发是当epoll实例监听的文件描述符有事件就绪时，就调用epoll\_wait及时处理，只要该数据仍然可读写，该事件会一直保持触发状态，若事件没有即使处理，下一次调用时候仍会先处理上一次已经就绪的事件，适合持续监听描述符上的事件。

epoll\_wait调用多次，上下文切换开销大；若程序处理速度较慢，则会一直处于处于触发状态，增加系统负载。产生饥饿问题：某个线程由于资源被其他线程所占用而无法得到即时处理的情况。

边缘触发是当epoll实例监听的描述符上有数据时，会触发一次事件通知，但如果没有处理完，下一次将不再触发，除非客户端再发送新的数据。只有在状态发生改变的时候才会再通知一次。它更适合于一次性处理所有就绪事件，减少了通知次数。

边缘触发可能会导致明明有数据可读，却不会触发事件，导致程序陷入阻塞。因此它常配合非阻塞式IO的fd：make\_not\_blocking

1. 在Time\_wait的时候使用地址重用选项（SO\_REUSEADDR）允许套接字绑定到一个处于 TIME\_WAIT 状态的地址和端口上,，从而可以更快地重新启动服务器程序。
2. 仍有改进的方法，用struct sockaddr\_storage
3. 为什么不能绑定1024以下的端口？

因为这些端口是系统保留端口，需要超级用户权限才能使用

1. 客户端每次接收文件内容时接收1000的整数倍的字节是一种优化策略，减少系统调用次数，并提高文件写入效率。
2. 流程：客户端发送请求，服务器访问数据库资源，处理请求，服务器发送响应，客户端处理响应

项目流程：

服务端：

创建套接字（socket）： 使用 socket() 函数创建一个套接字，指定地址族（IPv4 或 IPv6）、套接字类型（通常是流式套接字 SOCK\_STREAM 或数据报套接字 SOCK\_DGRAM）和协议（通常是 0 表示默认协议）。

绑定地址（bind）： 使用 bind() 函数将套接字绑定到一个特定的 IP 地址和端口号上，以便客户端可以连接到该地址。可以使用 struct sockaddr 结构体来表示地址信息。

监听连接（listen）： 使用 listen() 函数将套接字标记为被动套接字，以便它可以接受连接请求。同时指定一个最大连接数（backlog），以限制同时等待连接的客户端数量。

接受连接（accept）： 使用 accept() 函数接受客户端的连接请求。这个函数会阻塞当前进程，直到有客户端发起连接请求。一旦有连接请求到达，它会返回一个新的套接字描述符，该套接字用于与客户端进行通信。

与客户端通信： 使用返回的新套接字描述符与客户端进行通信。可以使用 send() 和 recv() 函数发送和接收数据。

关闭套接字： 使用 close() 函数关闭套接字，释放资源。

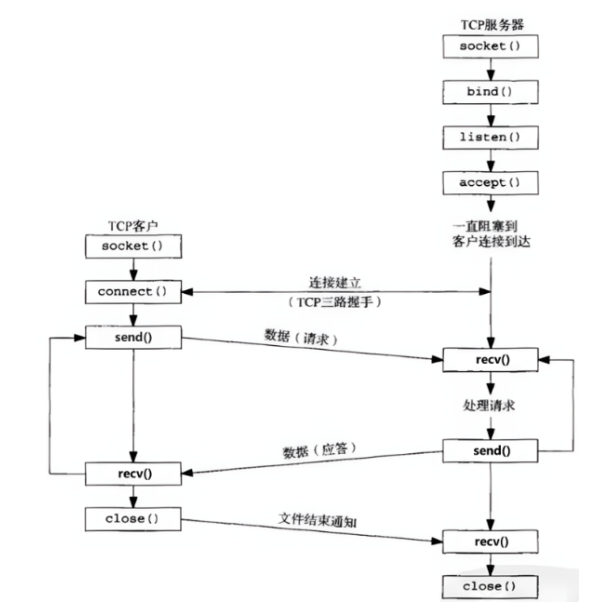
客户端：

创建套接字（socket）： 使用 socket() 函数创建一个套接字，同样指定地址族、套接字类型和协议。

连接服务端（connect）： 使用 connect() 函数连接到服务端的地址。可以使用 struct sockaddr 结构体表示服务端的地址信息。

与服务端通信： 连接建立之后，可以使用套接字描述符与服务端进行通信，发送和接收数据。

关闭套接字： 使用 close() 函数关闭套接字，释放资源。通常在完成通信后，或者出现错误时关闭套接字。



1. 项目是创建一个进程，并在这个进程内部创建线程池，无需使用 **fork()** 来创建子进程。
2. 虚拟指的是文件系统的逻辑结构是在数据库中进行模拟和管理的，而不是在实际的文件系统中。在这种虚拟文件系统中，逻辑结构和操作方式都是基于数据库的，文件实际上是存储在数据库中的。

软件工程：

内聚：用于描述模块内部之间相互联系紧密的程度。

耦合：用于描述不同模块之间相互依赖程度。

模块化：将软件系统划分为相互独立、可组合的模块的过程。

抽象：用于隐藏底层实现的细节，复杂性，并提供简单的接口。

数据结构：

二叉搜索树：是一种二叉树，其中每个结点的值大于其左子树所有节点的值，且小于其右子树所有结点的值，插入删除查找为ologn。（会由于插入删除不可控，导致退化成链表，于是有了平衡二叉树）

平衡二叉树：指的是树任意节点的左右子树高度差不超过1，能保持平衡的二叉搜索树。

左小于根小于右，递归定义

查找是从根节点开始逐层比较遍历左右的树

插入删除查找操作都ologn，空间复杂度on

红黑树:是二叉搜索树的一种

它的根叶黑，红节点子节点为黑色，

黑节点子节点可以是黑色或红色，但不能有两个相邻的红节点，

从任意节点到其子树每个叶子节点的路径包含了相同数量的黑色节点（黑高）

插入删除查找都是ologn

堆排序，时间复杂度onlogn空间o1，稳定性

构建大根堆，将待排序数组视作完全二叉树，从最后一个非叶子节点开始调整使其满足大根堆特征

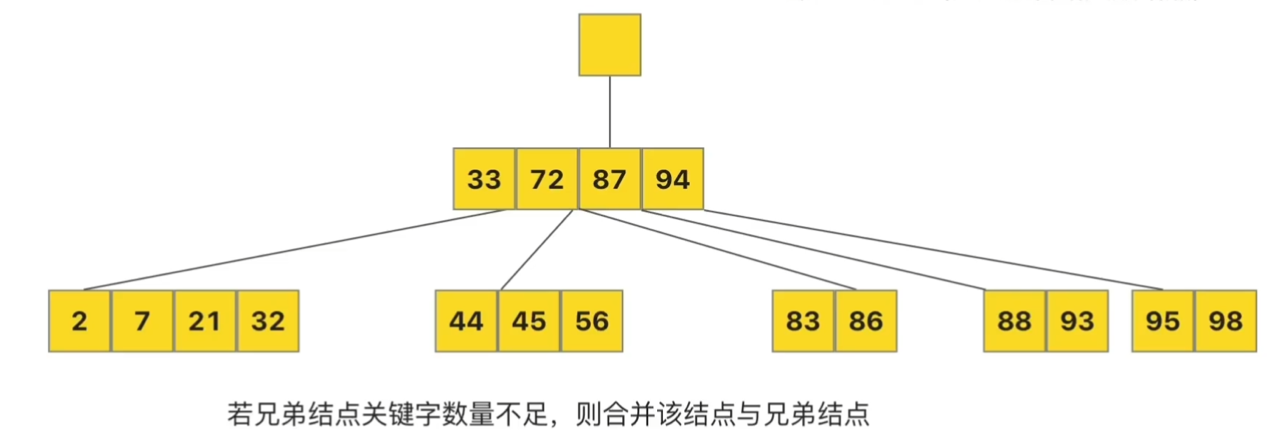
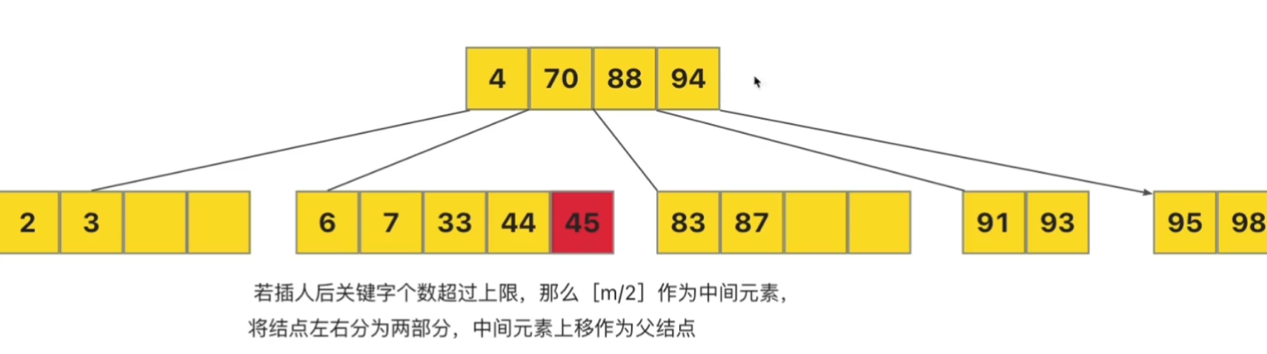
调节，将大根堆第一个元素与最后元素互换位置，对剩余不满足大根堆的元素再进行一次调整

排序，将上面的步骤重复执行，直到堆的大小为1

完全二叉树是特殊的二叉树，从上到下，从左到右构建的二叉树，除了最后一层以外，其他每一层的节点都是满的，而且最后一层的节点都在左侧

B树是一种多路搜索树，它的每个节点拥有多个子节点，并且每个节点的子节点数目相同且有序，保持了平衡，能支持范围查询

B树会通过分裂和合并动态调整树的平衡，其中分裂操作用于将节点中的关键字分成两部分并提升中间节点到父节点，合并操作用于将节点与其兄弟节点合并为一个节点，并将父节点中的一个关键字下降到合并后的节点中。



B+树：是一种自平衡多路搜索树，B+树的每个非叶子节点不存储数据，只存储索引信息，而数据全部存储在叶子节点中，同时B+树的叶子节点使用链表连接。

首先，B树的非叶子节点和叶子节点都可以存储数据，而B+树的非叶子节点只存储索引信息，所有的数据都存储在叶子节点中。

在B树中，每个节点都包含键值对，B+树中非叶子节点只包含键，负责索引，而叶子节点只包含值/数据，且叶子节点之间形成有序链表。

B树的查询涉及到递归，而B+树使用链表更加高效

B+树拥有更大的阶数和更小的层次，B树层次更多。

B树适合文件系统查询，B+树适合数据库索引

操作系统：

进程：是正在执行的程序过程，是操作系统资源分配的基本单位。每个进程拥有自己独立的地址空间。进程的状态：就绪态，运行态，阻塞态，终止态

线程：是操作系统执行和调度的最小单位，能缩小创建和切换进程的开销，一个进程可以包含多个线程，线程可以并发运行，会共享同一进程的地址空间。（栈独立，堆，fd共享）。

线程在并发执行的情况下，引入了互斥锁确保对共享资源的互斥访问。

**并发**：多个任务在同一时间段**交替**运行，同一时间只执行一个任务，不一定会同时执行。

**并行**：同一时间点上，多个任务**同时**执行。

串行：同一时间点上，只有一个任务被执行。

（不同在于任务执行的方式，时间关系）

互斥：保证在同一时刻只有一个进程能访问共享资源

互斥锁：控制共享资源。保证同一时刻只有一个线程访问共享资源，实现互斥。访问前加锁，访问后解锁。

有锁后尝试加锁，陷入阻塞状态，一个锁不能被两个线程同时拥有。

条件变量：用一个标志位表示线程是否可运行。这个状态是多线程共享的，所以修改的时候要加锁使用，条件变量必须配合互斥锁。

同步：指的是任务按顺序依次执行，一个任务的完成取决于前一个任务的状态。同步可能会导致死锁。

异步：指的是任务可以**独立**执行，不受其他任务影响。异步可以减少死锁的发生。

死锁：**多个**线程因争夺同一共享资源而陷入无限等待的状态。

饥饿：**某个**线程或进程资源分配策略不合理导致长时间无法获得所需的资源的现象。

陷入：程序执行特定的陷入指令，执行相对应的陷入处理程序

中断：来自CPU执行指令之外的事件发生，会执行与当前执行指令无关的事件。中断是由硬件或软件事件触发，适合执行异步事件。

异常：指CPU执行指令本身出现的问题，由程序内部状态所触发。

中断处理过程：关中断；保存断点；引出中断服务程序；（中断隐指令/硬件）

保存现场和屏蔽字；开中断；执行中断服务程序；关中断；恢复现场和屏蔽字；开中断，中断返回。（中断服务程序/软件）

系统调用：操作系统为用户态提供与硬件设备交互的接口，用软中断（执行软件指令）的方式使用户态陷入内核态。

银行家算法：避免死锁的方法，在进程请求资源前判断系统是否有足够资源分配给进程，若没有就不分配，保证系统处于安全状态，避免死锁的产生

计算机网络：

TCP/IP协议族：应用层，传输层，网络层，网络接口层

网络接口层：将比特流转换成数据帧

网络层：将数据帧封装成IP数据报

传输层：将IP数据报封装成TCP段

应用层：处理应用程序

Linux中，数据链路层，网络层，传输层已经有内核开发者写好，并且以内核线程形式运行，即内核协议栈。开发者只需关注应用层。私有协议：开发者自己设计并且不公开。

TCP协议：传输层，是可靠的，面向连接的，全双工的端到端协议。

3次握手：

客户端发起链接，发送SYN报文给服务器端，说明自己连接的端口和客户端序列号。

服务器端收到SYN报文，发起反方向链接，发送SYN给客户端说明信息。

客户端收到SYN报文，需要确认，发送ACK给服务器。

目的是建立连接，交换初始序列号。

4次挥手：

主动关闭方发送FIN端表示希望断开连接。

被动关闭方收到FIN，回复确认信息.

被动关闭方发送完数据了，发送FIN。

主动关闭方回复确认，并且等待一段时间。

Socket:套接字，用来建立连接，网络通信的设备

Connect:客户端使用connect建立和TCP服务端的链接（完成三次握手）

Bind:给套接字赋予一个本地协议地址（=IP地址加端口号）

Listen:使TCP服务端开启监听。服务端开启了listen就可以开始接受客户端的链接

Accept:服务端调用，从全连接队列中取出下一个已经完成的TCP连接

Send，Recv:将数据在用户态和内核态的缓冲区进行传输/将数据拷贝到内核态

TCP是流式协议：支持全双工通信，消息以字节流的方式传输，消息和消息之间是没有边界的，会产生粘包问题（多个消息在一次传输中发送接收），半包问题（一个消息要多个传输才被接收）

文件描述符：用于标识和访问文件、设备、套接字等 I/O 资源的抽象概念

套接字：用于网络通信的抽象概念，处于传输层和应用层之间

全双工：通信双方可以进行双向数据传输，双方可以同时接收和发送数据

半双工：通信双方同一时刻只能进行单向数据传输

三次挥手连接，四次握手断开

客户端time\_wait状态：确保连接的可靠关闭，而长连接通常会避免频繁地进入和退出 TIME\_WAIT 状态

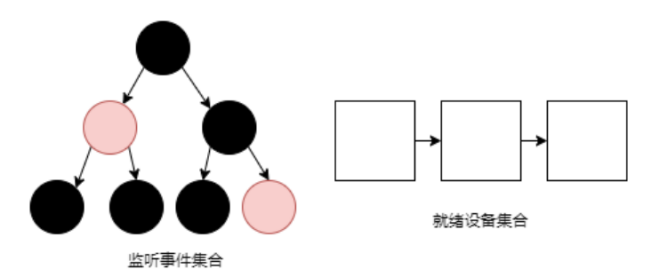
阻塞：程序或线程执行某个操作时被操作系统在内核态暂时挂起，直到某个条件满足或操作完成才会继续执行

**阻塞式IO：**在等待数据以及拷贝数据的过程，应用程序始终处于阻塞状态

**非阻塞式IO：**若数据没有就绪，则立刻返回（错误）

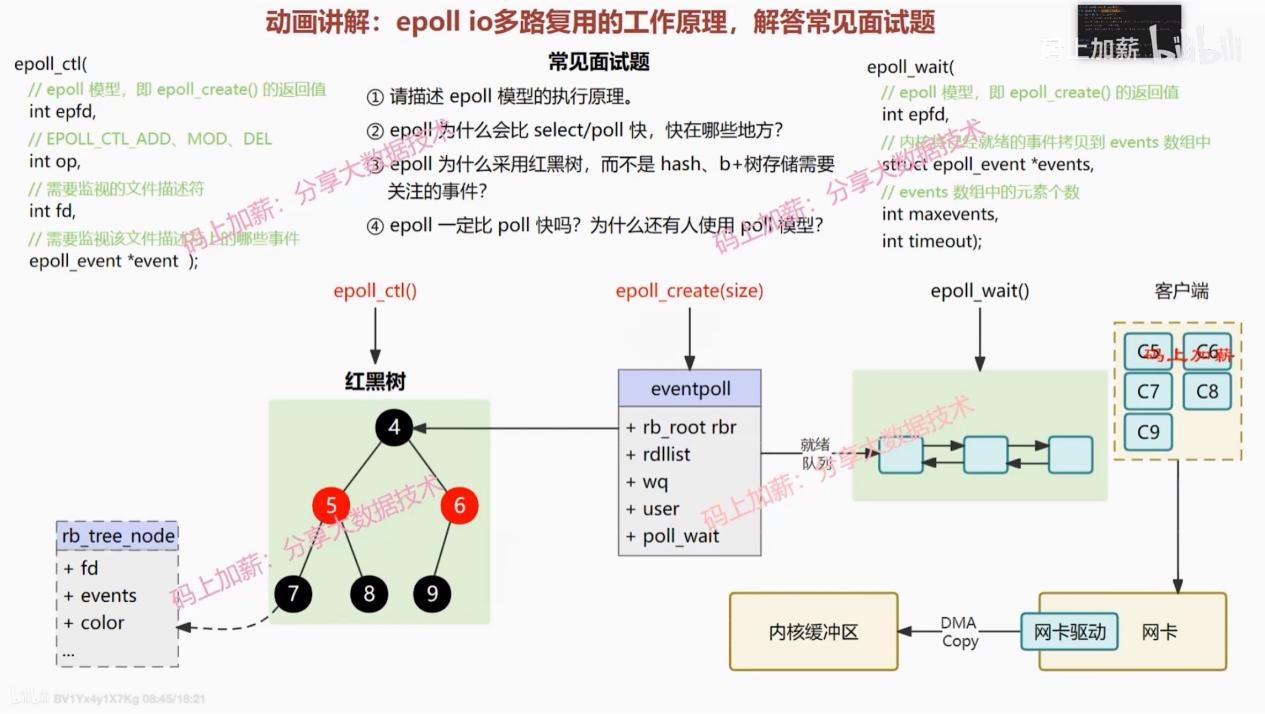
Epoll：在内核中维持两个数据结构：**监听事件集合&就绪事件队列。**

监听事件集合：存储所有要关注的文件描述符和操作。当文件描述符有事件产生时，操作系统会将就绪事件拷贝到就绪事件队列中，并且找到阻塞在epoll\_wait的线程，让其就绪。应用程序从就绪队列中获取事件并处理。监听事件集合是红黑树，就绪事件队列是线性表。



优势：监听事件集合在内核态，不需要每次将集合从用户态拷贝到内核态。

用两个数据结构存储，用户可以直接遍历就绪事件队列，而不需要在所有事件当中进行轮询。



epoll\_create 函数后，它将创建一个新的 epoll 实例，并返回一个文件描述符，我们可以通过这个文件描述符来操作该 epoll 实例。

epoll\_create函数创建一个epoll实例,"epoll实例"是指创建的epoll对象，它类似于一个事件监听器，用于管理和监控多个文件描述符的状态,在内核中创建一个新的epoll实例，并返回一个文件描述符，我们可以通过这个文件描述符来操作该epoll实例,使用这个文件描述符来向epoll实例注册需要监控的文件描述符，并设置关注的事件类型。之后，epoll实例会根据注册的事件，当有文件描述符的状态发生变化时及时响应.

Epoll\_ctl:

int ret = epoll\_ctl(epfd, EPOLL\_CTL\_DEL, fd, &event);

int epoll\_ctl(int epfd, int op, int fd, struct epoll\_event \*event);

它的返回值为 0 表示操作成功，返回值为 -1 表示操作失败

Epoll\_wait:

epoll\_wait 函数用于等待 epoll 实例中文件描述符上的事件触发，并获取触发事件的文件描述符及相关信息

int epoll\_wait(int epfd, struct epoll\_event \*events, int maxevents, int timeout);

maxevents：是 events 数组的大小，表示最多能存储多少个事件。timeout：是超时时间，用于指定 epoll\_wait 函数等待事件的最长时间，单位是毫秒。传递 -1 表示一直等待，直到有事件触发。传递 0 表示立即返回，传递正整数表示等待指定的毫秒数后返回，如果没有事件触发则返回 0。

当有事件触发时，会将触发事件的文件描述符及相关信息保存在 events 数组中。然后遍历 events 数组，处理触发的事件。返回值是触发事件的文件描述符的数量.

epoll 的就绪队列是由操作系统内核维护的，程序无法直接访问。程序通过调用 epoll\_wait 函数来获取就绪队列中的事件，并将就绪的事件信息填充到用户提供的 struct epoll\_event 数组中。这个数组中的每个元素表示一个就绪的事件。因此，程序可以通过遍历这个数组来处理就绪的事件。

