**题目：网络协议栈设计与实现**

**姓名：王怡恒**

**学号：2022300385**

**班号：10012201**

**时间：2024/12/16**

**计算机学院**

**目 录**

[一、实验目的 3](#_Toc14432)

[二、实验要求 3](#_Toc20782)

[三、算法设计 3](#_Toc15056)

[（一）TCP 协议通信模型 3](#_Toc21242)

[（二）接口函数设计与实现 4](#_Toc15471)

[（三）数据结构设计 7](#_Toc9573)

[（四）滑动窗口技术原理 7](#_Toc5488)

[（五）连接建立三握手和连接释放四握手分析 10](#_Toc5113)

[四、运行结果与分析 12](#_Toc32274)

[（一）运行结果 12](#_Toc22933)

[（二）结果分析 12](#_Toc22739)

[五、实验过程发现的新技术问题及优化方向 13](#_Toc18488)

[（一）发现的问题 13](#_Toc2911)

[（二）优化方向 13](#_Toc17371)

[六、参考文献 13](#_Toc12994)

**一、实验目的**

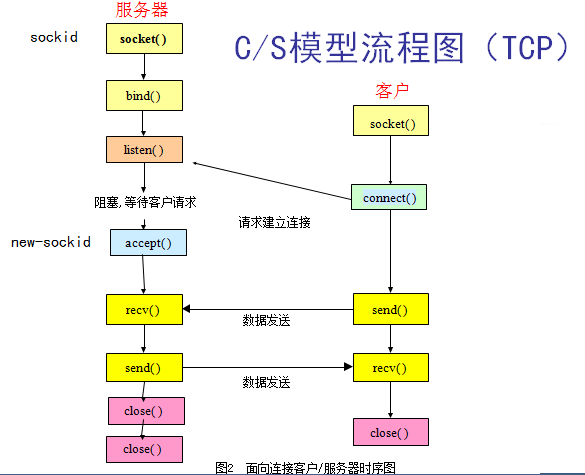
本次实验旨在设计并实现 TCP 协议模块，深入理解 TCP 协议的通信机制、协议流程以及相关接口函数的功能与实现方式，通过实际操作掌握 TCP 协议在网络通信中的应用，能够构建一个简单的基于 TCP 协议的通信系统，并对其进行测试和分析。

**二、实验要求**

1. 按照给定的 TCP 协议通信模型和接口函数设计规范，完成 TCP 协议模块的设计与实现。
2. 编写清晰、详细的实验报告，包括实验目的、实验要求、算法设计、运行结果与分析、实验过程中发现的问题及建议、参考文献等内容。
3. 对实验结果进行准确的记录和深入的分析，能够解释实验现象，评估实验成果。

**三、算法设计**

**（一）TCP 协议通信模型**



实验采用典型的客户 / 服务器（C/S）模型进行 TCP 协议通信，其通信流程如下：

1. **服务器端**
   * 通过socket()函数创建套接字，指定网络地址类型为AF\_INET，通信协议类型为SOCK\_STREAM，网络通信协议为IPPROTO\_IP，成功创建后返回套接字标识符sockid。
   * 使用bind()函数将服务器进程标识（IP 地址和周知端口号）与sockid绑定，以便服务器在指定地址和端口上监听连接请求。
   * 调用listen()函数，设置连接请求队列长度，使服务器进入监听状态，等待客户端连接请求。
   * 当客户端连接请求到达时，accept()函数从连接请求队列中取出队首请求，构建新的套接字new\_sockid，用于与该客户端进行通信，并返回给服务器进程。
   * 通过recv()函数接收客户端发送的数据，使用send()函数向客户端发送数据，实现数据的双向传输。
   * 通信结束后，服务器端两次调用closesocket()函数，分别释放new\_socketid和old\_socketid相关资源，包括动态数据结构变量、全局变量和定时器等。
2. **客户端**
   * 同样使用socket()函数创建套接字，获取套接字标识符。
   * 调用connect()函数，主动向服务器发送连接请求，将服务器的 IP 地址和端口号等信息填充到通信五元组中，并构造 TCP 连接请求报文段发送给服务器。
   * 连接成功后，使用send()函数向服务器发送数据，通过recv()函数接收服务器的响应数据。
   * 通信完成后，客户端调用closesocket()函数释放套接字相关资源。

**（二）接口函数设计与实现**

1. **socket**()**函数**
   * 功能：构造通信五元组数据结构变量并初始化。五元组包括本地地址、本地端口、目标地址、目标端口和通信协议类型。
   * 实现细节：根据传入参数af、type和protocol，在内存 “堆” 地址空间动态分配数据结构变量。对于local\_address初始化为本地 IP 地址（客户端为随机分配的临时端口号，如 5000；服务器为指定端口号，初始化为 -1），target\_address初始化为 0.0.0.0（无法使用），target\_port初始化为 -1（无法使用），SOCK\_STREAM初始化为指定的通信协议类型（如SOCK\_STREAM）。
2. **bind**()**函数**
   * 功能：将服务器进程标识（IP 地址 + 端口号）与创建的套接字绑定。
   * 实现细节：将传入的sockid对应的通信五元组中的local\_address和local\_port分别初始化为服务器的 IP 地址和周知端口号，target\_address和target\_port保持不变。
3. **connect**()**函数**
   * 功能：客户端主动请求与服务器建立连接。
   * 实现细节：
     + 通信五元组赋值：将服务器 IP 地址和端口号填充到五元组的相应位置。
     + 构造 TCP 连接请求报文段：包括随机生成的初始序号（SEQ）、设置 SYN 标志位为 1、指定接收窗口大小、源端口和目的端口等信息，以及在选项字段设置 MSS 为 1400B。
     + 计算 TCP 首部简单校验和：对伪首部 + 首部 + 数据进行校验和计算。
     + 将 TCP 连接请求交付给网络层 IP 协议发送。
     + 接收服务器发送的连接请求 + ACK 应答，并构造 ACK 应答发送给服务器。
     + 初始化客户端全局变量，包括客户和服务器初始序号、接收窗口大小、MSS 大小（取客户 / 服务器 MSS 最小值）。
4. **listen**()**函数**
   * 功能：使服务器在指定套接字上监听连接请求，并设置连接请求队列长度。
   * 实现细节：服务器 IP 协议接收到 IP 分组后，判断是否为 TCP 连接请求，若是则将其存放在 TCP 连接请求队列中，若队列溢出则暂缓存后续连接请求。
5. **accept**()**函数**
   * 功能：服务器处理客户端的连接请求，返回新的套接字用于与该客户端通信。
   * 实现细节：
     + 从 TCP 连接请求队列中取队首请求。
     + 依据源端口号和源 IP 地址构造clientaddr对象变量。
     + 构建new\_sockid对象五元组变量，包括服务器 IP 地址、周知端口号、客户端 IP 地址、客户端口号和通信协议类型。
     + 构造 TCP 反向连接请求 + ACK 应答报文并发送，接收客户端应答后初始化全局变量。
     + 返回new\_sockid对象指针。
6. **send**()**函数**
   * 功能：在 TCP 连接上发送数据。
   * 实现细节：
     + 将发送数据存入发送缓存队列。
     + 根据发送窗口限制，将窗口内数据分装为 TCP 报文段，调用IP\_SEND(TCP - BUF)交付给网络层 IP 协议。
     + 每发送一个 TCP\_BUF，启动重发定时器，超时未收到应答则重发。
     + 根据应答序号和接收窗口大小滑动发送窗口，继续发送后续报文段。
     + 返回每次交付成功的 TCP\_BUF 大小。
7. **recv**()**函数**
   * 功能：在 TCP 连接上接收数据。
   * 实现细节：
     + 检查 TCP 首部目的端口号是否与sockid结构中一致，校验和是否正确，接收数据序号是否在接收窗口内，若有问题则采取相应策略（如丢弃）。
     + 将 TCP 报文段数据部分复制到接收缓存队列的接收窗口中。
     + 根据接收窗口状态交付 MSS 大小数据给上层应用。
     + 滑动接收窗口，继续接收后续数据并发送 ACK 应答。
     + 返回接收的有效数据大小。
8. **closesocket**()**函数**
   * 功能：释放套接字相关资源。
   * 实现细节：释放socketid对象的动态数据结构变量、全局变量和定时器。服务器端需两次调用，分别释放new\_socketid和old\_socketid相关资源；客户端调用一次释放自身套接字相关资源。

**（三）数据结构设计**

1. **通信五元组数据结构**TCP\_FIVE - PARA
   * 成员变量：local\_address（字符指针，存储本地 IP 地址）、local\_port（整数，存储本地端口号）、target\_address（字符指针，存储目标 IP 地址）、target\_port（整数，存储目标端口号）、SOCK\_STREAM（整数，存储通信协议类型）。
2. **发送缓存队列数据结构**（循环队列基础上增加指针成员）
   * 成员变量：循环队列用于存储待发送数据，以及窗口后延指针、发送指针和窗口前沿指针用于管理发送窗口。
3. **接收缓存队列数据结构**（循环队列基础上增加指针成员）
   * 成员变量：循环队列用于存储接收数据，以及窗口后延指针、接收指针和窗口前沿指针用于管理接收窗口。

**（四）滑动窗口技术原理**

滑动窗口是 TCP 协议实现高效可靠数据传输的关键机制。其主要目的是在保证数据可靠传输的前提下，充分利用网络带宽，提高传输效率。

*发送窗口*

1. **窗口构成**
   * 发送窗口由三个指针界定范围：窗口后延指针、发送指针和窗口前沿指针。窗口后延指针指向已发送但未收到确认的数据段的起始位置；发送指针指向当前可发送数据的位置；窗口前沿指针指向发送窗口的结束位置。
   * 例如，假设发送窗口大小为 5，初始时窗口后延指针、发送指针和窗口前沿指针都指向第一个待发送数据段。随着数据的发送，发送指针向前移动；当收到部分数据的确认时，窗口后延指针根据确认序号移动。
2. **数据发送与窗口滑动**
   * 当发送窗口未满时，发送方可以持续发送数据段。发送方根据接收方返回的接收窗口大小来确定可发送的数据量，避免发送过多数据导致接收方缓冲区溢出。
   * 每发送一个数据段，发送指针向前移动一个位置。同时，启动重发定时器，若在定时器超时前收到了对该数据段的确认，发送方根据确认序号移动窗口后延指针，从而实现发送窗口的滑动，继续发送后续报文段；若定时器超时未收到应答，则重发该数据段，确保数据的可靠传输。

*接收窗口*

1. **窗口作用**
   * 接收窗口用于控制接收数据的范围，确保接收方有足够的缓冲区来处理接收到的数据。接收方根据自身的缓冲区大小和已接收数据的情况，在确认应答中告知发送方接收窗口的大小。
2. **数据接收与窗口调整**
   * 接收方检查接收数据序号是否在接收窗口内。如果在窗口内，则将数据复制到接收缓存队列的接收窗口中，并根据接收窗口状态交付 MSS（最大段大小）大小数据给上层应用。
   * 接收方根据接收情况调整接收窗口指针，滑动接收窗口，继续接收后续数据。例如，当接收方成功接收并处理了部分数据后，接收窗口向前滑动，接收指针移动到新的位置，准备接收下一批数据。

*在实验中的应用*

（一）发送窗口相关函数与操作

1. **send()函数中的滑动窗口处理**
   * 在实验的send()函数实现中，首先将发送数据存入发送缓存队列。然后，根据发送窗口限制，将窗口内数据分装为 TCP 报文段，调用IP\_SEND(TCP - BUF)交付给网络层 IP 协议。
   * 以下是一个代码片段，展示了发送窗口相关操作：

**def** send(sockid, buf, buflen, flags):  
 *# 将数据存入发送缓存队列*  
 send\_buffer.put(buf)  
 *# 获取发送窗口大小*  
 window\_size = get\_send\_window\_size()  
 **while** len(send\_buffer) > 0 **and** window\_size > 0:  
 *# 从发送缓存队列中取出窗口内数据分装为TCP报文段*  
 tcp\_segment = create\_tcp\_segment(send\_buffer.get(), window\_size)  
 *# 发送TCP报文段并启动重发定时器*  
 IP\_SEND(tcp\_segment)  
 start\_retransmission\_timer(tcp\_segment)  
 *# 更新发送指针和窗口大小*  
 update\_send\_pointer()  
 window\_size = get\_send\_window\_size()  
 **return** len(buf)

* 在上述代码中，send\_buffer是发送缓存队列，get\_send\_window\_size()函数用于获取当前发送窗口大小，create\_tcp\_segment()函数用于根据窗口大小封装 TCP 报文段，start\_retransmission\_timer()函数启动重发定时器，update\_send\_pointer()函数更新发送指针位置。

1. **重发定时器与窗口滑动**

* 当重发定时器超时时，需要检查该 TCP 报文段对应的应答是否收到。如果未收到，则重发该 TCP 报文段。重发操作会影响发送窗口的状态，因为重发的数据段可能会占用窗口空间，直到收到确认。
* 以下是重发定时器超时处理的代码（retransmission\_timer为定时器对象，tcp\_segments`为已发送但未确认的数据段列表）：

**def** on\_retransmission\_timer\_timeout():  
 **for** tcp\_segment **in** tcp\_segments:  
 **if** **not** is\_ack\_received(tcp\_segment):  
 *# 重发TCP报文段*  
 IP\_SEND(tcp\_segment)  
 *# 更新重发定时器*  
 retransmission\_timer.reset()

* 当收到确认应答时，需要根据确认序号移动窗口后延指针，实现发送窗口的滑动。代码如下：

**def** process\_ack(ack\_number):  
 **for** i, tcp\_segment **in** enumerate(tcp\_segments):  
 **if** tcp\_segment.seq\_number == ack\_number:  
 *# 移动窗口后延指针*  
 move\_send\_window\_backward(i)  
 **break**

（二）接收窗口相关函数与操作

1. **recv()函数中的滑动窗口处理**
   * 在recv()函数中，接收方首先检查 TCP 首部目的端口号是否与sockid结构中一致，校验和是否正确，接收数据序号是否在接收窗口内。若有问题则采取相应策略（如丢弃）。
   * 以下是recv()函数中接收窗口处理的示例代码：

**def** recv(sockid, buf, buflen, flags):  
 *# 接收TCP报文段*  
 tcp\_segment = receive\_tcp\_segment()  
 **if** tcp\_segment **and** is\_valid\_tcp\_segment(tcp\_segment):  
 *# 检查数据序号是否在接收窗口内*  
 **if** is\_in\_receive\_window(tcp\_segment.seq\_number):  
 *# 将数据复制到接收缓存队列的接收窗口中*  
 copy\_to\_receive\_buffer(tcp\_segment.data)  
 *# 根据接收窗口状态交付数据给上层应用*  
 deliver\_to\_application(buf, buflen)  
 *# 发送ACK应答并滑动接收窗口*  
 send\_ack(tcp\_segment.seq\_number)  
 move\_receive\_window\_forward()  
 **return** len(tcp\_segment.data)  
 **return** 0

* 在上述代码中，receive\_tcp\_segment()函数用于接收 TCP 报文段，is\_valid\_tcp\_segment()函数检查报文段的有效性，is\_in\_receive\_window()函数判断数据序号是否在接收窗口内，copy\_to\_receive\_buffer()函数将数据复制到接收缓存队列，deliver\_to\_application()函数将数据交付给上层应用，send\_ack()函数发送 ACK 应答，move\_receive\_window\_forward()函数滑动接收窗口。

1. **接收窗口指针调整**

* 当接收方成功处理了接收窗口内的数据后，需要调整接收窗口指针。例如，当接收方将接收缓存队列中的数据交付给上层应用后，接收指针向前移动，接收窗口向前滑动，准备接收下一批数据。
* 以下是接收窗口指针调整的代码：

**def** move\_receive\_window\_forward():
  
 *# 更新接收指针位置*  
 update\_receive\_pointer()
  
 *# 根据接收指针位置调整接收窗口范围*  
 adjust\_receive\_window()

**（五）连接建立三握手和连接释放四握手分析**

1. **连接建立三握手**
   * **第一次握手（客户端 -> 服务器）**：客户端调用connect()函数，构造 TCP 连接请求报文段。此时，客户端随机生成初始序号（SEQ），设置 SYN 标志位为 1，表示请求建立连接，同时指定接收窗口大小、源端口（随机分配的临时端口号，如 5000）和目的端口（服务器的周知端口号）等信息，并在选项字段设置 MSS 为 1400B。然后将该报文段发送给服务器。这一步骤在connect()函数中通过构造特定格式的 TCP 报文段并调用IP\_SEND(TCP\_BUFFER)实现。
   * **第二次握手（服务器 -> 客户端）**：服务器接收到客户端的连接请求报文段后，若同意建立连接，则构造 TCP 连接请求 + ACK 应答报文段。服务器为本次连接分配资源，初始化相关变量，并生成自己的初始序号（SEQ），设置 SYN 和 ACK 标志位为 1，确认序号（ACK number）为客户端的初始序号加 1，表示收到客户端的连接请求。然后将该报文段发送给客户端。在服务器端，这一过程在accept()函数中完成，包括构建应答报文段并发送。
   * **第三次握手（客户端 -> 服务器）**：客户端收到服务器的连接请求 + ACK 应答报文段后，构造 ACK 应答报文段。客户端将确认序号设置为服务器的初始序号加 1，表示确认收到服务器的连接请求，同时将 ACK 标志位设置为 1。然后将该报文段发送给服务器，完成连接建立的三次握手过程。之后，客户端初始化全局变量，包括客户和服务器初始序号、接收窗口大小、MSS 大小（取客户 / 服务器 MSS 最小值）。在客户端代码中，这一系列操作在connect()函数中对服务器应答的处理部分实现。
   * 三次握手的意义在于确保双方都具备收发数据的能力，并且同步了初始序号，为后续可靠的数据传输奠定基础。通过这种方式，双方可以在可靠的基础上开始数据传输，避免了因网络延迟、重复报文段等问题导致的错误连接建立。
2. **连接释放四握手**
   * **第一次握手（主动关闭方 -> 被动关闭方）**：假设客户端主动发起连接关闭，客户端调用closesocket()函数，此时客户端发送一个 FIN 报文段，其中 FIN 标志位被设置为 1，表示客户端不再发送数据，但仍可接收数据。
   * **第二次握手（被动关闭方 -> 主动关闭方）**：服务器收到客户端的 FIN 报文段后，发送一个 ACK 报文段作为应答，确认序号为客户端的 FIN 报文段序号加 1，表示服务器已经收到客户端的关闭请求。此时，服务器进入 CLOSE\_WAIT 状态，继续等待可能剩余的数据传输。
   * **第三次握手（被动关闭方 -> 主动关闭方）**：服务器完成剩余数据的发送后，发送一个 FIN 报文段给客户端，其中 FIN 标志位为 1，表示服务器也准备关闭连接。
   * **第四次握手（主动关闭方 -> 被动关闭方）**：客户端收到服务器的 FIN 报文段后，发送一个 ACK 报文段作为应答，确认序号为服务器的 FIN 报文段序号加 1。此时，客户端进入 TIME\_WAIT 状态，等待一段时间（通常为 2 倍的最大段生存时间，即 2MSL），确保服务器收到 ACK 报文段，然后客户端释放连接资源。服务器收到客户端的 ACK 报文段后，也释放连接资源，连接正式关闭。
   * 连接释放四握手的过程确保了在关闭连接时，双方都能够有序地停止数据传输，并且保证了在网络中可能存在的延迟或丢失的报文段都能够被正确处理，避免数据丢失或连接异常关闭的情况。

**四、运行结果与分析**

**（一）运行结果**

1. 服务器端启动后，成功绑定指定 IP 地址和周知端口号，进入监听状态，等待客户端连接。
2. 客户端启动后，通过connect()函数成功与服务器建立连接，双方可以进行数据传输。
3. 在数据发送过程中，服务器端和客户端能够正确调用send()函数发送数据，并通过recv()函数接收对方发送的数据，数据传输稳定可靠。
4. 通信结束后，服务器端和客户端分别调用closesocket()函数释放资源，程序正常退出。

**（二）结果分析**

1. **连接建立过程**
   * 客户端发送的连接请求报文段能够正确到达服务器，服务器响应的连接请求 + ACK 应答也能被客户端正确接收，这表明connect()和accept()函数的实现正确，能够按照 TCP 协议规范完成连接建立的三次握手过程。
   * 成功建立连接后，双方通信五元组中的相关信息被正确初始化，为后续数据传输奠定了基础。
2. **数据传输过程**
   * send()函数能够根据发送窗口大小合理地将数据分装为 TCP 报文段发送，并且在重发定时器的作用下，保证了数据的可靠传输。例如，当网络出现短暂拥塞导致部分报文段丢失时，重发机制能够及时重发丢失的报文段，确保数据的完整性。
   * recv()函数对接收数据的处理也符合预期，能够正确检查 TCP 首部信息，将有效数据存入接收缓存队列并交付给上层应用，同时通过滑动接收窗口和发送 ACK 应答，保证了数据的有序接收和流量控制。
3. **资源释放过程**
   * 服务器端和客户端在通信结束后调用closesocket()函数，能够正确释放相关资源，包括动态分配的数据结构变量、全局变量和定时器等，避免了资源泄漏问题，保证了程序的稳定性和可扩展性。

**五、实验过程发现的新技术问题及优化方向**

**（一）发现的问题**

1. 在处理大量并发连接时，服务器的性能表现有待提高。随着连接数的增加，服务器响应时间变长，可能出现连接请求处理不及时的情况。
2. 对于网络异常情况（如网络中断、长时间延迟等）的处理不够完善。在实验过程中，当网络出现异常中断后，客户端和服务器端的连接恢复机制不够智能，需要手动重新启动程序才能恢复通信。
3. 发送和接收缓存队列的大小设置对性能有一定影响。如果设置过小，可能导致数据积压，影响传输效率；如果设置过大，会占用过多内存资源。

**（二）优化方向**

1. 优化服务器端的并发处理机制，采用多线程或异步 I/O 等技术提高服务器的并发处理能力。例如，可以为每个连接创建一个独立的线程来处理数据传输，或者使用异步 I/O 操作避免线程阻塞，提高服务器的响应速度和吞吐量。
2. 增强网络异常处理能力，实现自动重连和数据恢复机制。当检测到网络异常时，客户端和服务器端可以尝试自动重新建立连接，并根据之前的传输状态恢复数据传输，减少人工干预。
3. 根据实际应用场景和网络环境，合理调整发送和接收缓存队列的大小。可以通过性能测试和分析，找到一个最优的缓存大小配置，以平衡传输效率和内存资源占用。同时，可以考虑采用动态调整缓存大小的策略，根据网络状况和数据流量实时优化缓存性能。

**六、参考文献**

[1] 《TCP/IP 详解卷 1：协议》，W. Richard Stevens 著，机械工业出版社。  
[2] 《计算机网络：自顶向下方法》，James Kurose、Keith Ross 著，机械工业出版社。  
[3] 第5次课程-试点班-2