**深 圳 大 学 实 验 报 告**

**课程名称：­ 计算机网络**

**实验项目名称： 实验4 Socket网络编程**

**学院： 数学与统计学院**

**专业： 信息与计算科学（数学与计算机实验班）**

**指导教师： 王璐**

**报告人： 王曦 学号：2021192010 班级： 数计班**

**实验时间： 2023年03月24日 - 2023年04月07日**

**实验报告提交时间： 2023年04月07日**

**教务处制**

|  |  |
| --- | --- |
| **实验目的**   1. 理解UDP 与TCP 套接字的区别； 2. 掌握UDP 和TCP 套接字编程方法； 3. 了解简单网络应用的编程思路； 4. 了解网络编程相关的一些库 | |
| **实验环境**   1. 使用具有Internet 连接的Windows 操作系统； 2. Windows 10 操作系统； 3. Python3.8（base）+Anaconda+pycharm2020.3.5。 | |
| **实验内容：**   1. URL 请求程序； 2. 系统时间查询； 3. 网络文件传输； 4. 网络聊天室。   实验要求：参考讲义学习套接字编程的基础知识，了解网络编程的相关库；掌握编写简单网络应用的技能，依照步骤完成实验内容1——4。 |
| **实验步骤：**  （用文字描述实验过程，并用截图辅助说明）   1. **URL请求程序**   要求: 用Python的HTTP库的Requests实现一个简单的程序, 能够:   1. 请求一个网页, 并存储为HTML文件. 2. 计算所请求的网页的大小.    1. 思路 3. 用requests库中的get()函数可获取URL, 得到一个Response对象, 其中的content属性是网页的内容, 直接打印会显示乱码, 需将编码模式改为utf-8或gbk. 4. 将获取的Response对象编码后, 可以二进制写入文件中. 用os库中的path.getsize()函数可获取文件大小, 单位为Bytes. 用os库中的getcwd()函数可获得文件路径.    1. 实现 (request文件夹 – request.ipynb)   图1: URL请求程序代码   * 1. 运行效果   见实验结果部分.   1. **系统时间查询**   要求: 实现一个基于客户端/服务器的系统时间查询程序; 传输层使用TCP, 打印服务器与客户端的交互过程.  交互过程:   1. 客户端向服务器发送字符串“Time”. 2. 服务器收到字符串后, 返回当前系统时间. 3. 客户端向服务器发送字符串“Exit”. 4. 服务器返回“Bye”, 并结束TCP连接.    1. 思路与实现   下面在本地实现服务器与客户端的交互.   * + 1. 服务器 (Server文件夹)  1. 服务器相关配置 (config.ipynb)   服务器有如下图所示的配置. 将基础配置独立为一个配置文件, 有利于后续管理和维护. 如修改某请求或响应的标识符时, 无需在后续代码中寻找并修改, 只需在config.ipynb中修改.    图2: 服务器相关配置   * 1. 以“REQUEST\_”开头和”RESPONSE\_”开头的参数为服务器与客户端约定的通信协议中请求和响应的唯一标识符, DELIMITER为消息的分隔符(这要求后续实现聊天功能时用户不能输出字符’|’, 否则可能解析错误).   2. SERVER\_IP和SERVER\_PORT是服务器的IP地址和端口号. 因实验仅限在局域网内实现服务器与客户端的交互, 故IP地址设为本机IP地址127.0.0.1, 端口号为本机的任意开放的空闲的端口号, 此处以8090为例.   3. accounts{}为用户账号, 其中key为用户名username, val的第一个参数为密码password, 第二个参数为昵称nickname. 事实上, 此处直接用字典明文存储用户信息是不规范且存在风险的, 应使用MySQL等数据库. 但本实验不涉及数据库操作, 故以方便为主.   4. 其余参数待使用时再解释.  1. 服务器响应协议设置 (response\_protocol.ipynb)   本文件中实现了ResponseProtocol类, 用于拼接服务器响应协议的格式字符串.    图3: 服务器响应协议设置   * 1. response\_login\_result()函数用于生成用户登录结果的字符串.   2. response\_time()函数用于生成返回给用户的时间字符串.   3. response\_exit()函数用于生成返回给请求断开连接的客户端的字符串.  1. 自定义服务器套接字 (server\_socket.ipynb)   本文件中定义了继承于socket.socket类的ServerSocket类, 用于自定义服务器套接字, 此时定义服务器套接字时可完成套接字的初始化, 如设置传输协议类型、绑定地址和端口号、开启监听等.    图4: 自定义服务器套接字   1. 包装服务器套接字 (socket\_wrapper.ipynb)   本文件对数据收发操作、关闭套接字操作进行封装, 采用utf-8编码, 方便后续调用.    图5: 包装服务器套接字   1. 服务器核心 (server.ipynb)    1. 服务器的初始化: 创建服务器套接字, 并将各请求消息对应的处理函数注册到字典中, 同时创建一个用于保存当前在线用户的字典.     图6: 服务器的初始化   * 1. 获取客户端连接并提供服务: 因服务器需同时接收客户端消息和连接其他服务端, 故在主线程中开启一个子线程用于接收客户端消息. 注意主线程中不关闭客户端套接字, 否则子线程无法通信.     图7: 获取客户端连接并提供服务   * 1. 处理客户端请求: 持续接收客户端消息, 解析数据并调用各自的处理函数. 注意字典用[]访问时, key不存在会报错.     图8: 处理客户端请求   * 1. 处理下线客户端: 有客户端下线时, 找到用户名对应的套接字, 将其关闭后打印此时在线用户名单.     图9: 处理下线客户端   * 1. 解析客户端发送的数据: 先按分隔符分割消息, 再按请求类型分割参数, 将参数保存到字典中返回, 方便后续处理函数调用.     图10: 解析客户端发送的数据   * 1. 处理登录请求: 用登录请求中的用户名和密码, 到客户端账号中检查用户名和密码是否匹配, 若匹配, 则将当前用户添加到在线用户字典中. 无论是否登录成功, 都将登录结果返回给客户端.     图11: 处理登录请求   * 1. 处理获取服务器时间、客户端与服务器断开连接请求: 在当前在线用户的字典中查找对应用户名的用户, 将对应消息返回给客户端即可.   图12: 处理获取服务器时间、客户端与服务器断开连接请求  2.1.2 测试代码 (test.ipynb)  该任务中未实现一个严格意义上的客户端, 只实现了一个简易的测试代码, 用于测试服务器与客户端间的通信是否正常.    图13: 测试代码  程序先定义客户端套接字, 并连接到服务器. 程序持续将用户输入发送给服务器, 并将服务器返回的消息解码后打印. 因测试代码未封装客户端的请求协议, 故需手动拼接消息.   * 1. 运行效果   见实验结果部分.   1. **网络文件传输**   要求: 实现一个基于客户端/服务器的网络文件传输程序, 传输层使用TCP, 打印服务器与客户端的交互过程.  交互过程:   1. 客户端从用户输入获得待请求的文件名. 2. 客户端向服务器发送文件名. 3. 服务器收到文件名后传输文件. 4. 客户端接收文件, 重命名并存储在硬盘.    1. 思路与实现 5. 服务器相关配置 (config.ipynb)   本文件中规定了服务器文件所在文件夹”Files”、每次发送的二进制流大小64 Bytes和客户端接收文件的文件夹”Download”.    图14: 服务器相关配置   1. 处理获取服务器文件的请求: 服务器收到客户端请求的文件名后, 在Files文件夹中找到对应的文件, 并将文件大小返回给客户端.     图15: 服务器的Files文件夹    图16: 处理获取服务器文件的请求   1. 服务器分批发送文件: 服务器每次读取SEND\_CHUNK Bytes的文件, 将其以”gbk”格式编码(为防止中文乱码)后发送给客户端. 注意发送时不能发送过快, 因为可能发生消息拥塞和粘连.     图17: 服务器分批发送文件   1. 测试程序 (test.ipynb)   该任务中未实现一个严格意义上的客户端, 只实现了一个简易的测试代码, 用于测试服务器与客户端间的通信是否正常.  测试程序需用”sending”命令手动开启文件传输, 需手动设置保存的文件名  和接收的文件大小, 这些在客户端中都会更新.  测试程序按SEND\_CHUNK分批接收文件.    图18: 测试代码   1. **网络文件传输**   要求: 实现一个基于客户端/服务器的网络聊天程序, 传输层使用UDP, 能实现多个用户的群聊, 客户端打印聊天信息, 服务器打印系统信息.  4.1 思路与实现  4.1.1 服务器 (Server文件夹)  (1) 服务器相关配置 (config.ipynb)  服务器与客户端约定的通信协议中请求和响应的唯一标识符, 其中聊天请求为”0002”, 聊天响应为”1002”.    图19: 服务器相关配置  (2) 服务器响应协议 (response\_protocol.ipynb)    图20: 服务器响应协议   1. 服务器处理聊天请求 (server.ipynb)   服务器收到聊天请求后进行解析, 将消息转发给除发送者外的在线用户. 消息对发送者的展示由客户端完成.    图21: 服务器处理聊天请求  4.1.2 客户端 (Client文件夹)  (1) 客户端相关配置 (config.ipynb)  客户端相关配置与服务器相关配置几乎相同, 区别在于客户端无需存储服务器的文件所在的文件夹、客户端账户信息.    图22: 客户端相关配置  (2) 客户端套接字 (client\_socket.ipynb)  同服务器, 自定义客户端套接字, 继承于socket.socket类, 使得后续创建客户端套接字时可自动完成初始化等功能.    图23: 客户端套接字  (3) 客户端请求协议 (request\_protocol.ipynb)  同服务器, 客户端请求协议实现将用户输入的登录请求、聊天请求、获取服务器时间请求、获取服务器文件请求转化为协议字符串.    图24: 客户端请求协议   1. 客户端登录窗口 (window\_login.ipynb)   本次实验实现了一个如下图所示的客户端登录窗口.    图25: 客户端登录窗口UI  用户输入用户名和密码后登录, 若登录成功则跳转到聊天窗口. 无论用户是否登录成功都会给出提示信息. UI展示见实验结果部分, 本部分仅展示实现.   * 1. 初始化客户端登录窗口: 本文件实现了一个登录窗口WindowLogin类, 实现了对窗口的初始化, 包括设置窗口位置、大小、按钮点击事件等.     图26: 初始化客户端登录窗口   * 1. 填充登录窗口控件: 根据窗口表格, 填充各个控件的位置, 注意添加间隔, 防止各控件粘连在一起.     图27: 填充登录窗口控件   * 1. 控件事件的实现: 实现了从输入框中获取用户输入的用户名和密码、清空输入框、发送登录请求、窗口关闭的事件. 主函数中开启窗口主循环即可显示窗口.     图28: 控件事件的实现   1. 客户端聊天窗口 (window\_chat.ipynb)   本次实验实现了一个如下图所示的客户端聊天窗口, 用于展示不同用户间的聊天信息和系统信息.    图29: 客户端聊天窗口UI  用户可在下方的文本框中输入消息后, 点击右边的Send按钮发送. 在上方的可滚动的文本框中可看到自己和其他用户发送的消息和系统消息. 具体内容将在实验结果部分展示.   * 1. 初始化客户端聊天窗口: 本文件实现了继承于tkinter.Toplevel类的客户端聊天窗口WindowChat类, 将登录窗口作为根窗口(登录后隐藏), 客户端聊天窗口作为Toplevel窗口. 初始化部分完成对窗口的初始化、设置大小和填充组件.     图30: 初始化客户端聊天窗口   * 1. 填充组件: 客户端聊天窗口的填充组件实现如下, 其中定义了三个颜色标签”user\_self”、”user\_others”、”system”, 分别为绿色、蓝色、红色, 同多彩聊天气泡出现以前的QQ的聊天.     图31: 填充组件   * 1. 控件事件的实现: 实现了获取用户输入、发送消息、将消息添加到聊天区、关闭窗口的事件.      1. 对将消息添加到聊天区的事件, 将消息的发送人、发送事件连同消息本身一起添加到聊天区中, 根据发送者的不同显示不同的颜色, 同时向下滚动屏幕, 保证当前显示的是最新的消息.      2. 对关闭窗口的事件, 通过Toplevel.protocol中的“VM\_DELETE\_WINDOW”可设置窗口关闭后执行的命令. 实现该事件是因为用户未必每次都发送”Exit”来断开与服务器的连接, 有时用户直接关闭窗口. 故用户发送”Exit”或关闭窗口时都需关闭套接字、释放资源和更新服务器的在线用户字典等.     图32: 控件事件的实现   1. 客户端 (client.ipynb)   客户端用于实现对各窗口的控制和完成与服务器的数据交互.   1. 初始化客户端: 实现了对登录窗口、聊天窗口的初始化, 设置了各按钮的点击事件, 创建客户端套接字, 初始化消息处理函数. 为方便维护和扩展, 如后续添加新的功能和命令等, 将各种响应及其对应的消息处理函数加入字典中, 这同时也方便后续调用.     图33: 初始化客户端   1. 开启子线程: 与服务器相同, 因客户端也需同时接收服务器消息和监听等待其他客户端的连接, 故需开启多线程. 注意窗口显示的主循环是一个死循环, 需在开启主循环前开启接收消息的子线程.     图34: 开启子线程   1. 登录窗口事件: 实现了清空输入框、发送登录消息到服务器的事件, 其中后者调用RequestProtocol中的request\_login\_result()函数产生协议字符串, 并通过客户端套接字发送到服务器.     图35: 登录窗口事件   1. 发送消息到服务器: 从文本框获取用户输入后清空文本框, 调用聊天窗口的append\_message()函数将消息显示到发送者的聊天区中. 先检查消息是否是/time、/file等特殊命令, 若为特殊命令, 需调用对应的生成协议文本的函数; 否则, 调用聊天请求的生成协议文本的函数. 生成协议文本数据后, 通过客户端套接字发送到服务器.     图36: 发送消息到服务器   1. 接收服务器消息: 客户端在用户发送消息的同时也需接收服务器的消息. 在子线程中的response\_handle()函数用于不断接收服务器消息, 调用消息解析函数后根据响应累心那个调用不同的处理函数.     图37: 接收服务器消息   1. 解析消息: 客户端按分割符分割服务器发来的消息后, 根据响应类型的不同分割出其他参数, 将所有参数保存到一个字典中并返回.   图38: 解析消息   1. 登录响应: 客户端接收服务器发来的登录结果, 检查是否登录成功, 若是则关闭登录窗口, 显示聊天窗口, 同时保存登录用户的昵称, 供将消息添加到聊天区使用; 保存登录用户的账号, 供发送消息使用. 无论是否登录成功, 都弹出提示信息.     图39:登录响应   1. 聊天响应: 客户端收到服务器发来的聊天响应, 即收到服务器转发的其他用户发送的消息时, 将消息以特定颜色展示到聊天区.     图40: 聊天响应   1. 获取服务器时间和获取服务器文件响应: 将实验内容2、3中的获取服务器时间、获取服务器文件封装到客户端中, 客户端自动对消息进行编码和解析, 并输出响应的提示信息. 实现逻辑与实验内容2、3相同, 但无需用户手动设置保存文件名和文件大小.     图41: 获取服务器时间和获取服务器文件响应   1. 退出程序时释放资源: 用户关闭客户端聊天窗口时释放资源并退出程序, 具体地, 先关闭接收消息的子线程(由is\_running的flag记录), 再关闭客户端套接字, 最后调用os.\_exit()函数无报错地退出程序. 若使用exit(0), 会导致客户端关闭后未响应.     图42: 退出程序时释放资源  4.1.3 运行效果  见实验结果部分. |
| **实验结果：**  （给出个人对结果的分析、结论）   1. **URL请求程序** 2. 用户输入URL和保存的文件名后, 程序获取URL并保存到文件baidu.html中.     图43: URL请求程序运行效果1   1. 打开程序输出的路径, 发现文件的路径、大小与程序的输出一致.     图44: URL请求程序运行效果2   1. 用浏览器打开baidu.html, 观察到页面整体框架与百度的网页相同, 但图片和部分CSS样式缺失, 因为获取URL过程中只对网页的文字部分进行获取, 而未获取其中的图片、CSS样式等, 但这对于文字类信息的爬虫已足够.     图45: URL请求程序运行效果3   1. **系统时间查询** 2. 运行服务端, 服务端等待客户端连接. 运行客户端后, 服务端显示连接成功, 同时等待下一个连接, 体现了多线程.     图46: 服务端等待客户端连接   1. 客户端发送登录请求, 请求登录用户名为user1、密码为114514的用户.     图47: 客户端请求登录  服务器响应登录请求, 并打印在线用户字典.    图48: 服务器响应登录请   1. 客户端发送查询服务器时间的请求.     图49: 客户端请求查询服务器时间  服务器响应客户端查询服务器时间请求, 将服务器时间返回给客户端.    图50: 服务器响应客户端查询服务器时间请求   1. 客户端发送断开与服务器的连接的请求.     图51: 客户端请求断开与服务器的连接  服务器响应客户端断开与服务器的连接的请求, 将设置好的message返回给客户端, 并清除在线用户, 此时在线用户字典为空.    图52: 服务器响应客户端断开与服务器的连接的请求  客户端再次发送消息, 程序报错: 连接已断开.    图53: 检查连接是否已断开   1. **网络文件传输**   (1) 测试程序接收大文件(> 64 Bytes)测试: 观察到客户端分批接收文件并写入到”received\_large.txt”中.    图54: 测试程序接收大文件测试  打开测试程序同目录下的”Download”文件夹, 观察到文件大小>587 Bytes, 且文件中包含协议的内容, 这是因为服务器将文件内容编码后生成协议文本发送给测试代码, 而测试代码未对收到的内容进行解析, 这一点在后面的客户端中将完善.    图55: 测试程序接收到的大文件   1. 测试程序接收小文件(<= 64 Bytes)测试: 观察到客户端分批接收文件并写入到”received\_small.txt”中.     图56: 测试程序接收小文件测试  打开测试程序同目录下的”Download”文件夹, 发现收到了未解析的received\_small.txt文件.    图57: 测试程序接收到的小文件   1. **网络聊天室**   (1) 先运行服务器, 服务器等待客户端连接.    图58: 服务器等待客户端连接  (2) 运行两个客户端, 观察到服务器成功连接到客户端, 同时因为多线程, 继续等待其他客户端的连接.    图59: 服务器成功连接到客户端   1. 在一个客户端中登录用户名为user1、密码为114514的账户. 点击Clear按钮可清空输入框, 点击Login按钮可发送登录请求.     图60: 客户端登录账户   1. 无论是否登录成功, 都弹出提示信息.     图61: 登录失败    图62: 登录成功   1. 客户端登录成功后跳转到聊天窗口, 同时服务器打印在线用户字典. 同理在另一个客户端中登录用户名为admin、密码为1919810的账户.     图63: 客户端聊天窗口    图64: 服务器打印在线用户字典    图65: 登录两个客户端   1. 聊天功能测试: 在user1的聊天窗口中输入聊天内容, 点击Send按钮发送, 观察到admin的聊天窗口收到user1发送的内容, 服务器将user1发送的聊天信息转发给其他在线用户. Admin回复聊天内容, 观察到user1收到聊天内容.     图66: user1输入聊天内容    图67: admin收到聊天内容    图68: admin回复聊天内容   1. 获取服务器时间测试: 在admin的聊天窗口中输入”/time”并发送, 观察到服务器返回了服务器时间, 以消息红色展示. 因实验在本机上进行, 故延时较小, 观察到发送时间与获取到的服务器时间相同.     图69: admin查询服务器时间   1. 获取服务器文件测试: 在user1的聊天窗口中输入”/file large.txt”并发送, 观察到服务器收到获取服务器文件请求后分批发送文件,     图70: user1请求获取服务器文件    图71: 服务器分批发送文件    图72: 服务文件与客户端接收到的文件   1. 服务器与客户端的聊天过程见附件视频. |
| **实验小结：**  （实验中出现问题的解决方法，实验心得体会等）  本次实验遇到了如下困难:  (1) 对python面向对象语法不熟悉: 学习优秀的工程文件, 了解python面向对象的设计方法和工程设计规范.  (2) 对多线程不熟悉: 查阅操作系统相关资料, 了解进程、线程、协程间的关系. 查阅python的threading库的说明文档, 了解python中多线程的实现.  (3) 对python的request库不熟悉: 学习了python爬虫的相关知识, 阅读了爬虫项目的代码.  (4) 对python的tkinter库和UI设计不熟悉: 学习了tkinter库的使用教程, 了解Windows窗口设计相关知识.  (5) 对本机的端口和防火墙不熟悉: 复习了计网实验2中的网络命令, 查看、添加、修改本机的端口, 并允许实验用到的端口连接.  (6) 对套接字编程不熟悉: 参照实验指导和参考讲义, 学习了UDP套接字编程、TCP套接字编程, 并用python的socket库实现.  本次实验有如下心得:  (1) 虽然本次实验任务量大、时间长, 实际完成过程中出现了各种各样的bug, 花费了大量的调试时间, 但在本次实验中收获很大, 主要有以下几点:   1. 学习了python面向对象编程. 2. 学习了工程编写规范. 3. 学习了用python实现多线程. 4. 学习了用python开发UI界面. 5. 学习了python爬虫. 6. 学习了套接字编程. 7. 加深了对本机IP、端口的理解. 8. 理解了QQ、微信等聊天软件的聊天功能和文件传输功能的底层原理.   (2) 本次实验还有如下可优化和扩展的地方:   * 1. 未实现丢包时的重发机制.   2. 未实现文件发送时数据包的拥塞控制.   3. 未实现客户端获取的URL或文件不存在时的处理.   4. 添加命令的等级和用户权限功能, 如特定的用户可使用不高于特的级别的命令.   5. 美化客户端, 如在客户端中添加图片, 或采用WPF窗口.   6. 将用户信息存储到MySQL等数据库中.   7. 实现对发送文件的加密和解密, 以及服务器发送时生成文件摘要, 客户端接受后校验.   8. 用数字签名实现对用户的身份认证, 防止伪造用户进行通信.   9. 将程序部署到有公网IP的服务器上, 通过该程序与网友通信. |
| **指导教师批阅意见：**  **成绩评定：**  **指导教师签字：**  年 月 日 |
| 备注： |