**《网络与通信》课程实验报告**

**实验2: Socket通信编程**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 姓名 | 刘彦辰 | | 院系 | | 计算机学院 | | | 学号 | | 21121319 | | |
| 任课教师 | | 刘通 | | | | 指导教师 | 刘通 | | | | | |
| 实验地点 | |  | | | | 实验时间 |  | | | | | |
| 实验课表现 | | 出勤, 表现得分 (10) | |  | | 实验报告  得分 (40) |  | | 实验总分 | | |  |
| 操作结果得分 (50) | |  | |
| 实验目的: | | | | | | | | | | | | |
| 1. 掌握Socket编程过程； 2. 编写简单的网络应用程序. | | | | | | | | | | | | |
| 实验内容: | | | | | | | | | | | | |
| 利用你选择的任何一个编程语言, 分别基于TCP 和UDP 编写一个简单的Client/Server 网络应用程序. 具体程序要求参见《实验指导书》.  要求以附件形式给出:   * 系统概述: 运行环境, 编译, 使用方法, 实现环境, 程序文件列表等； * 主要数据结构； * 主要算法描述； * 用户使用手册； * 程序源代码； | | | | | | | | | | | | |
| 实验要求: (学生对预习要求的回答) (10分) | | | | | | | | | | | 得分: | |
| **Socket编程客户端的主要步骤**  在C语言中进行Socket编程时，客户端的主要步骤通常包括以下几个阶段：   1. 创建Socket： 首先，客户端需要创建一个套接字（Socket），以便与服务器建立连接。可以使用socket函数来创建套接字，指定套接字的类型（如SOCK\_STREAM用于TCP套接字）和协议族（如AF\_INET用于IPv4）。 2. 设置服务器地址和端口： 客户端需要指定服务器的地址和端口号，以便连接到服务器。通常，服务器地址是服务器的IP地址，端口号是服务器监听的特定端口。 3. 连接到服务器： 使用connect函数来连接到服务器。这将建立与服务器的连接。 4. 发送和接收数据： 一旦连接建立，客户端可以使用send函数发送数据给服务器，使用recv函数接收来自服务器的数据。这是进行实际通信的阶段。 5. 关闭Socket： 最后，客户端使用close函数关闭套接字，释放资源。   **Socket编程服务器端的主要步骤**  在C语言中进行Socket编程时，服务器端的主要步骤通常包括以下几个阶段：   1. 创建Socket： 首先，服务器端需要创建一个套接字（Socket），以便监听客户端的连接请求。可以使用 socket 函数来创建套接字，指定套接字的类型（如 SOCK\_STREAM 用于TCP套接字）和协议族（如 AF\_INET 用于IPv4）。 2. 绑定Socket到地址和端口： 服务器端需要将套接字绑定到一个特定的IP地址和端口号上，以便监听客户端的连接请求。通常，服务器会绑定到一个固定的IP地址（如 INADDR\_ANY 表示任意可用的IP地址）和一个特定的端口号。 3. 监听连接请求： 使用 listen 函数来监听客户端的连接请求。服务器端可以指定队列中等待连接的最大数量（通常称为“backlog”），以决定同时处理的最大连接数。 4. 接受连接： 服务器端使用 accept 函数来接受客户端的连接请求。这个函数会阻塞等待连接请求，一旦有客户端连接，它会返回一个新的套接字，用于与客户端通信。 5. 接收和处理数据： 一旦连接建立，服务器端可以使用 recv 函数接收来自客户端的数据，然后根据协议和应用程序的需求进行处理。处理的逻辑可以包括数据解析、业务逻辑执行等。 6. 发送响应数据： 服务器端可以使用 send 函数向客户端发送响应数据，根据协议和应用程序的需求来构造响应。 7. 关闭Socket： 最后，服务器端使用 close 函数关闭套接字，释放资源。 | | | | | | | | | | | | |
| 实验过程中遇到的问题如何解决的？ (10分) | | | | | | | | | | | 得分: | |
| **问题1: 头文件缺失**  实验指导书给出的 C/C++ 代码中, 除了明显易改的错误外, 还有一个比较明显的错误: 没有包含需要的头文件. 具体来说, server.c 中有以下代码:  函数 inet\_ntoa() 的声明在头文件 arpa/inet.h 中. 因此只要在调用前包含此头文件即可.  **问题2: 数据类型强制转换**  实验指导书给出的 C/C++ 代码中, 有如下一段程序让我产生了一些疑问:    需要注意, address 是 sockaddr\_in 类型, 我很好奇为什么它的内存能被解释为 sockaddr. 于是我查看了这两种类型的定义, 如下图所示:    可以看到 sockaddr 有两成员, 其中 sa\_ 就是一个 uint16, 占 2 Bytes; 而 sa\_data 是一个 char 数组, 占 14 Bytes. 因此一个 sockaddr 变量实际占用 16 Bytes.  sockaddr\_in 的构成看似复杂, 可实际上是由一个 2 Bytes 的 uint16 (sin\_), 一个 2 Bytes 的 uint16 (sin\_port), 一个 4 Bytes 的 uint32 (sin\_addr) 和 8 Bytes 的 char array (sin\_zero) 组成.  首先可以确定这两个数据类型的内存占用相同, 都是 16 Bytes; 但是我不太理解, 既然 sockaddr 和 sockaddr\_in 的数据成员不同, 为什么还允许强制类型转换.  后来我仔细想了想, 恍然大悟, C 的编写逻辑和 C++ 不太一样.  在 sockaddr 中, sa\_data 实际上取到了占位的作用, 也就是规定了任何 socket 地址的具体实现中的数据部分都需要 14 Bytes 内容; 而 sockaddr\_in 中没有用满 14 Bytes, 因此添加了 8 Bytes 的 sin\_zero 进行补位; 补位后由于 sockaddr\_in 和 sockaddr 占用内存相同, 可以执行强制的类型转换.  继续反思, 在 C++ 中, 程序员大多不会这样实现代码. 我认为 C++ programmer 会做的是将 sockaddr 写成一个基类 (或者多态类), 让 sockaddr\_in 作为派生类继承 sockaddr; 这样做更加符合人类的思维模式.  这个例子让我看到了 C 和 C++ 不同的特性. C programmer 可能更会从内存 / 底层的角度思考如何编写代码; 而 C++ programmer 会活用 OOP 特性, 从面向对象的角度开发更易扩展和维护的代码.  **问题3: 输入与接受消息并行的实现**  我希望实现输入和接受消息并行, 即用户在输入想要发送给服务器的内容时, 继续接收服务器发来的消息. 我所知的实现方法有两种: 要么用多进程 (Multi-Process), 要么用多线程 (Multi-Treading).  由于本学习在操作系统课程中学习了多进程的使用方法, 我采用了多进程实现该过程.  此外, 我还利用多进程以及跨进程的共享内存实现对用户输入的监控, 即当用户输入某个值 (例如 ‘q’) 后, 对服务端或者客户端的部分进程和功能进行操控.  以下代码是我定义的利用不同 KEY 产生不同共享内存的函数:    **问题4: 多用户同时登录**  通过服务端多线程实现.  **问题5: 关闭某个进程**  通过调用函数 kill() 现. 实际上这么实现非常不好, 但我不会别的, 我只在操作系统课上学了 fork(). 说实话我对网络通信这块的编程不怎么感兴趣, 所以一直没学线程池进程池之类的东西.  **问题6: 客户端用户输入消息和输出接收到的服务器消息冲突**  众所周知, 在控制台 (终端) 输入字符时, 如果同时输出了某些内容, 会导致输入的连续文本被打断.  为了解决这个问题,我设置了跨进程的共享内存, 将该块内存的值作为信号;    如果信号为 “正在输入”, 则将接收到消息存入一个缓存中, 直到输入结束后恢复信号, 缓存中的消息全部输出. | | | | | | | | | | | | |
| 本次实验的体会 (结论) (10分) | | | | | | | | | | | 得分: | |
| 本次实验我除了完成基础的服务端客户端通信外, 解决了如下几个关键问题:   1. **利用 fork() 产生多进程, 实现多客户端对一服务端的通信.** 2. **利用 fork() 产生多进程, 利用共享内存技术实现了统一关闭服务器所有进程, 实时监控用户输入指令等功能.** 3. **实现了客户端在输入模式下的缓存机制, 确保在输入时依旧能够接受服务端信息.**   在解决问题过程中, 我学习到了很多知识, 知识迁移了操作系统课上学到的东西, 完成了本实验.  我其实不喜欢关于网络的编程, 但是我依靠自己的能力写出了能运行的代码. 我承认我写的代码肯定不符合现代网络通信编程的规范, 因为我关闭进程全用的 kill() 函数, 这非常不优雅; 而且我完全没有使用线程池, 进程池, 因为我没学过. 但是我单纯靠学过的 fork() 实现了我想要的功能, 比较开心. | | | | | | | | | | | | |
| 思考题: (10分) | | | | | | | | | | | | |
| 思考题1: (4分) | | | | | | | | | | | 得分: | |
| **你所用的编程语言在Socket通信中用到的主要类及其主要作用.**  我使用的语言是 C/C++. 在C/C++ 中, Socket通信是一种用于网络编程的重要技术, 它允许不同计算机之间通过网络进行数据交换. Socket通信涉及多种类型, 函数和操作, 下面列举了一些主要的类型, 函数以及它们的主要作用.  1. 套接字类型 (Socket Types) :  在C语言中, Socket通信主要使用两种套接字类型: 流套接字 (Stream Socket) 和数据报套接字 (Datagram Socket) .  流套接字 (SOCK\_STREAM) : 这种类型的套接字用于建立可靠的, 面向连接的通信. 它基于TCP协议, 数据传输是可靠的, 确保数据按照发送的顺序到达目标, 适用于需要可靠通信的场景, 如文件传输和网络应用.  数据报套接字 (SOCK\_DGRAM) : 这种类型的套接字用于无连接的通信, 基于UDP协议. 它适用于需要快速数据传输, 但不需要可靠性的场景, 如实时视频流和游戏通信.  2. 套接字创建函数:  socket() : 用于创建一个新的套接字. 它接受参数, 指定套接字的类型 (如SOCK\_STREAM或SOCK\_DGRAM) 和协议族 (如AF\_INET用于IPv4或AF\_INET6用于IPv6) .  3. 服务器端函数:  bind() : 将套接字与特定的IP地址和端口绑定, 以便监听客户端连接请求.  listen() : 用于监听客户端连接请求, 指定允许同时连接的最大客户端数量.  accept() : 接受客户端的连接请求, 创建一个新的套接字用于与客户端通信.  4. 客户端函数:  connect() : 用于与服务器建立连接. 客户端使用此函数连接到服务器的IP地址和端口.  5. 数据传输函数:  send() : 用于将数据从套接字发送到连接的对等方. 在流套接字中, 它通常用于发送连续的数据流.  recv() : 用于从连接的对等方接收数据. 它在流套接字中用于接收连续的数据流.  sendto() : 在数据报套接字中用于向指定的目标地址发送数据.  recvfrom() : 在数据报套接字中用于接收来自指定源地址的数据.  6. 关闭函数:  close() : 用于关闭套接字连接, 释放相关资源.  7. 错误处理函数:  perror() : 用于打印与套接字操作相关的错误信息.  8. 辅助函数:  htonl() : 将32位主机字节序整数转换为网络字节序 (大端字节序) .  ntohl() : 将网络字节序整数转换为主机字节序.  htons() : 将16位主机字节序整数转换为网络字节序.  ntohs() : 将网络字节序整数转换为16位主机字节序. | | | | | | | | | | | | |
| 思考题2: (6分) | | | | | | | | | | | 得分: | |
| **说明TCP和UDP编程的主要差异和特点.**  TCP (Transmission Control Protocol) 和 UDP (User Datagram Protocol) 是计算机网络中最常用的两种传输层协议. 它们是互联网和其他网络通信的基础.  TCP是一种面向连接的协议, 这意味着在数据传输之前要建立连接. 以下是TCP编程的主要特点和差异:   * 可靠性: TCP非常可靠. 它确保从一端发送的数据按照相同的顺序到达, 不会丢失. 如果在传输过程中丢失或损坏了数据段, TCP会请求重新传输, 以确保数据的完整性. * 面向连接: TCP使用三次握手机制在发送方和接收方之间建立连接, 然后才进行数据传输. 这确保了双方都准备好进行通信. * 流式传输: TCP以字节流的形式传输数据. 这意味着发送方将数据视为连续的字节流, 而接收方将其重新组装成原始消息. 这有助于确保数据的顺序性. * 拥塞控制: TCP具有内置的拥塞控制机制, 可调整数据的传输速率, 以避免网络拥塞并确保公平性. 它会根据网络状况自动调整发送速度. * 适用场景: TCP适用于需要可靠, 有序数据传输的应用程序, 如Web浏览, 电子邮件, 文件传输和远程登录. 它确保了数据的完整性和可靠性, 但由于建立连接和拥塞控制的开销, 可能会引入一些延迟.   UDP是一种无连接的协议, 具有一些与TCP相反的特点:   * 不可靠性: UDP不提供数据的可靠性保证. 它只是尽力将数据从发送方传递到接收方, 但不保证数据的到达顺序, 也不提供重传机制. * 无连接: UDP不需要在数据传输之前建立连接. 发送方只是将数据包发送到接收方的IP地址和端口, 而不需要进行握手或建立持久连接. * 数据报传输: UDP将数据分割成较小的数据报, 每个数据报都有自己的目标地址和端口. 这意味着每个数据报可以独立处理, 而不必等待其他数据. * 低开销: 由于UDP不涉及连接建立, 拥塞控制和重传机制, 因此它的开销较低, 适用于对实时性要求较高的应用程序. * 适用场景: UDP适用于需要低延迟, 实时性要求高的应用程序, 如音频和视频流, 在线游戏和VoIP通信. 虽然UDP不提供可靠性保证, 但在某些情况下, 应用程序可以通过实现自己的错误处理和重传机制来弥补这一不足. | | | | | | | | | | | | |
| 指导教师评语: | | | | | | | | | | | | |
| 日期: | | | | | | | | | | | | |