**17、堆、栈**

堆：操作系统有一个记录空闲内存地址的链表，当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请空间的堆结点，然后将该节点从空闲节点链表中删除，并将该结点的空间分配给程序，另外，对于大多数系统，会在这块内存空间中的首地址处记录本次分配的大小，这样代码中的delete语句才能够正确的释放内存空间。常说的内存泄漏，最常见的就是堆泄露（还有资源泄露），它是指程序在运行中出现泄露，如果程序被关闭掉的话，操作系统会帮助释放泄露的内存。

栈：在函数调用时第一个进栈的主函数中的下一条指令（函数调用语句的下一条可执行语句）的地址就是函数的各个参数，在大多数C编译器中，参数是由右往左入栈，然后是函数中的局部变量。

C++所编译的程序通常在内存中分为以下几个部分

1、栈区：由编译器自动分配释放，存放函数的参数值，局部变量的值等等。其操作方式类似于数据结构中的栈。

2、堆区：一般由程序员分配和释放，若程序员不释放，程序结束时可能有系统回收，但也可能不回收造成内存泄漏。他与数据结构中的堆结构并不相同，分配方式更类似于链表。

3、全局区（静态区）（static）：全局变量和静态变量的存储是在一起的，初始化全局变量和静态变量在一个区域，未初始化的全局变量和未初始化的静态变量在相邻的另外一片区域，程序结束后由系统释放。

4、文字常量区：常量字符串就是放在这里的。程序结束后由系统释放

5、程序代码区：存放函数体的二进制代码

二、例子程序

这是一个前辈写的，非常详细

//main.cpp

int a = 0; 全局初始化区

char \*p1; 全局未初始化区

main()

{

int b; 栈

char s[] = "abc"; 栈 //更正：abc 分配在静态存储区，不是栈上

char \*p2; 栈

char \*p3 = "123456"; 123456\0在常量区，p3在栈上。

static int c =0； 全局（静态）初始化区

p1 = (char \*)malloc(10);

p2 = (char \*)malloc(20);

分配得来得10和20字节的区域就在堆区。

strcpy(p1, "123456"); 123456\0放在常量区，编译器可能会将它与p3所指向的"123456"优化成一个地方。

}

申请方式：

栈：系统自动分配，如申请局部变量int b;系统自动会在栈中开辟一个空间给b

堆：程序员自己申请，并指明大小，比如c中的malloc函数。C++的new运算符

P1=(char\*)malloc(10);

但是指针p1本身是在栈中的

申请后系统的相应：

栈：只要栈的剩余空间大于所申请的空间，系统将为程序提供内存，否则将报异常退出。

堆：当系统收到程序的申请时，会遍历该链表，寻找第一个空间大于所申请结点的堆结点，然后将该节点从空闲节点链表中三处，将结点的空间分配给程序。对于大多数系统，会在这块内存空间中的首地址记录本次分配的代销，这样，代码中的delete语句才能正确的释放本内存空间。由于找到的堆结点的大小不一定正好等于申请的大小，系统会自动的将多余的那部分重新放入空闲链表中。

**18、简述对称密钥密码体系与公钥密码体系的区别**

1．对称密钥密码体系

对称密钥密码体系也叫密钥密码体系，它是指消息发送方和消息接收方必须使用相同的密钥，该密钥必须保密。发送方用该密钥对待发消息进行加密，然后将消息传输至接收方，接收方再用相同的密钥对收到的消息进行解密。这一过程可用数学形式来表示。消息发送方使用的加密函数encrypt有两个参数：密钥K和待加密消息M，加密后的消息为E，E可以表示为E=encrypt(K，M)消息接收方使用的解密函数decrypt把这一过程逆过来，就产生了原来的消息：

M=decrypt(K，E)=decrypt(K，encrypt(K，M))

2．非对称密钥密码体系

    非对称密钥密码体系又叫公钥密码体系，它使用两个密钥：一个公共密钥PK和一个私有密钥SK。这两个密钥在数学上是相关的，并且不能由公钥计算出对应的私钥，同样也不能由私钥计算出对应的公钥。这种用两把密钥加密和解密的方法表示成如下数学形式。假设M表示一条消息，pub—a表示用户a的公共密钥，prv—a表示用户a的私有密钥，那么：

 M=decrypt(pub—a，encrypt(prv—a，M))

**21、进程与线程间的通信**

Linux进程间通信：管道、信号、消息队列、共享内存、信号量、套接字(socket)

Linux线程间通信：互斥量（mutex），信号量，条件变量

Windows进程间通信：管道、消息队列、共享内存、信号量 （semaphore） 、套接字(socket)

Windows线程间通信：互斥量（mutex），信号量（semaphore）、临界区（critical section）、事件（event）

**22、死锁产生的原因**

产生死锁的原因主要是：

（1） 因为系统资源不足。

（2） 进程运行推进的顺序不合适。

（3） 资源分配不当等。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则

就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

产生死锁的四个必要条件：

（1） 互斥条件：一个资源每次只能被一个进程使用。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3） 不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。

（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

这四个条件是死锁的必要条件，只要系统发生死锁，这些条件必然成立，而只要上述条件之

一不满足，就不会发生死锁。

死锁的解除与预防：

理解了死锁的原因，尤其是产生死锁的四个必要条件，就可以最大可能地避免、预防和

解除死锁。所以，在系统设计、进程调度等方面注意如何不让这四个必要条件成立，如何确定资源的合理分配算法，避免进程永久占据系统资源。此外，也要防止进程在处于等待状态的情况下占用资源。因此，对资源的分配要给予合理的规划。

**23、分布式理论**

分布式中，一般要求的是CAP，分别为一致性，可用性，可分区性

因为要在分布式系统中，所以我们优先保障可分区性，即P，此时C和A只能保证一个：

如果保证了一致性，则我们的每次操作，都要求完全一致后才成功，这会严重影响可用性，

如果保证了可用性，则无法保证一致性。

好像分布式系统中最常用的是最终一致性。

**25、TCP协议三次握手的过程**

TCP协议三次握手的简析：

1、tcp标志位有6种标识：SYN（synchronous建立联机）,ACK（acknowledgement确认）,PSH（push传送）,FIN（finish结束）,RST（reset重置）,UGR（urgent紧急）。

Sequence number（顺序号码），Acknowledge number（确认号码）

第一次握手：主机A发送位码syn=1，随机产生seq number=1234567的数据包到服务器，主机B由syn=1知道，A要求建立联机。

第二次握手：主机B收到请求后要确认联机信息，向A发送ack number=（主机A的seq+1），syn=1，ack=1，随机产生seq=7654321的包

第三次握手：主机A收到后检查ack number是否正确，即第一次发送的seq number+1，一以及位码ack是否为1，若正确，主机A会再发送ack number=（主机B的seq+1），ack=1，主机B收到后确认seq值与ack=1则表示连接建立成功。

完成了三次握手之后，主机A与主机B开始传送数据

简述：TCP/IP协议中，TCP协议提供可靠的连接服务，采用三次握手建立一个连接。

第一次握手：建立连接的时候，客户端发送syn包（syn=j）到服务器，并进入SYN\_SNED状态，等待服务器的确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1)，同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN+ACK包，向服务器发送确认包ACK（ack=k+1），此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。完成三次握手后，客户端和服务器端开始传送数据。

**TCP/IP连接时候的状态**：客户机开始的时候处于CLOSED状态，客户机发起一个TCP之后，TCP发送一个SYN报文段，发送完之后进入SYS\_SENT状态。当客户机TCP处在SYN\_SENT状态时，服务器收到客户机的报文段之后进入SYN\_RECV状态，客户机等待来自服务器TCP客户机所发报文段进行确认的且SYN比特被置为1的一个报文段。收到这样的一个报文段之后，客户机TCP进入ESTABLISHED状态。

**TCP/IP的关闭**：这引起客户机TCP发送一个FIN比特被设置为1的TCP报文段，并进入FIN\_WAIT\_1状态，客户机TCP等待一个来自服务器的带有确认信息的TCP报文段。当他收到该报文段时，客户机TCP进入FIN\_WAIT\_2状态。当初在FIN\_WAIT\_2状态的时候，客户机等待来自服务器的FIN比特被置为1的另一个报文段，收到该报文段后，客户机TCP对服务器的报文段进行确认，并进入TIME\_WAIT状态。TIME\_WAIT状态使得TCP客户机重传最终确认报文，以防ACK丢失。在TIME\_WAIT状态中所小号的时间是与具体实现有关的，一般是30秒、1分钟、2分钟。经过等待后，链接就正式关闭，客户机端所有与连接有关的资源将被释放。

**26、进程和线程的区别**

一个程序至少有一个进程，一个进程至少有一个线程。线程的划分空间要小于进程。对于线程来说，同一个进程的多个进程共享一个执行单元（程序的执行效率会提高），但线程会有自己的堆栈和局部变量。同一程序的多个进程有自己的独立地址空间。进程是资源分配的基本单位，线程是资源调度的基本单位。这也就意味着对于操作系统来说，并不会讲多线程看做一个独立的应用，来进行调度和资源分配。

**27、HTTP状态码与HTTP请求**

HTTP请求主要在Web浏览器与Web服务器之间完成了下列7个步骤

1、建立TCP连接（一般端口号为80端口）

2、Web浏览器向Web服务器发送请求命令：建立了TCP连接，Web浏览器就会向Web服务器发送请求比如说“GET","POST"等等

3、Web浏览器发送请求头信息

4、Web服务器应答：客户机向服务器发出请求后，服务器会向客户机回送应答，比如成功的时候返回HTTP/1.1 200 OK

5、Web服务器发送应答头信息

6、Web服务器向浏览器发送数据

7、Web服务器关闭TCP连接

HTTP请求的格式，包含3部分

1、请求方法：比如GET,POST,HEAD,OPTIONS,PUT,DELETE等等

2、请求头

3、请求正文

HTTP应答码

1XX-信息类，表示收到了Web浏览器请求，正在进一步的处理中

2XX-成功类，表示用户请求被正确的接受（比如常见的200 OK）

3XX-重定向类，表示请求没有成功，客户必须采取进一步的动作

4XX客户端错误，表示客户端提交的请求有错误，比如常见的（404 NOT Found）

5XX服务器错误-表示服务器不能完成对请求的处理比如常见的500，501

**28、简述银行家算法**

银行家算法是一种避免死锁的一种重要方法。

现在我们来复习一下死锁发生的条件

1、互斥条件

2、请求和保持条件

3、不可抢占剥夺条件

4、环路等待条件

银行家算法是避免死锁的一种重要的方法（注意，是避免死锁的方法），程序实现的思路来源于银行的借贷业务，一定数量的本金要应多个客户的借贷周转，为了防止银行加资金无法周转无法而倒闭，对每一笔贷款，必须考虑其是否能限期归还。系统中有限的资源要供给多个进程使用，必须保证得到的资源的进程能在有限的时间内归还资源，避免死锁的发生。

把一个进程需要和已占有的资源的情况记录在进程控制中，假定进程控制块PCB其中的“状态”有就绪态、等待态、完成态。当进城处于等待态时，表示系统不能满足该进程当前的资源申请。“资源需求量”表示进程在整个执行过程中总共要申请的资源量。显然，每个进程的资源需求总量不能超过系统拥有的资源总数，银行家算法进行资源分配可以避免死锁。

**29、生产者消费者问题**

有的也称“有限缓冲问题”，是一个多线程同步的经典案例，该问题描述了共享固定大小缓冲区的连个线程——即所谓“生产者”和“消费者”——在实际运行时发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区，然后重复此过程。与此同时，消费者也在缓冲区中消耗这些数据。该问题的关键就是要保证生产者不会再缓冲区满的时候加入数据，消费者也不会再缓冲区空的时候消耗数据。

要解决该问题，就必须让生产者在缓冲区满时休眠，等待下次消费者消耗缓冲区中的数据的时候，生产者才会被唤醒，开始往缓冲区里添加数据。同样，也可以让消费者在缓冲区空时进入休眠，等待生产者往缓冲区添加数据之后，在唤醒消费者。通常采用进程间通信的方法解决该问题，常用的方法有“信号灯法“等。如果解决方法不够完善，则容易出现死锁的情况。出现死锁的时候，两个线程都会进入休眠，等待对方唤醒自己。该问题也能被推广到多个生产者和消费者的情形。

未完待续……

**30、中断和异常是什么？他们有什么区别？**

中断：CPU对系统发生的某个事件做出的一种反应，CPU暂停正在执行的程序，保留现场后自动的转去执行相应的处理程序，处理完该事件后在返回断点继续执行之前被打断的程序。常见的比如I/O中断，时钟中断，控制台中断等等。

异常：来自CPU的内部事件或程序执行中的事件引起的过程，比如由于CPU本身故障，程序故障，请求系统服务的引起的中断异常等等。

CPU在处理中断和异常的时候其实大概的处理本质是完全一致的，异常又称同步中断，中断又称异步中断。异常分为三大类

陷阱：有意的异常、返回行为，总是返回到下一条指令

故障：潜在的可恢复的错误，返回行为、返回到当前命令

终止：不可恢复型的错误，返回行为、不会返回

中断一般发生在程序执行的时候

异常一般发生在一条指令终止之后

**31、TCP如果两次握手会怎样？**

一个例子：已经失效的连接请求报文段因为某个网络节点滞留了，以致于延误到了某个连接已经释放的server。本来这是一个早已经失效的报文段，server收到此失效的连接请求报文段之后，会误认为是client再次发出了一个新的连接请求。于是就想client发送了确认报文段，同意连接。如果不采用三次握手，那么只要server确认，新的连接就建立了，然而client并没有发送这样的一个连接的请求。但如果使用了三次握手，client是不会理睬这个server的确认，也不会发送ack包给server。

**32、简述ARP协议和NAT协议**

ARP：地址解析协议，根据IP地址获取物理地址的一个TCP/IP协议。主机发送信息时将包含IP地址的ARP请求广播到网络上所有的主机，并接受返回消息，依次确定目标的物理地址；收到返回消息后将该IP地址和物理地址存入本机的ARP缓存中并保留一定时间，下次请求时直接查询ARP缓存以节约资源。

算法部分

36、TCP的四次分手中，主动关闭方为什么要等待2MSL时间

1、保证TCP协议的全双工连接能够可靠关闭

2、保证这次的连接重复数据段从网络中消失（防止新连接造成影响）（如果不等待2MSL，同时立即建立新连接，那么在关闭连接前发送的失效报文段可能影响本次连接）

**41、多进程写文件的问题（Linux）**

1、单独的将所有写文件的请求发送至一个Linux日志文件服务

启动一个Logger进程，其他进程向logger发消息，即将写文件的数据发送给logger，由logger来写文件（实现的话可能相对复杂一些）

2、采用文件锁

如果不想大费周章写Logger进程启动服务，可以使用文件锁

flock():文件级别的锁，针对整个文件进行加锁

fcntl():段级别的加锁，能够针对文件的某个部分进行加锁

使用O\_APPEND打开文件，可以将打开文件、写入文件末尾合并为一个原子操作，防止多个进程相互覆盖写入的情况

一个使用文件锁的例子

if( (fp = fopen(path, "a")) == NULL ) {

fprintf(stderr, "无法打开出错日志文件[%s] %s\n",

path, strerror(errno) );

fp=stderr;

}

fd = fileno(fp);

lockf(fd, F\_LOCK, 0l);

fprintf( fp, "this is process[%d]'s message \n", getpid() );

lockf(fd, F\_ULOCK, 0l);

}

**43、Linux的IO多路复用select和epoll之间的差别**

select、epoll都是一种IO多路复用机制（还有poll），可以监视多个描述符，但是本质上在进行读写操作的时候还是同步的。

select在调用的时候需要将fd\_set从用户空间拷贝到内核空间

然后遍历所有的fd，调用其对用的poll方法

poll方法返回一个文件描述是否就绪的mask掩码

如果遍历完一遍之后均没有一个可写的掩码，则进入睡眠。等待别人唤醒或者定时器到时自动唤醒

select的缺点很明显

1、每次调用select，都需要把fd集合从用户态拷贝到内核态，这个开销在fd很多的时候会很大。

2、每次调用select都需要在内核中遍历一遍fd，这个开销在fd多的时候也很大

3、select支持的文件描述符太小了，默认只有1024

epoll针对的就是select的这三个缺点进行的改进

对于1，在epoll\_ctl函数中，每次注册新的事件到epoll的handler中，会把所有的fd拷贝进内核，保证了每个fd在整个过程中只会拷贝一次

对于2，epoll并不会在每次执行时都轮训，还是只轮训一次，并在轮训的时候指定每个fd的回调函数，当唤醒队列上有等待者时，就会调用这个回调函数

对于3，epoll没有限制，所支持的上线最大即为可以打开的fd数目

**47、OSI模型**

应用层：提供应用程序间的通信

表示层：处理数据格式、数据加密

会话层：建立、维护和管理回话

传输层：建立主机端到端的连接

网络层：寻址和路由选择

数据链路层：提供介质访问、链路管理等

物理层：比特流传输

**一些算法**

二分搜索 Binary Search

分治 Divide Conquer

宽度优先搜索 Breadth First Search

深度优先搜索 Depth First Search

回溯法 Backtracking

双指针 Two Pointers

动态规划 Dynamic Programming

扫描线 Scan-line algorithm

快排 Quick Sort

**数据结构部分**

栈 Stack

队列 Queue

链表 Linked List

数组 Array

哈希表 Hash Table

二叉树 Binary Tree

堆 Heap

并查集 Union Find

字典树 Trie