# 计算机网络

## TIPS

电路交换 这跟网线 如果被两台机器用来通信了,其他信息就无法通过这跟网线来传递了,就得等这两台机器的通信断了才能用这跟网线.

分组交换 就是把数据切成一个个小的数据包,包里面标记上要发给谁,然后通过这个网线发送,这样的话很多对机器就可以同时使用这跟网线.

中继器 物理层的路由器 网桥 数据链路层的路由器

交换机 数据链路层的路由器

路由器 ip层的延长网络的设备

以太网首部->ip包首部->tcp包首部->数据

在一个局域网(以太网)内,用mac就可以互相发数据,但是一旦要跨局域网mac就不行了,就得借助于ip了!!!

ARP(网络层协议)通过ip找mac

Mac地址是下一跳的地址老换

Ip地址是指示着我们前进的最终地址

一个网卡只有一个mac,却可以有多个ip

所以说ping命令就是icmp协议,ip也是网络层的,但是这两个协议之前肯定会有一个arp的流程,获取一跳跳的mac地址.

IGMP协议

组管理协议 就是ip多播(组播)的组管理的,也是网络层的协议.

信令就用tcp,音视频啊数据包啊就用udp

Tcp的可靠性的保证 玩的就是序列号和ack机制

这样就会有超时重传机制.

三次握手

Client 我要连你了 协商点东西吧

Server 就这些信息 看看行不

Client 可以可以 明白了 通知你下

Client xxxx

Xxx

Xxx

Client发完第三次握手后就开始发数据了.

主机地址的那些位都是1的就是广播地址.

172开头的是B类地址,192开头的是C类地址

由于链路层有路径MTU的存在,所以ip层会存在分片.

v4 4个8比特 v6 8个16比特

滑动窗口(可以不收到ack还继续发送数据)

快速重传(收到了a4的ack,但是没有收到a3的ack,说明a3包丢了,我们马上重传a3的包)

由于滑动窗口 发送端要有个缓存区 缓存那些发了但是还没收到ack的消息

窗口大小动态变化

拥塞控制 分为慢启动(一开始发送包的速度是1248指数增长)和拥塞避免(到达一定数量后就由指数变成线性)

快速恢复就是一旦发现开始丢包了,把速率直接砍一半,然后不用慢启动了,直接用拥塞避免这种线性的.

http无状态无上下文,之所以做了这么简单是为了更大的并发量,如果想保存些上下文就需要用到cookie和session.

http长连接可以避免大量的tcp连接创建断开问题.

所谓通信中的隧道就是通过加密啊或者加头啊,封装一层外面看不懂的东西,对外面就是透明的就是看不懂的字节流,那么两个节点的通信就像是有一个隧道一样.

URI就是唯一资源标识符,URL是URI的一种实现方式.

## HTTPS

非对称加密 私钥加密 公钥可以解密 公钥加密 私钥也可以解密。

一般两端需要交互，都是先把自己的公钥给对方（公钥可以给很多人），私钥只有自己有，别人拿到了你的公钥，给你发消息时就会用公钥加密发给你，加密后的消息别人截获了也没有，别人没有你的私钥也解不了。

一般都是服务器把公钥发给用户，私钥服务端自己保存，用户收到服务器的公钥后把自己的公钥用服务器的公钥加密，发给服务器，服务器用自己的私钥解开后就拿到了用户的公钥，这样两端都保存了对方的公钥，发的消息都用对方的公钥加密了，通信就是安全的了。

问题就是第一步服务器把自己的公钥发给用户的时候还是不安全的。

解决方案就是服务端把自己的公钥告诉一个第三方机构 CA机构，CA把自己的信息啥的和服务器的公钥用自己的私钥加密成一个证书发给服务器，服务器把这个证书发给客户端，因为浏览器维护了大部分CA机构的名称地址和公钥信息，就可以验证这个证书是不是合法的对的，如果是真的，就用对应的CA机构的公钥解密这个证书拿到服务器的公钥就可以继续后面的流程了。

## 三次握手与四次挥手

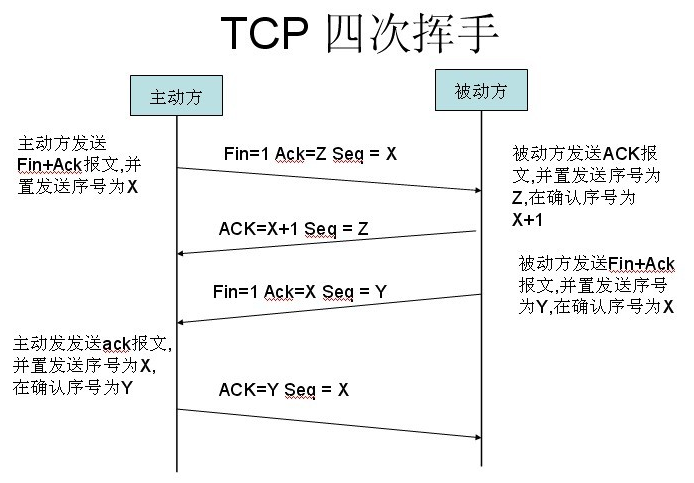


SYN叫做同步位,ACK叫做响应位,这俩大写的,每次都是1,是不会变的.

seq小写的这个才是序列号,注意client和server都会维护自己的seq序列号的,client和server也都是会给对面回复ack(小写)的时候ack=对面seq+1的.

为啥要三次握手呢,tcp连接是全双工的,也就是A给B发消息的同时B也能给A发消息.

那么A给B发完消息后,需要收到B的ack才能确定A->B的通路打通,如果只有两次握手的话client->server的通路可以确定通了,但是server->client的还没有通,因为server给client发的消息(第二次握手)没有收到ack,所以要有第三次握手.



四次挥手两边谁都可以触发,顺序并不一定是上面那样,也就是谁确定自己不给对面发消息了,就给对面发个FIN,但是对面可能还给你发,所以在四次挥手的时候会有个半关闭的概念.

四次挥手就是你给我发FIN,我给你ack,我给你发FIN,你给我回ACK.

这是因为服务端的LISTEN状态下的SOCKET当收到SYN报文的连接请求后，它可以把ACK和SYN(ACK起应答作用，而SYN起同步作用)放在一个报文里来发送。但关闭连接时，当收到对方的FIN报文通知时，它仅仅表示对方没有数据发送给你了；但未必你所有的数据都全部发送给对方了，所以你可能未必会马上会关闭SOCKET,也即你可能还需要发送一些数据给对方之后，再发送FIN报文给对方来表示你同意现在可以关闭连接了，所以它这里的ACK报文和FIN报文多数情况下都是分开发送的。

# 操作系统

进程是无法直接操作IO设备的,其必须通过系统调用来内核协助完成IP动作,而内核会为每个IO设备维护一个buffer.

整个请求为:用户进程发起请求,内核接受到请求后,从IO设备中获取数据到buffer中,再将buffer中的数据copy到用户进程的地址空间.

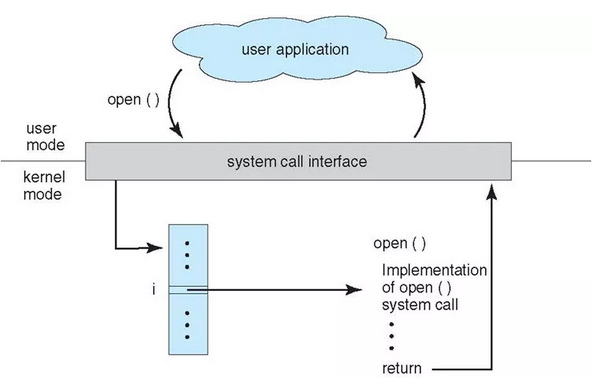
这里是用户进程请求内核把buffer的数据拷贝过来,这里就用到了系统调用read()函数了哦.

如果用户进程在read的时候buffer里面还没有数据,下面就是阻塞非阻塞啊,同步异步那套了.

## 系统调用

这里先说下系统调用,避免下面的不好理解.

如果一个进程在用户态需要使用内核态的功能,就进行系统调用从而陷入内核态,由操作系统代为完成.



Linux的系统调用主要有以下这些

| **Task** | **Commands** |
| --- | --- |
| 进程控制 | fork(); exit(); wait(); |
| 进程通信 | pipe(); shmget(); mmap(); |
| 文件操作 | open(); read(); write(); |
| 设备操作 | ioctl(); read(); write(); |
| 信息维护 | getpid(); alarm(); sleep(); |
| 安全 | chmod(); umask(); chown(); |

**看到了么,fork啊 sleep啊 read()啊这些操作系统的api其实都是系统调用.**

## Fork与phread\_create

首先以前我们说过linux是没有进程和线程的区别的,对于内核来说确实是这样的,线程和进程在内核里都是task.

但是呢,对于我们应用层来说,就是有进程和线程的区别的,不然fork(创建子进程)和pthread\_create(创建线程)这俩函数就没有意义了.

对于应用层来说,linux的进程线程机制就是一个挺标准的一个进程对应多个线程的机制.

下面完以下fork和phread\_create.

任务是,进程A创建一个进程B,进程B里面有两个线程,一个不停的做计算,另一个等待控制台的输入啥的.

代码块一:

进程的创建,

int main(){

pid\_t pid;

pid=fork(); 这里调用了fork函数,创建子进程.

if(pid<0){

printf("Fork failed");

exit(-1);

}

else if(pid==0){

printf("Child process start,child pid is %d.\n",getpid());

//exec..

char \*buf[]={"/home/wzp/charpter2/procedure2","procedure2",NULL};

execve("/home/wzp/charpter2/procedure2",buf,NULL);

这里pid=0说明这里是子进程,我们以前说过子进程会execve()一个文件,这个文件就是子进程执行的内容,这个问题可以是个脚本,也可以是一个ELF文件,也就是一个linux中的可执行文件(注意linux中的可执行文件的类型都是ELF文件)

所以说这个execve函数的参数,就是一片代码编译的可执行文件

}

else{

printf("Hello world,father pid is %d.\n",getpid());

wait(NULL);

exit(0);

}

}

上面是代码块一,开了进程,那么我们看下编译出子线程执行文件的代码是什么样子的

void \*Input(void\* arg);

void \*Sum(void\* arg);

union member{

int a;

char b;

}x;

int sum=0;

void \*Input(void\* arg){

printf("Please input an integer or a character:\n");

//printf("\*\*pthread 1 starts.\n");while(1)

{

if(scanf("%d",&x.a))

{

m=x.a;

flag=1;

conditions=1;

//printf("\*\*pthread 1 signal conditions to 2.\n");

}

else

{

scanf("%c",&x.b);

n=x.b;

flag=0;

conditions=1;

//printf("\*\*pthread 1 signal conditions to 2.\n");

}

//printf("\*\*pthread 1 releases lock.\n");

sleep(2);

}

}

void \*Sum(void\* arg){while(1)

{

//printf("\*\*pthread 2 starts.\n");

int i;

sleep(1);

while(!conditions) sleep(1);

sleep(2);

//printf("\*\*pthread 2 gets conditions.\n");

//printf("\*\*flag in pthread2 is %d\n",flag);

if(flag)

{

for(i=0,sum=0;i<=m;i++)

{

sum=sum+i;

}

printf("the sum from 0 to %d is %d\n",m,sum);

conditions=0;

sleep(1);

printf("Please input an integer or a character:\n");

}

else

{

//printf("\*\*judging the type of character.\n");

switch(n)

{

case 'p':

{

printf("Please input an integer or a character:\n");

while((!flag)&&(n=='p'))

{

sleep(1);

}

//conditions=1;

break;

};//stop Sum, use loop

case 'e':

{

printf("exit.\n");

exit(0);

break;

};//exit sum.exe

default:

{

printf("error.\n");

printf("Please input an integer or a character:\n");

setbuf(stdin, NULL);

break;

};//child process continue.

}

//printf("\*\*pthread 2 ends.\n");

}

}

}

int main(){

printf("exec success.\n");

pthread\_t id1,id2;

**pthread\_create(&id1,NULL,Input,(void\*)NULL);**

**pthread\_create(&id2,NULL,Sum,(void\*)NULL);**

**看到么,由于上面指定了执行这个文件,那么就从main开始,main函数就是调用了Pthread\_create函数创建两个线程,其中一个参数就是一个函数指针,这个指明了这个线程跑起来要做什么!!!!!!!!!!!!!!!**

pthread\_join(id1,NULL);

pthread\_join(id2,NULL);

Pthead\_join的意思就是这个线程等那个线程执行完再执行

//pthread\_mutex\_destroy(&mutex);

//pthread\_cond\_destroy(&cond);

exit(0);

return 0;

}

所以这些明白了fork和pthread\_create的使用方式和区别了.

## 基本特征

并发 共享 虚拟 异步

## 作用

内存管理 文件管理 进程管理 设备管理

## 内核分类

大内核,大内核是将操作系统功能作为一个紧密结合的整体放到内核,由于各个模块共享信息,因此有很高的性能.

微内核,由于操作系统不断复杂,因此降一部分操作系统功能移出内核,从而降低内核的复杂性.移出的部分划分成若干的服务,相互独立.

微内核一个模块运行在内核态,其他的运行在用户太,这样需要内核态用户太经常切换,性能会低一些.

## 中断的分类

### 外中断

由cpu执行指令以外的事件引起,如I/O中断(包括键盘,鼠标啊,硬盘这些引起的中断),还有其他的时钟中断,控制台中断等.

### 异常

由CPU执行指令的内部事件引起,如非法操作码,地址越界,除0等.

### 陷入

用户程序中使用了系统调用.

## 进程与线程

进程与线程的区别,特定,状态.

调度算法.

临界区 同步与互斥 信号量

死锁 哲学家吃饭问题 银行家算法

似乎线程有很多优势，比如，数据共享效率高，可应对并发操作，有效利用等待时间等等，但是多线程的编程比多进程要复杂，同时，多进程的可靠性较好，因为进程间不会相互影响。实际情况还是需要自己分析拿捏的。但是一般来说，实际应用中常常采用“进程+线程”结合的方式，而不是非此即彼

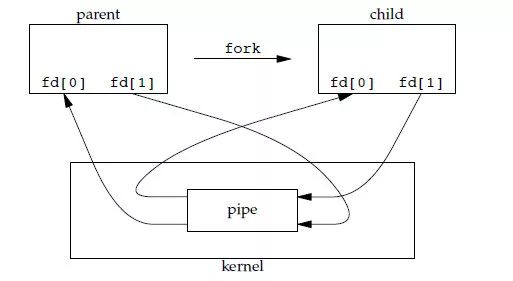
## 进程间通信 套接字是可以在不同机器进程中通信的方式!!!

### 管道pipe

这个有专门的api

#include <unistd.h>  
int pipe(int fd[2]);

只能在父子进程中使用,



### 消息队列

非常吊,好用.

### 信号量

这玩意只是一个计数器,用于为多个进程提供对共享数据对象的访问,

好像作用很小啊,就是个计数器啊,根本不如消息队列好用.

### 共享存储

允许多个进程共享一个给定的存储区.因为数据不需要再进程之间复制,所以这是最快的一个进程间通信.

以前我们知道进程是独立的拥有资源的单位,这个通信方式是让多个进程共享一块资源,所以这是一个特殊的方式,这个会借助信号量的帮助.

多个进程可以将同一个文件映射到他们的地址空间从而实现共享内存.

### 套接字

套接字可以用于不同机器之间的进程间通信.

## 内存管理

### 虚拟内存

虚拟内存的目的是为了让物理内存扩充成更大的逻辑内存,就是这个意思,操作系统把物理内存映射成逻辑内存,通过页关联起来,其实逻辑的内存比真正的物理内存要大,比如你的物理内存只有2g,但是逻辑内存可以映射出8g.

你的程序操作内存操作的是逻辑内存,比如你的这个程序要占用3g内存,那么你用的是逻辑内存的8g,但是呢,这样的话8g肯定无法映射到2g上啊,多啊,其实是这样的,一个程序在运行的时候并不需要把自己占用的逻辑内存全都加载到物理内存中,用到哪些逻辑页,把这些逻辑页加载到物理内存上就可以了.所以你的2g物理内存是可以搞出8g逻辑内存供应用程序使用的.

### 用户空间和内核空间

如果是4g的虚拟内存,对于linux操作系统来说,会有1g当作内核空间,保存内核的代码和数据,然后有3g分给应用程序使用,保存代码和数据.

外面的进程也可以通过系统调用进入内核,内核空间是所有进入内核的进程共享的,所以这样来看,应用进程可以使用的空间还是4g.

## 零拷贝

零拷贝描述的是CPU不执行拷贝数据从一个存储区域到另一个存储区域的任务，这通常用于通过网络传输一个文件时以减少CPU周期和内存带宽。

减少甚至完全避免不必要的CPU拷贝，从而让CPU解脱出来去执行其他的任务.

减少内存带宽的占用.

通常零拷贝技术还能够减少用户空间和操作系统内核空间之间的上下文切换.

上面我们也提到了,应用程序是不能操作内核中的给IO设备准备的buffer的,需要cpu把IO设备buffer中的内容复制到应用进程的地址空间里,零拷贝就是减少这步操作.

首先零拷贝这种东西一看就是要内核实现的东西,和java没有任何关系的.

通过sendfile实现的零拷贝I/O

带有DMA收集拷贝功能的sendfile实现的I/O

通过mmap实现的零拷贝I/O

上面这些都是linux实现零拷贝的机制.

select和epoll 函数监视的文件描述符分3类，分别是writefds、readfds、和exceptfds。