**考 察 报 告**

**学生姓名： 王冠桥 学 号： 201922230108 指导教师：丁旭阳**

**实验地点： 主楼A2-412 实验时间：2019/12/11**

**一、实验室名称：**

Linux环境高级编程实验室

**二、实验项目名称：**

服务器故障的同步转移实现

**三、实验学时：**

4学时（附加课外课时）

1. **实验内容：**

某企业自己运营某Web应用，服务器采用Linux系统。平时主要由A服务器提供业务服务，当A服务器发生业务故障时，自动切换到B服务器提供业务服务，并且将A服务器当前还未执行完的程序同步到B服务器继续执行（假定A服务器发生业务故障时，网络功能不受影响，仍可以获取当前执行信息）。

**五、实验步骤：**

**1、构建一个web网页**

我们在实验三中实现了客户端和服务端的TCP连接，但是没有web服务器的实现过程，所以仍要从网页与服务器的过程中提取数据，由于我们的任务是实现两个服务器的服务数据的转移，所以我们至少需要获得网页上传输的数据，因此我们需要实现web到后端的数据传输，于是需要一些预备知识。

**一、TCP的三次握手和四次挥手**

如图1.1，是TCP的有限状态机：

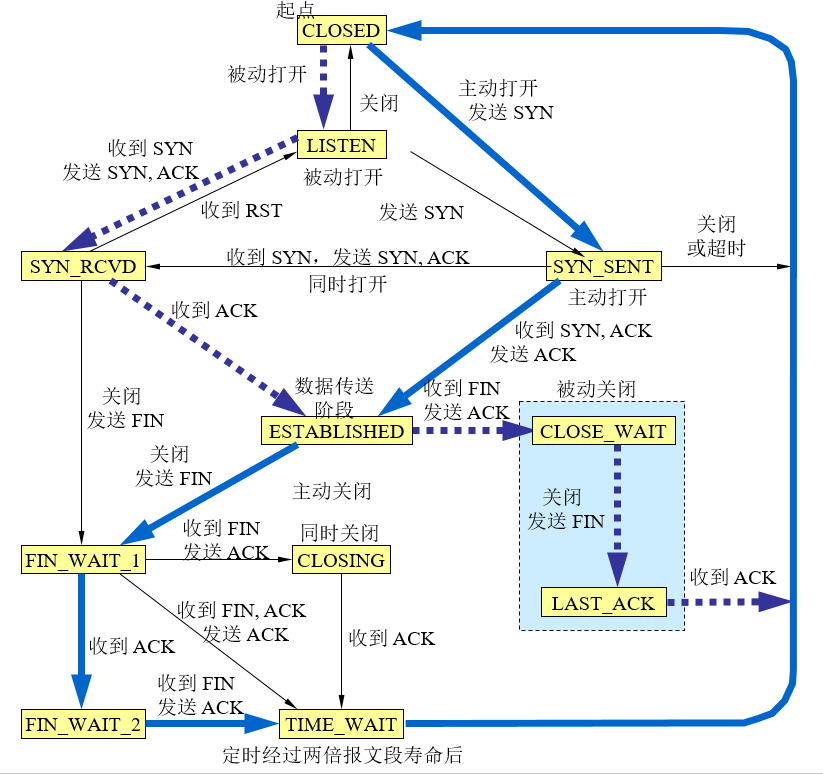


图1.1 TCP有限状态机

三次握手：

1、服务器需要准备好接受外来连接，通过socket bind listen三个函数完成，然后我们称为被动打开。

2、客户则通过connect发起主动连接请求，这就导致客户TCP发送一个SYN（同步）分节去告诉服务器客户将在待建立的连接中发送的数据的初始序列号，通常SYN不携带数据，其所在IP数据只有一个IP首部，一个TCP首部以及可能有的TCP选项。

3、服务器确认客户的SYN后，同时自己也要发送一个SYN分节，它含有服务器将在同一个连接中发送的数据的初始化列序号，服务器在单个分节中发送SYN和对客户SYN的确认

4、客户必须去确认服务器的SYN

四次挥手：

1、某一个应用进程首先调用close，称为该端执行主动关闭，该端的TCP会发送一个FIN分节，表示数据已经发送完毕

2、接到FIN的对端将执行被动关闭，这个FIN由TCP确认，它的接受也作为一个文件结束符传递给接收端应用进程（放在已排队等候该应用进程接收的任何其他数据之后），因为FIN的接收意味着接收端应用进程在相应连接上已无额外数据可以接收

3、一段时间后，接收到这个文件结束符的应用进程会调用close关闭它的套接字，这会导致它的TCP也要发送一个FIN

4、接收这个最终FIN的原发送端TCP（即执行主动关闭的那一端）确认这个FIN

**二、线程池：**

这里使用的是半同步／半反应堆线程池。该线程池通用性比较高，主线程一般往工作队列中加入任务，然后工作线程等待后并通过竞争关系从工作队列中取出任务并且执行。而且应用到服务器程序中的话要保证客户请求都是无状态的，因为同一个连接上的不同请求可能会由不同的线程处理。若工作队列为空，则线程就处于等待状态，就需要同步机制的处理。

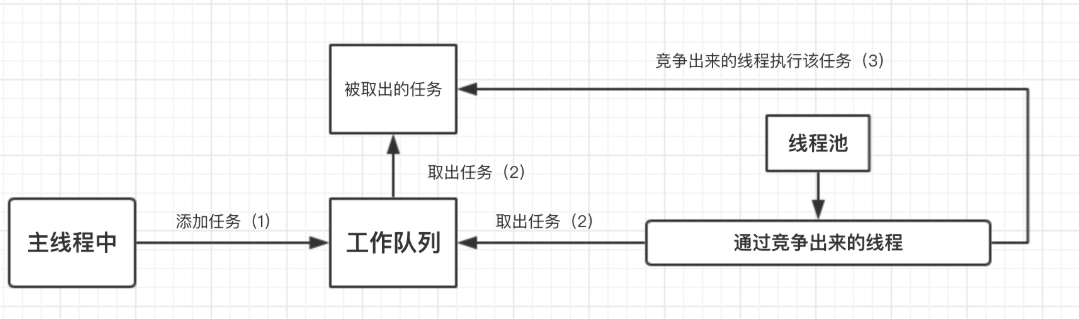


图1.2 线程池工作机制

其实这里并不太需要线程池的作用，因为只有一个web网页，访问的服务器也只有一个，只需要很少的流量访问，但是由于可能对网页的访问量过大，内存开销比较大，所以可以部署一个线程池来进行实现访问。



图1.3 线程池部署

如图1.3以及图1.4所示，分别是线程池的部署以及使用，当然，还有一些线程增加以及线程封装的一些函数，都放在代码里了。



图1.4 线程池执行

**三、同步机制的包装类**

因为采用了线程池，就相当于用了多线程编程，此时就需要考虑各个线程对公共资源的访问的限制，因为方便之后的代码采用了三种包装机制，分别是信号量的类，互斥锁的类和条件变量的类。在服务器中我使用的是信号量的类。其中信号量的原理和System V IPC信号量一样，所以我们只需要使用信号量的互斥就可以实现，简言之，就是进行加锁和减锁操作。

图1.5显示了我们mylock.h文件中的互斥量的一些操作声明:



图1.5 互斥量的操作声明

声明了这些互斥量操作之后，就是对这些操作的具体实现了，例如通过pthread\_mutex\_lock函数和pthread\_mutex\_unlock函数对进程进行加锁和解除锁，因此不需要花费太多的功夫。

在完成了一些基本函数的构造后，我们就可以开始构建一个网页了，构建网页的基本思路如下:

1、首先创建和客户端的连接

2、服务器通过客户端的HTTP请求解析来判断返回何种结果.HTTP解析是以行为单位的，前提条件是根据\r\n来判断是否完整度入一行，若完整读入一行了那么就可以进行解析了。

3、通过HTTP请求的解析后，在写缓冲区写如HTTP响应，发送给客户端（HTTP应答包括一个状态行，多个头部字段，一个空行和资源内容，其中前三个部分的内容一般会被web服务器放置在一块内存中，而文档的内容通常会被放到另一个单独的内存中）

4、发送响应首行后，就可以发送主要的消息体了

以上思路的主要实现主要封装在myhttp\_coon.h中，如图1.6所示，在myhttp\_coon.h文件中实现了一些http的解析要求以及一些基本的网络要求，比如接口的开启关闭等，最重要的是进行了http协议的解析，因为我们所需要执行的功能会从网页的前端得到http响应，然后传递一些参数给我们的后端，我们的后端接收到了参数之后，才能够实现一些数据转移操作。



图1.6 http\_coon.h函数声明

我们在http\_coon.h中实现一些解析操作，如解析请求行，解析头部信息等等，见图1.7和图1.8，之所以把这两部分的代码单独拿出来，是因为这两部分的代码对我们十分重要，是我们对网页传递到服务器信息的解析，这里需要注意得是，我们需要解析http信息，所以我们需要对http响应所反馈的对象进行分析，对该响应的每部分内容都进行了解，这样我们才能够在解析的时候代码能够写的成功，因此我们可以得到如图1.9所示的http响应格式。

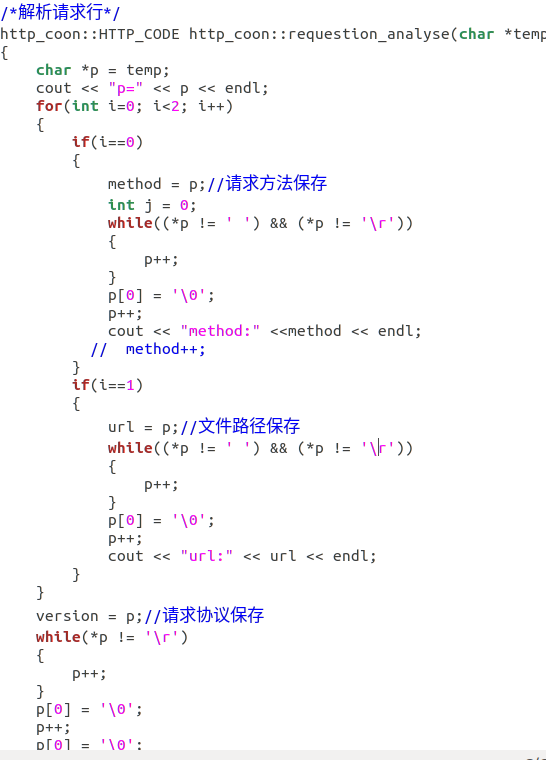


图1.7 解析请求行



图1.8 头部信息解析

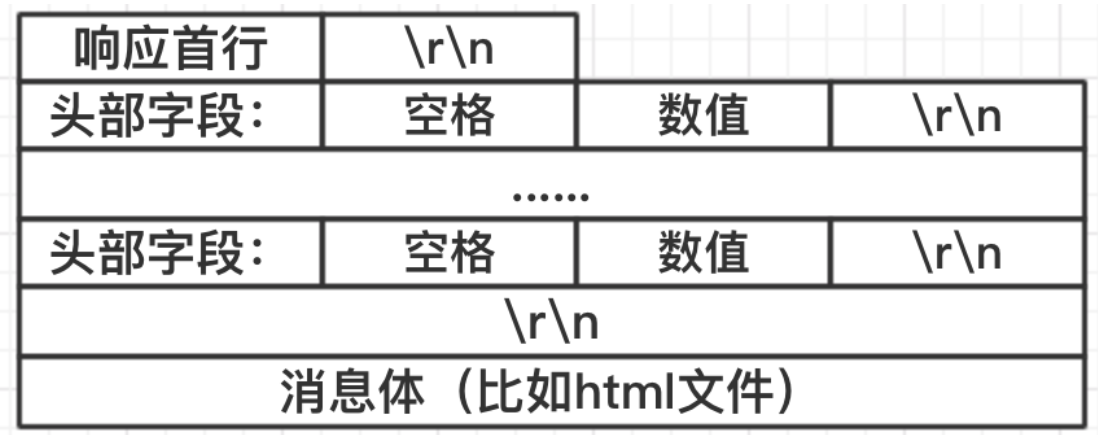


图1.9 http响应格式

有了http的响应格式，我们就能够写出如图1.7和图1.8的解析代码了，解析代码通过将一些关键的http信息进行拆分，将关键的信息反馈给我们。

然后我们需要写一个简单的web来对服务器进行响应，如图1.10所示，我们做了一个简单的登录注册界面，只需要输入账号和密码即可，实际上只是输入一个简单的两行字符串即可，这里我们随便输入两个33。

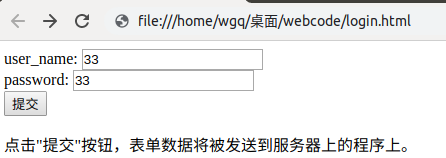


图1.10 http响应格式

提交点击，我们可以从命令行中得到我们传递的数据，如图1.11所示，我们通过http协议得到了两个数据，显示在第一行，分别是33和33，这里我们可知，我们从网页那里得到了我们想要的数据。

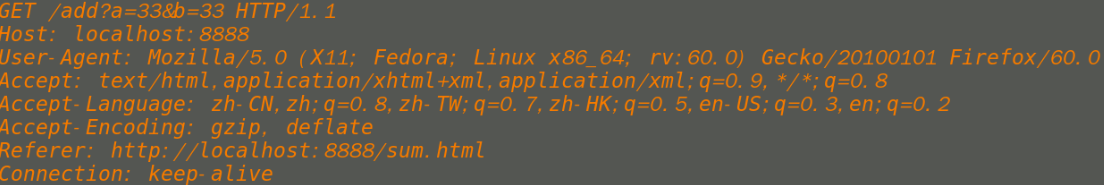


图1.11 http响应格式

至此，我们一个简单的socket网页就写好了，能够实现简单的网页和服务器端的通信。

**2、心跳机制实现两个服务器的连接**

在TCP socket心跳机制中，心跳包可以由服务器发送给客户端，也可以由客户端发送给服务器，不过比较起来，前者开销可能更大，所以我们采用后者。

这里实现的是由客户端给服务器发送心跳包，通过心跳包实现两个服务器之间的交流，基本思路如下：

1） 服务器为每个客户端保存了IP和计数器count，即map<fd, pair<ip, count>>。服务端主线程采用 select 实现多路IO复用，监听新连接以及接受数据包（心跳包），子线程用于检测心跳：

如果主线程接收到的是心跳包，将该客户端对应的计数器 count 清零；

在子线程中，每隔3秒遍历一次所有客户端的计数器 count：

若 count 小于 5，将 count 计数器加 1；

若 count 等于 5，说明已经15秒未收到该用户心跳包，判定该用户已经掉线；

2） 客户端则只是开辟子线程，定时给服务器发送心跳包（本示例中定时时间为3秒）。

如图2.1，我们可以看到心跳包传输关键的代码，在客户端中，我们设置了时间，如果count大于5也即15秒，如果我们仍然没有收到心跳包，那么则代表我们的服务端已经出了故障，所以需要进行数据迁移，由于我们这里是做测试，只能够主动的去断开连接，所以设置了发送15次心跳包就自动断开，因此一定会做出一次数据迁移。

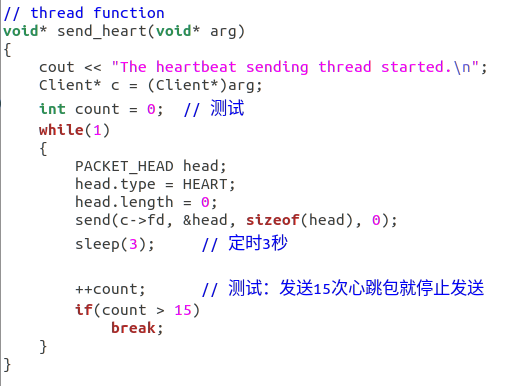


图2.1 客户端心跳包机制

跟我们在实验二中做的内容基本类似，我们在服务端中仍然需要做出一些对客户端的响应，为了判断我们是否传输心跳包成功，所以我们可以在bind函数和listen函数中提醒用户已经开始接受数据，最重要的是recv函数，如果接受到了心跳包，则显示心跳包接受成功，否则就对连接进行关闭，如果关闭了服务端，我们需要在accept中加上检测失败的语句，以反映检测的失败，关键代码见下图2.2，2.3，2.4，展示了服务端的一些操作。

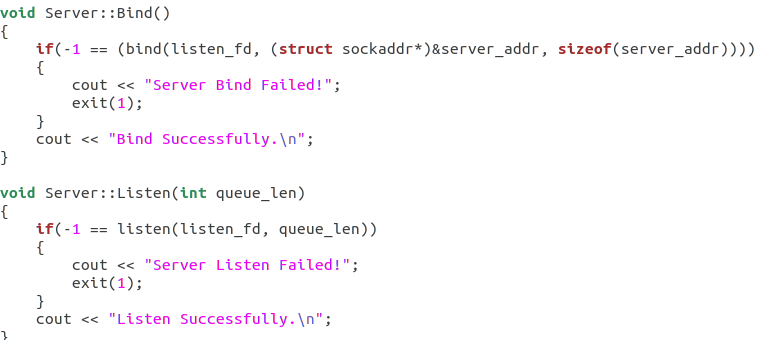


图2.1 Bind函数与Listen函数



图2.2 accetp函数



图2.3 recv函数

通过客户端和服务端的操作，关键代码基本都写完了，然后再类比实验二中的TCP函数，将启动关闭等函数加上，我们就可以进行心跳包的测试，如图2.4，我们可以看到，在15次的心跳包传输之后，强制结束了TCP连接，因此，我们通过心跳包的操作是可行的。

至此，我们实现了心跳包的抛出，从客户端到服务端，控制服务器的自动断开。

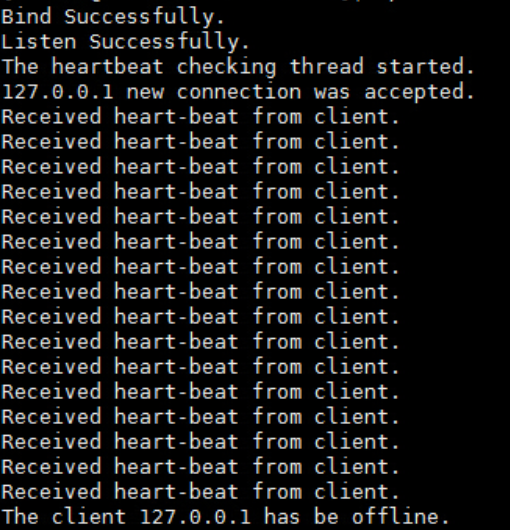


图2.4 测试结果

**3、序列化实现两个服务器的数据迁移**

通过上面的实验，我们只需要实现服务器之间的数据转移即可，我们在实验二中通过序列化实现数据的传输，然后与实验三进行结合，就可以实现服务器之间的数据迁移了。

我们对实验二的版本2进行修改，在版本2中，我们通过C语言的结构化编程实现实现了TCP数据的传输，如图3.1中，我们传输的数据仅仅是简单的一句Hello World。

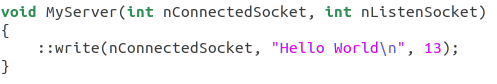


图3.1 MyServer

我们可以想象我们传输的是一个序列化后的文件，因为序列化转化后的data.bin也是一个二进制文件，所以我们可以进行传输，在client端中，进行序列化，然后在Server端里面，将数据进行反序列化，这样就可以实现数据的传输了，如图3.2和3.3

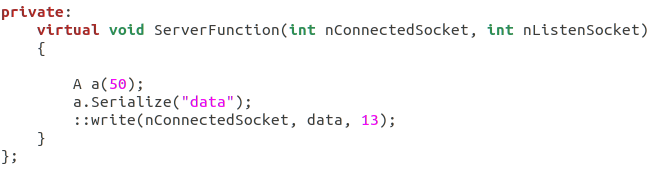


图3.2 服务端传输的数据为序列化后的数据

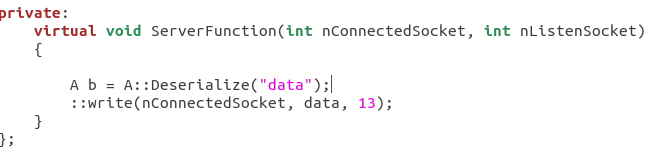


图3.3 客户端得到的数据为反序列化后的数据

通过这样的序列化和反序列化，我们可以实现服务端和客户端之前的数据传输，也能够实现数据的迁移了，测试结果如图3.4，成功的实现了数据的迁移。

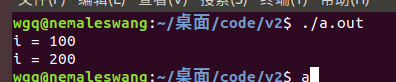


图3.4 测试结果

**4、整合实现web服务器**

通过上面的步骤，我们的web就基本实现了，首先部署两个服务器，分别为服务器A和服务器B，服务器A只执行一小段时间就被服务器B的心跳包停止发送，然后服务器B接受服务，并且在该过程中服务器A把得到数据传递给服务器B。

如图4.1所示，我们提交一个数据为33和33，这时候A服务器得到了我们的数据，如图4.2，因此，我们通过该实验完成了我们想要的内容，实现了服务器间数据的迁移。

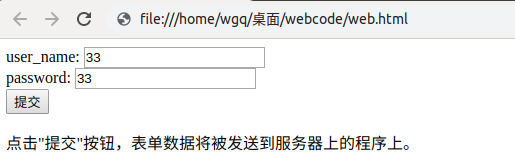


图4.1 传递数据



图4.2 服务器间传递数据