1.

Q:网络中进程是怎么通信的？（补充：进程是系统分配资源的基本单位，线程是系统调度执行的基本单位。进程与进程之间是独立的；一个进程可以包含多个线程，同一进程的多个线程共享这个进程的资源！）

A:因为在本地的话，可以用一个pid表示一个进程，但是在网络里这是行不通的。那么，网络里咋通信的？首先，在网络层，每台主机又有一个IP地址，这相当于是每台主机的身份证号。然后在 传输层，用 协议+端口号   ，标识一个主机的一个进程，用这种方式就可以确定一个进程（即  IP地址 + 协议 + 端口  ，用这样的三元组确定一个进程，然后就可以让进程之间通信啦）

2.

Q：什么是socket？

A：socket相当于是  “open –  read/write –  close 这样模式的一种实现”。socket一般是用在网络通信中，既然是通信，那么肯定得进行数据交换(我意思是指 数据的读、写)。用一个socket套接字来标识这个数据帧。然后对这个指定的数据帧进行读、写、关闭 操作。（可能不太准确啊，但是我觉得这样理解会简单一些）

3.

Q：socket怎么用？

A:

以TCP为例，下面是一套流程（关于怎么用socket？创建socket描述符，bind绑定给他分配一个地址，之后listen监听，accept阻塞，等待连接，之后就是read、write一套读写数据的，最后close就ok啦~）

socket()函数原型：int socket(int domain, int type, int protocol);

socket()函数对应普通文件的打开操作。普通文件的打开操作返回一个文件描述字，socket()返回一个文件描述符，指定不同的参数返回不同的文件描述符，后续的操作都需要用到这个描述符。socket()函数有3个参数，domain、type、protocol。

domain：协议族

type：用于指定socket类型

protocol：指定协议

当调用socket函数创建一个socket描述符的时候，返回的socket描述符存在于协议族空间里，他没有具体的地址。想给他赋一个地址的话必须用bind()绑定，否则，调用connect()、listen()时系统会自动给他分配一个端口。

bind()函数

原型：int bind(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);

socketfd：前面socket()函数生成的socket描述符

addr：一个const struct sockaddr \*指针，指向要绑定给sockfd的协议地址。这个地址结构根据地址创建socket时的地址协议族的不同而不同。什么ipv4啊、ipv6啊，这个用到的时候详细看看吧。

addrlen：它对应的是地址的长度

注意啊：通常 服务器创完socket之后要绑定一个众所周知的地址给大家访问，客户端的话，就不绑定了，反正我是去访问服务器的，又没人访问我，那就直接坐等系统分配~（在connect()的时候系统会自动分配，上面提过了）

listen()、connect()函数

如果作为一个服务器，在调用socket()、bind()之后，就会调用listen()来监听这个socket，如果客户端这时调用connect()发出连接请求，服务器端就会接收到这个请求。

原型：int listen(int sockfd, int backlog);  
           int connect(int sockfd, const struct sockaddr \*addr, socklen\_t addrlen);  
listen()函数的第1个参数即为要监听的socket描述字，第2个参数为相应socket可以排队的最大连接个数。socket()函数创建的socket默认是一个主动类型的，listen函数将socket变为被动类型的，等待客户的连接请求。

connect()函数的第1个参数即为客户端的socket描述字，第2参数为服务器的socket地址，第3个参数为socket地址的长度。客户端通过调用connect函数来建立与TCP服务器的连接。

accept()函数

原型：int accept(int sockfd, struct sockaddr \*addr, socklen\_t \*addrlen);

TCP服务器端依次调用socket()、bind()、listen()之后，就会监听指定的socket地址了。TCP客户端依次调用socket()、connect()之后就想TCP服务器发送了一个连接请求。TCP服务器监听到这个请求之后，就会调用accept()函数取接收请求，这样连接就建立好了。之后就可以开始网络I/O操作了，即类同于普通文件的读写I/O操作。

accept函数的第一个参数为服务器的socket描述字，第二个参数为指向struct sockaddr \*的指针，用于返回客户端的协议地址，第三个参数为协议地址的长度。如果accpet成功，那么其返回值是由内核自动生成的一个全新的描述字，代表与返回客户的TCP连接。

注意啊：accept的第一个参数为服务器的socket描述字，是服务器开始调用socket()函数生成的，称为监听socket描述字；而accept函数返回的是已连接的socket描述字。一个服务器通常通常仅仅只创建一个监听socket描述字，它在该服务器的生命周期内一直存在。内核为每个由服务器进程接受的客户连接创建了一个已连接socket描述字，当服务器完成了对某个客户的服务，相应的已连接socket描述字就被关闭了。

read、write等函数

网络I/O操作有下面这些组：

read()/write()  
recv()/send()  
readv()/writev()  
recvmsg()/sendmsg()  
recvfrom()/sendto()

close()函数

原型：int close(int fd);

服务器和客户端建立连接之后肯定会进行读写操作吧（没有算我没说。。）读写完了就要关闭描述字了。跟Linux里面的文件读写操作一样的，用完就管好习惯！关完之后嘞，返回到调用进程，然后这个已经关闭的socket描述符就不能再被用了，啥意思嘞，就是不能访问了、不能作为read、open这些函数的参数了

注意啊：close关完之后只是socket描述符的引用计数减 1 。当引用计数为0的时候，会触发客户端向服务器发送 终止连接请求。ok，gameover。

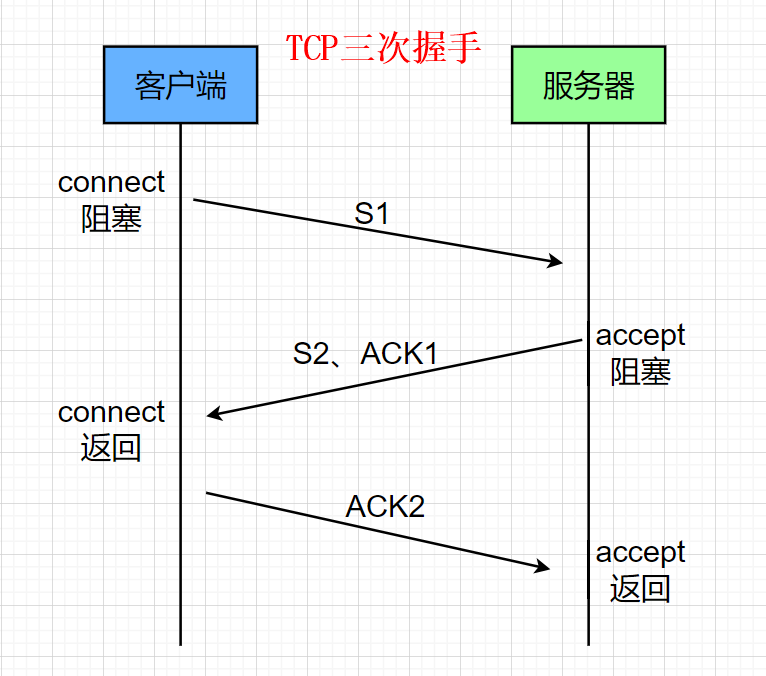
都打字打到这了，说一下TCP的三次握手吧

第一次：客户端调connect()函数，发送S1告诉服务器我要连接，然后connect()函数阻塞，等服务器回消息

第二次：服务器一直在监听呢，突然收到S1了，兴奋啊，赶紧发送一个S2告诉客户端我收到了，并且附带一个ACK1，让客户端确认一下。然后accept函数阻塞，等客户端回消息

第三次：客户端收到了服务器发来的俩信号，结束connect()阻塞状态，发送一个 ACK2 告诉服务器，我确认完毕了，连！然后服务器收到ACK2之后，accept函数结束阻塞。连接建立！！！

上图！！



4.tar命令

tar -czf wonderful.tar.gz pm         这是创建一个以.tar.gz结尾的包文件，调用gzip程序 将当前目录下的pm文件压缩到这个以.tar.gz结尾的文件里

tar -xf wonderful.tar.gz         将.tar.gz压缩包里的 文件/目录 解压到当前目录下

tar -tf /lianxi/wonderful/wonderful.tar.gz           查看压缩包里的文件和文件夹

解压：

tar -xvf    解压 \*.tar文件

tar -xzf     解压\*.tar.gz   或者   \*.tgz

unzip  \*.zip   解压zip文件到当前目录

5.tcp模型

先了解一下TCP和UDP。

udp协议定义了端口，同一主机上的每个应用程序都需要指定唯一的端口号，然后让网络中的数据包也要包含端口信息，这样数据到达主机之后，根据端口找应用程序。

tcp协议就是高级一点的udp。因为udp协议发送数据没有确认机制，tcp在udp基础上增加了确认机制，A收到信息要给B回一个确认，否则A就认为发送的数据丢失了，会再次发送。 为了保证数据传输的可靠性，TCP协议在正式发送数据之前有三次握手，这样一确认，就建立了一个可靠连接，然后进行传输数据。

tcp四层模型：应用层、传输层、网络层、网络接口层。

osi七层模型：应用、表示、会话、传输、网络、数据链路、物理层。

传输层的主要工作是 定义端口(源端口、目的端口)，表示应用程序身份，实现端口到端口的通信。

6.gdb调试(C/C++代码)

(调试的应该是生成的可执行文件，bitbake编译完成后  /DATA/Joynext/niu\_s2/new\_bmw\_ut/src/poky/build/tmp-glibc/work/x86\_64-linux/jn-power-service-unittests/0.1-r0/build/result/exe  文件夹下绿色的文件 )

主要功能是四个方面：

1）设置断点:断点就是可以暂停程序运行的标记。程序会在断点处停止，以便于用户查看程序的运行情况。这里的断点可以是行数、程序名称或条件表达式。        例如，在gdb模式下输入：b  src/CPowerClient.cpp:68     这是在CPowerClient.cpp第68行设置断点

2）查看信息:在断点停止后，可以查看程序的运行信息和显示程序变量的值。

3）分步运行:可以让程序一个语句一个语句地执行，这时可以及时查看程序的信息。       s

4）改变环境:可以在程序运行时改变程序的运行环境和程序变量

run（r） 是运行程序，break（b）是设置断点，ignore是忽略某个断点：ignore  3  1     忽略第3个断点1次   ，continue（c）  继续运行 ，quit（q）：退出gdb   ， info  (i)：显示信息   ，例：info threads   显示线程信息， thread   n  :切换到n线程(n是线程id号)

7.C++ 模板

模板写法：（class关键字可以用typename替换，如： template <typename 类型参数1, typename 类型参数2, ...>）

template <class 类型参数1, class类型参数2, ...>  
返回值类型 模板名(形参表)  
{  
函数体  
}

[https://blog.csdn.net/low5252/article/details/94622335?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168924450116800225567879%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=168924450116800225567879&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_positive~default-1-94622335-null-null.142^v88^control\_2,239^v2^insert\_chatgpt&utm\_term=%E6%A8%A1%E6%9D%BF%E5%87%BD%E6%95%B0&spm=1018.2226.3001.4187](https://blog.csdn.net/low5252/article/details/94622335?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522168924450116800225567879%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=168924450116800225567879&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-94622335-null-null.142%5ev88%5econtrol_2,239%5ev2%5einsert_chatgpt&utm_term=%E6%A8%A1%E6%9D%BF%E5%87%BD%E6%95%B0&spm=1018.2226.3001.4187)

[https://blog.csdn.net/JAN6055/article/details/123801017?ops\_request\_misc=%257B%2522request%255Fid%2522%253A%2522168924438316800188572717%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request\_id=168924438316800188572717&biz\_id=0&utm\_medium=distribute.pc\_search\_result.none-task-blog-2~all~top\_click~default-2-123801017-null-null.142^v88^control\_2,239^v2^insert\_chatgpt&utm\_term=c%2B%2B%E6%A8%A1%E6%9D%BF&spm=1018.2226.3001.4187](https://blog.csdn.net/JAN6055/article/details/123801017?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522168924438316800188572717%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=168924438316800188572717&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_click~default-2-123801017-null-null.142%5ev88%5econtrol_2,239%5ev2%5einsert_chatgpt&utm_term=c%2B%2B%E6%A8%A1%E6%9D%BF&spm=1018.2226.3001.4187)

8.map用法及赋值（这东西就是键值对，一个键对应一个值）

std::map<std::string, std::string> mngConfig; 给mngConfig赋值：1）insert方法，2）operator[]方法

1）insert方法：

std::map<std::string, std::string> mngConfig;

mngConfig.insert(std::make\_pair("key1", "value1"));

mngConfig.insert(std::make\_pair("key2", "value2"));

// 继续插入其他键值对...

2）operator方法

std::map<std::string, std::string> mngConfig;

mngConfig["key1"] = "value1";

mngConfig["key2"] = "value2";

// 继续赋值其他键值对...



9.C++正则表达式（做ut的时候遇到的。一般用来在源字符串中匹配东西，基本都是关键字、符号之类的）

头文件：#include <regex>

主要用法：regex\_match()，regex\_serach()，regex\_iterator

1）regex\_match()：比较源字符串和正则表达式，匹配返回true，否则返回false

2）regex\_serach()：在输入的字符串中提取匹配的字符串，就是 在源字符串中匹配正则表达式，将匹配的结果放到一个对象里。

语法：\  这是转义字符，\w 匹配一个 字母、数组、或者下划线  ；\W 匹配一个 字母、数组或下划线  以外的字符 ； \s  匹配空白字符，如空格、tab键、制表符  ， \S  匹配非空白字符 ；\d  匹配数字字符（0到9） ；  \D  匹配非数字字符

\b  匹配单词的边界 ；\B  匹配非单词边界  ； \\  匹配  \ 本身；\n  匹配换行符  ； \r  匹配回车符

10.迭代器

用途：用来遍历容器的。（一般都是遍历容器，有正向迭代和反向迭代，正向：指向容器开头，然后++，从前往后遍历 ；逆向迭代：指向容器结尾，然后++，从后往前遍历）

1.map<int , string > m; m.insert(1,"hello");m.insert(2,"hh");    //这里是创建一个map对象m，然后插入数据

map<int , string>::iterator  m1;  //创建一个正向的迭代器对象，

for(m1=m.begin(); m1 != m.end() ;  m1++)  //让它去遍历上面的m容器，用begin , end 去遍历，然后打印数据，结果是：1  hello    2  hh

map<int ,string>::reverse\_iterator m2;  //创建一个反向的迭代器对象，让它去遍历上面的m容器，用rbegin，rend去遍历，然后打印数据，结果是 ： 2  hh   1  hello

11.函数指针的用法

看代码的时候发现的，在一个模块的 A.h文件声明一个全局的函数，然后在相应的 A.cpp里面定义它，将这个函数作为参数传到其他模块，其他模块typedef一个函数指针，参数和A.h里面声明的函数参数一致，然后用函数指针声明一个变量，用这个变量指向传入的参数（也就是A.h里面的那个全局的函数）。

12.遍历map容器( 用  for(auto const & xxx : 容器)     ，这个xxx就是一个临时的对象，并且是auto、系统自动匹配生成的类型  )

map<std::string s1,std::string s2> mm;

mm.insert(std::make\_pair("key1","value1"));

mm.insert(std::make\_pair("key2","value2"));

mm.insert(std::make\_pair("key3","value3"));

for(auto const & m : mm)

{

std::cout << m.first << m.second <<std::endl;

}

13.sprintf函数的用法

作用：将格式化的数据存入字符串中

例子：

char buffer[1024] = {0};

sprintf(buffer,"this is a person's name  %s  ,  age  is  %d\n" , "zhang san" , 19 );

printf("%s\n",buffer);

打印结果： this is a person's name  zhang san  ,   age  is  19

14.strtol函数

C 库函数strtol原型为：

     long int strtol(const char \*str, char \*\*endptr, int base) ；

其运行过程为：

    把参数 str 所指向的字符串根据给定的 base 转换为一个长整数（类型为 long int 型），并将该长整数转换为十进制数返回，如果没有符合的结果，则返回一个0值；

    base 必须介于 2 和 36（包含）之间，或者是特殊值 0；

    str中不符合base的部分存储于\*endptr中。  
例子：

char buffer[] = "14279462Start111";

char \*ptr;

printf("%d\n", strtol(buffer,ptr,8) );   //把字符串buffer中的数字看作是8进制，然后转换成10进制输出出来，因为函数的参数3是8，所有有效的数字是0,1,2,3,4,5,6,7，

printf("%s\n",ptr);

输出结果：

791

9462Start111

15.perror函数

这是个库函数，调这个函数会直接把错误提示字符串打印出来。如果想在错误提示字符串前添加一些提示，可以这样：perror("attention !!!")，这样attention!!!就会出现在错误提示字符串前面。

16.rfind函数

查找函数。find函数是正向匹配、正向查找。

rfind函数是逆向匹配、逆向查找。

例子：

string str = "abcdeabcda";

int32\_t ret = str.rfind("a");

那么：ret值为 9 ，也就是最后一个a的下标

如果是str.find("a");的话，返回值就是 0 ，也就是第一个a的下标

17.iperf3

用途：测试 丢包率 和 带宽

例子：iperf3 -c 192.168.0.1 -u -b 100M -t 60，表示在60秒内以100Mbps的带宽测试UDP丢包率。

18.string类型调用compare函数（三个参数，第一个参数是起始索引，第二个参数是要比较的长度，第三个参数是要和源字符串比较的字符串）

string.compare函数可以用来比较字符串的不同部分，并通过索引来指定比较的范围。根据引用和引用的描述，该函数的语法如下： string.compare(start-index, end-length,‘C-string’)。 其中，start-index表示要比较的起始索引，end-length表示要比较的长度，'C-string’表示要与字符串进行比较的C字符串。通过指定起始索引和比较长度，可以比较字符串的指定部分。

例子：

std::string buffer[] = "hello,world";

buffer.compare(6,2,"wo");

也就是从下标6开始，比2个字符，即  world  与wo  相比，比较两个字符，=》 wo  == wo   , 所以结果为0

19.echo命令

echo  1  >  /usr/test.txt     这是把 1 写入test.txt中，如果test.txt中有内容，系统会先将改文件中内容清除，然后把 1 写入

echo  2  >>  /usr/test.txt    这是把2 写入test.txt中，如果test.txt中有内容，系统会在该内容后另起一行，写入 1

20.kill

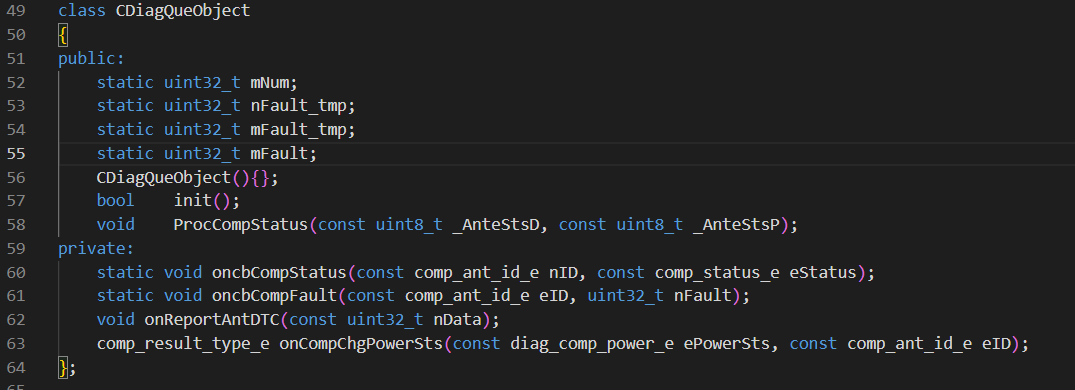
A screenshot of a computer screen

Description automatically generated

21.

一个类里(.hpp文件中)声明了静态成员变量，需要在.cpp文件中定义该变量的值。且需要加类型。如下，在类中声明了static uint32\_t mNum，需要在c.pp里面定义  uint32\_t CDiagQueObject::mNum = 0;

一个类里声明了静态成员函数，在类外定义的时候，该静态成员函数里面只能使用静态成员变量，使用非静态成员变量会报错。



A screen shot of a computer program

Description automatically generated