网络编程技术 TCP 编程作业报告

李晨昊 2017011466

2020 年 4 月 4 日

目录

1 运行方式

直接执行`make`即可编译出`server`(需要支持`C++ 17`的编译器)。`server`可以带一个参数执行或者直接执行，参数用于指示监听的地址，不带参数时默认监听 0.0.0.0 地址。总是监听 8000 端口。

我自己已经编写了一个客户端软件来验证它的正确性，只是没有一起提交上来。

2 协议设计

我实现了一个简单的带登录功能的文件上传/下载服务器。

用户建立连接后，首先需要依次发送两个大端序 16 位无符号整数，分别表示接下来要发送的用户名和密码的长度。接着依次发送对应长度的用户名和密码。发送的这些数据之间都没有分隔符，后续的操作也是一样的。服务器将校验用户名和密码，如果用户名已经存在则视为登录操作，取出文件中的密码与之进行比对；否则视为注册操作。

完成登陆/注册后，对于每个操作，用户先发送两个大端序 32 位无符号整数，称之为`arg1`和`arg2`。`arg1`的低两位表示操作类型，总共有四种：

0: 下载文件：arg1 的 17-3 位表示文件名长度。接着发送对应长度的文件名，服务器完成文件读取后发送文件内容。

1: 上传文件：arg1 的 17-3 位表示文件名长度，arg2 表示文件长度。接着依次发送对应长度的文件名和文件，服务器保存这个文件，接着发送信息表示成功。

2: 列出所有文件：服务器发送一个字符串，其内容与`ls`的结果类似。

3: 中断连接。

在以上操作的任何一步中遇到错误时，服务器都会发送一个错误信息并且中断连接。唯一的一个例外是如果试图下载一个不存在的文件，服务器只会发送一个错误信息，而不会中断连接。

所有服务器发送的信息的格式都是先发送两个大端序 32 位无符号整数，第一个整数为零表示操作成功，否则为失败。第二个整数表示接下来的信息的长度，然后发送对应长度的字节串。

3 程序架构设计

3.1 基本设计

我实现的是一个简单的基于`pthread`的多线程服务器，主线程不断监听到来的连接，建立好连接之后派生出线程，每个线程负责一个连接的处理。处理连接的时候，首先进行登录/注册操作，为每个用户建立一个密码文件和一个用于存储用户文件的文件夹，然后开始类似 REPL 的收发过程。

我考虑了一种可能的线程冲突的情景，即多个线程同时执行注册操作时可能会发生不一致，所以我在这里使用了锁。其余地方我都没有使用锁，如果是在服务过程中发生的冲突，可能对用户而言会产生困惑，但是不会影响到服务器的工作状态，所以无需保护。

我为 socket 设定了一些课上提到的参数，包括`SO_REUSEADDR`，`TCP_NODELAY`，`SO_LINGER`等，还设置了忽略`SIGPIPE`信号，detach 派生出的线程。

3.2 资源管理

`C++`中提倡用 RAII-style 来管理和释放资源，但是我们往往需要和文件描述符，`mmap`的内存等打交道，这些在标准库中并没有直接的管理者，如果都要用 RAII-style 的类管理起来的话会增加很多代码量，所以我选择了一种折衷的办法，仍然需要程序员手动指定释放资源的操作，但是不需要指定释放资源的时机，而是在当前代码块结束后自动释放：

```
// need C++17 template deduction guide
#define DEFER(f) Defer DEFER_0(__deferred)([&]() { f; })

template <typename F>
struct Defer {
  F f;
  Defer(F f) : f(f) {}
  ~Defer() { f(); }
};
```

这是模仿Go的资源管理管理方式，不过实现的机制和Go并不一样。在一个语句块中使用DEFER(f)，就会在这个地方定义一个保存了这个操作的变量，在它析构时会执行这个操作。这里DEFER_0宏是用来给变量名字加上一个计数器后缀的，这样可以允许一个块中多次使用DEFER。

3.3 错误处理

在我的错误处理的设计中，最核心的部分是TRY宏：

```
#define TRY(op, require) \
  ({ auto __x = op; if (!(__x require)) return perror(#op), false; __x; })
```

这就相当于Rust中的try!宏或者是?运算符的一个非常简易的模拟。它的作用是在一个返回bool值的函数中执行一个操作，如果返回值不满足要求则输出错误信息 (这不是必要的，仅做调试用途) 并返回false，中止当前的执行并且把错误信息报告给上层；如果满足则将这个返回值作为一个表达式参与后续操作。一个简单的例子如下，这是我的上传文件的实现的一部分：

```
if (![=] {
  Buffer p = get_path(connfd, arg1 >> 2 & 0xFFFF);
  int fd = TRY(openat(dirfd, p.get(), O_RDWR | O_CREAT | O_TRUNC, 0666), >
   0); // 打开文件，只有返回值 > 0时才继续操作
  DEFER(close(fd));
  TRY(ftruncate(fd, arg2), == 0); // 调整文件大小，检测返回值为0后丢弃之
  ...
  return true;
}()) { ERRMSG("failed to put"); } // 有一个操作失败了，向用户发送错误信息
```