**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：董文龙**

**学 号：202222080411**

**指导教师：丁旭阳**

**实验地点：主楼B412**

**实验时间：2022年12月6日**

**一、实验室名称：**

Linux环境高级编程实验室

**二、实验项目名称：**

课程考察

**三、实验学时：**

4学时

**四、实验目的：**

考察学生对本课程学习情况，让学生基于Linux环境综合使用序列化、Socket编程和插件等知识开发较为复杂的项目，使学生能够更好地了解知识之间的关系，提高学生综合运用知识的能力。

**五、实验内容：**

某公司受A、B两所学校委托，需要为A、B两所学校开发一个在线点餐平台。平台要求如下：

1. 考虑到两所学校所在地不同，为保证短时间内平台的并发流量在可承受范围内，拟搭建S1、S2两台服务器，其中S1服务器面向A学校提供服务，S2服务器面向B学校提供服务；
2. 当S1或S2其中一台服务器异常时，其业务需要快速迁移至另一台正常的服务器，以保证业务不中断；
3. 当服务器迁移后，两个学校的业务不能出现混乱，即不能因为迁移导致A校学生可以购买B校食品；
4. 为适应餐饮食品的动态增加，平台需支持管理员新增、修改、删除相关食品信息；
5. 支持学生在线选购商品和结算。

**六、实验步骤：**

**1.项目结构如下：**

|  |
| --- |
| - .vscode(vscode 编辑器配置文件)  - others（第三方库）  - json(使用json进行数据结构的序列化和反序列化)  - src（源码相关）  - client（客户端）  - server1（服务器A）  - server2（服务器B）  - .clang-format（vscode格式化文件） |

**2. 客户端相关实现内容**

2.1 选择连接的服务器A/B

|  |
| --- |
| cout << "请选择A/B服务器:";      string name;      cin >> name;      int money = 10000; |

2.2定义一些数据结构

|  |
| --- |
| // 定义服务器 sockaddr\_in      struct sockaddr\_in servaddr;      memset(&servaddr, 0, sizeof(servaddr));      servaddr.sin\_family = AF\_INET;      if (name == "A") {          servaddr.sin\_port = htons(11111); // 服务器端口      } else if (name == "B") {          servaddr.sin\_port = htons(22222); // 服务器端口      }      servaddr.sin\_addr.s\_addr = inet\_addr("127.0.0.1"); // 服务器 ip，inet\_addr 用于 IPv4 的 IP 转换（十进制转换为二进制） |

2.3 尝试连接选择的服务器，如果连接出错，去访问备用服务器。

|  |
| --- |
| // 如果连接正常的服务器失败，就去连接备用服务器。      if (connect(sock\_cli, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {          servaddr.sin\_port = htons(61111); // 服务器端口          if (connect(sock\_cli, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {  // do something |

2.4 进行食品的购买

|  |
| --- |
| while (true) {              size\_t res = recv(sock\_cli, recvbuf, sizeof(recvbuf), 0); /// 接收              if (res <= 0) {                  cout << "发送出错" << endl;                  break;              }              cout << recvbuf << endl;              cout << "请输入购买的序号:";              string buf;              cin >> buf;              res = send(sock\_cli, buf.c\_str(), strlen(buf.c\_str()), 0);              if (res <= 0) {                  cout << "发送出错" << endl;                  break;              }              res = recv(sock\_cli, recvbuf, sizeof(recvbuf), 0); /// 接收              if (res <= 0) {                  cout << "发送出错" << endl;                  break;              }              string price = recvbuf;              cout << "购买成功" << endl;              money -= stof(price);              cout << "余额：" << money << endl;              // 接受或者发送完毕后把数组中的数据全部清空（置0）              memset(sendbuf, 0, sizeof(sendbuf));              memset(recvbuf, 0, sizeof(recvbuf));          } |

**3. 服务端A相关实现内容**

3.1 Server1结构体定义

|  |
| --- |
| class Server {  public:      // 服务器名字，IP，端口      string name\_;      string ip\_;      int port\_;      // 标志      int flag\_;  // 是否有新的内容写入文件了      int flag2\_; // 是否另外一个服务器发生故障了      // 菜单数据转成的字符串      string data\_str\_;      // 菜单数据      vector<Food> foods\_;      // 构造函数      Server();      Server(const string name, string datafile);      // 析构函数      ~Server();      void addFood();      void deleteFood();      void updateFood();      void show();      void saveData();      void writeFileFromString(const string &filename, const string &body);  private:      //读取json文件返回json对像      Json::Value readJsonFile(const string &filename);      //把json对像写入文件      void writeJsonFile(const string &filename, const Json::Value &root);  }; |

3.2 食品结构体定义

|  |
| --- |
| // 食品  struct Food {      string name; // 食品名称      float price; // 食品价格      int stock;   // 食品库存  }; |

3.3 服务器相关函数实现

* 添加食物

|  |
| --- |
| void Server::addFood() {      cout << "添加一个食物" << endl;      Food food;      cout << "请输入食物名字" << endl;      cin >> food.name;      cout << "请输入价钱" << endl;      cin >> food.price;      cout << "请输入库存" << endl;      cin >> food.stock;      this->foods\_.push\_back(food);  } |

* 删除食物

|  |
| --- |
| void Server::deleteFood() {      if (foods\_.size() < 1) {          return;      }      cout << "删除一个食物" << endl;      for (int i = 0; i < this->foods\_.size(); i++) {          cout << i << ":" << this->foods\_[i].name << " 价钱:" << this->foods\_[i].price << " 库存:" << this->foods\_[i].stock << endl;      }      cout << "请输入要删除的序号:" << endl;      int index;      cin >> index;      this->foods\_.erase(this->foods\_.begin() + index);  } |

* 修改食物

|  |
| --- |
| void Server::updateFood() {      cout << "修改一个食物" << endl;      cout << "请输入要修改的序号:";      int index;      cin >> index;      cout << endl           << "请输入修改后的名字:";      string name;      cin >> name;      cout << endl           << "请输入修改后的价钱:";      float price;      cin >> price;      cout << endl           << "请输入修改后的库存:";      float stock;      cin >> stock;      foods\_[index].name = name;      foods\_[index].price = price;      foods\_[index].stock = stock;  } |

* 显示所有食物

|  |
| --- |
| void Server::show() {      for (int i = 0; i < this->foods\_.size(); i++) {          cout << i << ":" << this->foods\_[i].name << " 价钱:" << this->foods\_[i].price << " 库存:" << this->foods\_[i].stock << endl;      }      cout << "请按任意键继续\n"           << endl;      cin.get();      cin.get();  } |

* 重点在于下面的函数，当Server函数启动时，会启动三个线程。

第一个线程为点菜线程，用来与客户端交互。

|  |
| --- |
| while (true) {          conn = accept(ss, (struct sockaddr \*)&client\_addr, &length);          // 如果accpet成功，那么其返回值是由内核自动生成的一个全新描述符，代表与所返回客户的TCP连接。          // accpet之后就会用新的套接字conn          if (conn < 0) {              perror("connect");              exit(1);          }          char buffer[1024 \* 1024];          while (true) {              memset(buffer, 0, sizeof(buffer));              string menu;              for (int i = 0; i < s->foods\_.size(); i++) {                  menu += to\_string(i) + ":" + s->foods\_[i].name + " 价钱:" + to\_string(s->foods\_[i].price) + " 库存:" + to\_string(s->foods\_[i].stock) + "\n";              }              // 发送菜单给客户端              size\_t res = send(conn, menu.c\_str(), strlen(menu.c\_str()), 0);              if (res <= 0) {                  cout << "发送出错" << endl;                  break;              }              // 从客户端接收数据              size\_t len = recv(conn, buffer, sizeof(buffer), 0);              if (len <= 0) {                  cout << "接收错误" << endl;                  break;              }              string number = buffer;              int num = stoi(number);              s->foods\_[num].stock--;              // 将处理后的数据发送给客户端              res = send(conn, to\_string(s->foods\_[num].price).c\_str(), strlen(to\_string(s->foods\_[num].price).c\_str()), 0);              if (res <= 0) {                  cout << "发送出错" << endl;                  break;              }          }          close(conn); // accpet 函数连接成功后还会生成一个新的套接字描述符，结束后也需要关闭      } |

第二个线程用来发送本服务器菜单到对方服务器。

|  |
| --- |
| // cout << "启动发送" << endl;      while (true) {          if (s->flag\_ == 1) {              s->flag\_ = 0;              // cout << "发送" << endl;              size\_t res = send(sock\_cli, s->data\_str\_.c\_str(), strlen(s->data\_str\_.c\_str()) + 1, 0);              if (res <= 0) {                  cout << "发送出错" << endl;                  break;              }          }          usleep(56);      } |

第三个线程用来接收对面服务器菜单到本机服务器。

|  |
| --- |
| // cout << "启动接收" << endl;      while (true) {          memset(buffer, 0, sizeof(buffer));          int len = recv(conn, buffer, sizeof(buffer), 0);          if (len <= 0) {              cout << "对方出现异常" << endl;              s->flag2\_ = 1;              break;          }          s->writeFileFromString("another-server.json", buffer);          // cout << buffer << endl;      } |

**3. 服务端B相关实现内容**

服务器B与服务器A的内容基本一致，只有端口不一致。A的两个端口为33333和44444，B的端口则为44444和33333相互对应。

**4.main\_server文件**

两个服务器中都会有这个文件，主要用来处理输入和调用 Server中的函数，主要为商家所服务。

1. 执行可执行程序

|  |
| --- |
| if (argc != 2) {          cout << "Usage-----./server1 A or ./server2 B" << endl;          return 0;      }      Server \*s1 = new Server(argv[1], "");      Server \*temp = 0;      unsigned int number;      unsigned char ch;      string server\_name; |

1. 启动异常检测线程，检测对方服务器是否故障。

|  |
| --- |
| void \*start(void \*args) {      Server \*s = (Server \*)args;      while (1) {          if (s->flag2\_) {              if (s->name\_ == "A") {                  s2 = new Server("B", "another-server.json");              } else {                  s2 = new Server("A", "another-server.json");              }              break;          }          usleep(56);      }      return 0;  } |

1. 处理商家操作

|  |
| --- |
| while (true) {          if (s2 != 0) {              cout << "请输入A/B来选择任一服务器";              cin >> server\_name;              if (s1->name\_ == server\_name) {                  temp = s1;              } else if (s2->name\_ == server\_name) {                  temp = s2;              }          } else {              temp = s1; //切换成当前服务器          }          cout << "1:删除一个商品" << endl;          cout << "2:增加一个商品" << endl;          cout << "3:修改一个商品" << endl;          cout << "4:显示所有商品" << endl;          cout << "请输入操作序号:";          cin >> ch;       //读取序号          cout << "\033c"; //清屏          switch (ch) {          case '1':              temp->deleteFood();              break;          case '2':              temp->addFood();              break;          case '3':              temp->updateFood();              break;          case '4':              temp->show();              break;          default:              std::cout << "error input" << std::endl;          }          // system("cls");          temp->saveData();          cout << "\033c"; //清屏      } |

这个函数会调用一个saveData函数，用来保存当前数据。

|  |
| --- |
| void Server::saveData() {      Json::Value root;      for (int i = 0; i < this->foods\_.size(); i++) {          Json::Value item;          item["name"] = this->foods\_[i].name;          item["price"] = this->foods\_[i].price;          item["stock"] = this->foods\_[i].stock;          root[i] = item;      }      this->writeJsonFile("data.json", root);  } |

同时会调用writeJsonFile函数，讲当前数据写入“data.json”文件

|  |
| --- |
| void Server::writeJsonFile(const string &filename, const Json::Value &root) {      flag\_ = 1;      this->data\_str\_ = root.toStyledString();      Json::StyledWriter writer;      ofstream os;      cout << "保存" << endl;      os.open(filename);      os << writer.write(root);      os.close();  } |

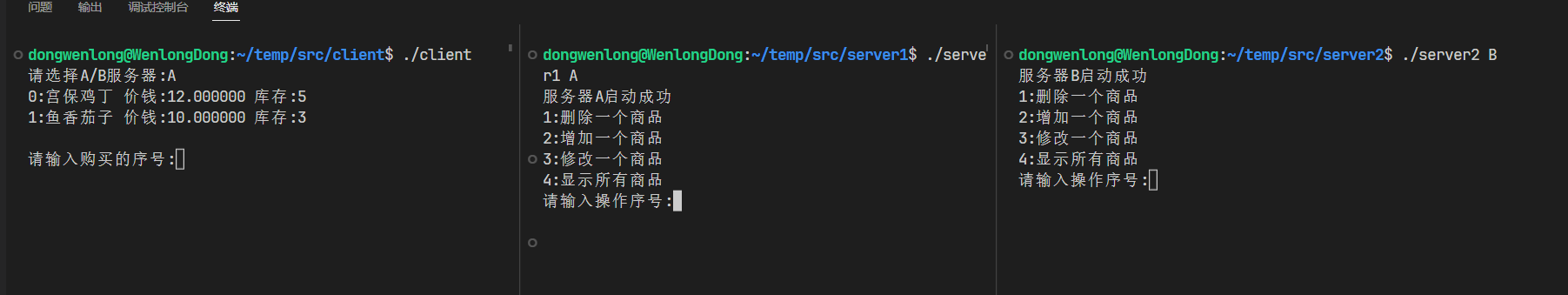
此时flag =1，sendData线程就会将自己的数据发给对方服务器进行保存，防止己方宕机而无法继续提供服务。

另一个服务器的receData线程就会工作。调用writefileFromString函数将发送过来的数据存入到文件中，这样当对方宕机时，直接序列化此文件即可。

|  |
| --- |
| // 把菜单字符串写入文件  void Server::writeFileFromString(const string &filename, const string &body) {      // cout << filename << endl;      ofstream ofile(filename);      ofile << body;      ofile.close();  } |

**5. 进行测试**

1. 启动服务器



2. 学生客户端可以访问不同的商家

图形用户界面

描述已自动生成

图形用户界面

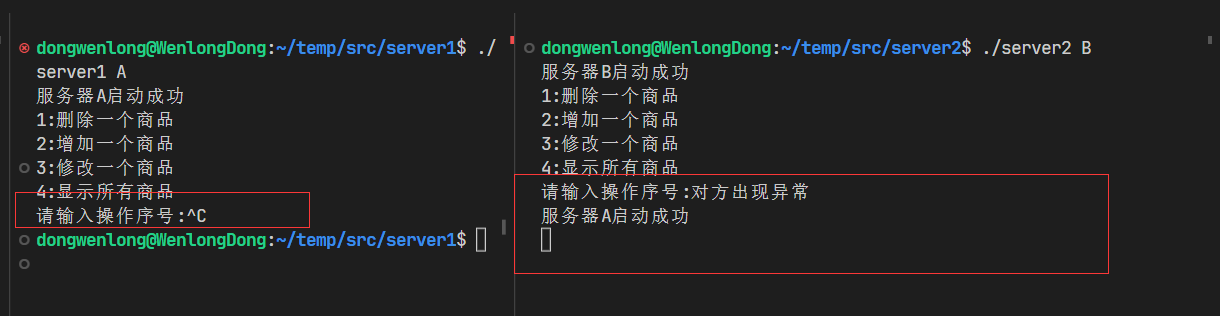
中度可信度描述已自动生成

3. 服务器A增加一个鱼香肉丝

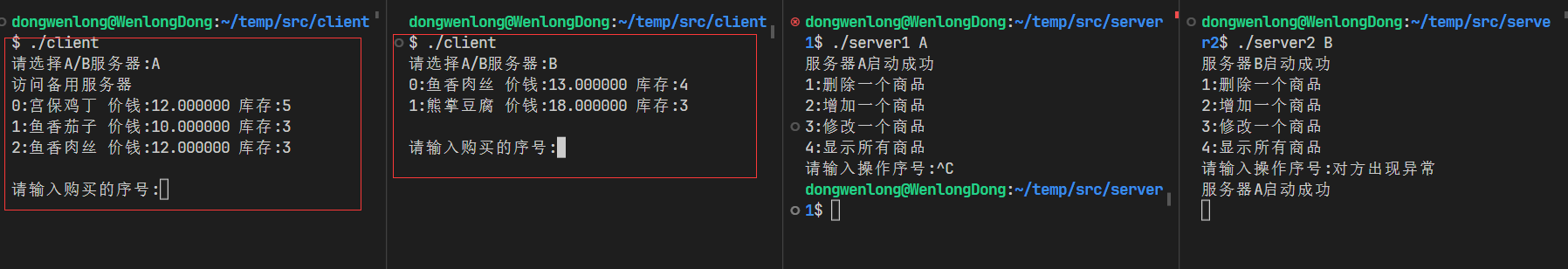
图形用户界面

中度可信度描述已自动生成

然后让服务器A宕机，B检测到异常，启动备用服务器A



此时仍然可以访问B服务器，不会因为序列化而导致程序错误。



**七、总结及心得体会：**

开发了一个能够满足A、B两所学校需求的在线点餐平台。实验中考虑了平台并发流量可承受性、服务器的容灾性、学校业务分离、管理员的操作方便、学生的选购体验等多个因素，使得平台具备了较好的实用价值。

在本次实验中，采用了S1、S2两台服务器的架构，实现了服务器的容灾性；同时，您通过设计与实现了管理员的操作界面，使得管理员可以方便地对食品信息进行新增、修改和删除操作；并且，您通过开发了学生的选购界面，使得学生能够在线购买食品并结算。

**八、对本实验过程及方法、手段的改进建议：**

无

**报告评分：**

**指导教师签字：**