基于 Socket 的 Windows 与 Linux 平台异步通信①

徐克宝, 武 慧, 文艺成

(山东科技大学 机械电子工程学院, 青岛 266590)

摘 要: 分析了 I/O 复用模型的原理和特点, 并针对 Windows 与 Linux 平台间通信的并发数量不足的问题, 提出在 Windows 与 Linux 平台间使用 socket(套接字)异步通信的方法. 通信采用客户端/服务器(Client/Server)模型. 在该模型中, 服务器端使用 Linux 提供的 epoll 接口, 而客户端使用.NET 提供的 AsyncCallBack(异步回调)的委托方式构建跨平台的异步通信. 此外, 结合多线程编程方法, 处理 socket 通信客户端的显示问题. 从观测通信过程来看, 服务器端能够同时接收多个客户端发送的数据并返回数据. 实验结果表明, 在 Windows 与 Linux 进行 socket 异步通信是完全可行的, epoll 接口的使用提高了 Linux 服务器端的并发性, 从而提高了 Windows 和 Linux 平台间网络通信的实时性.

关键词: 套接字; 跨平台; 客户端/服务器; 异步通信; 多线程

Asynchronous Communication Between Windows and Linux Platforms Based on Socket

XU Ke-Bao, WU Hui, WEN Yi-Cheng

(College of Mechanical and Electronic Engineering, Shandong University of Science and Technology, Qingdao 266590, China)

Abstract: This article analyses the principle and characteristics of I/O multiplexing models. It uses the socket's asynchronous communication to solve the problem of insufficient number of concurrent when Windows platform communicates with Linux platform. And the whole system adopts Client/Server model to build cross-platform asynchronous communication program. The server uses the epoll model provided by Linux and the client uses the AsyncCallBack provided by .NET to realize the cross-platform asynchronous communication. In addition, the program combines with the multithreading programming method to deal with the display problem of the client. From the observation of communication process, the server can receive data sent by multiple clients in the meantime and return data to them. In conclusion, with the using of the epoll interface, Linux server's concurrency is improved. For this reason, the real-time performance of the network communication between Windows and Linux platforms is also improved.

Key words: socket; cross-platform; client/server; asynchronous communication; multithreading

嵌入式系统已经成为计算机领域的一个重要组成部分[1]. 工业生产中, 人们常把稳定性高的 Linux 系统应用到工业控制领域, 以此来提高控制系统的性能[2]. 而工程应用软件大多数在 Windows 平台下进行开发. 因此, Linux 与 Windows 平台间的自由通信也越来越受到人们的重视. 当前, socket 通信又主要局限在同一平台下, 而对于 Windows 与 Linux 跨平台间的通信也主要局限在同步通信、单线程等方式, 显然这些方式已不能满足实际通信需求.

因此, 在 Windows 与 Linux 平台之间实现异步通信以及多线程编程方式显得十分重要. 本文的 Windows 与 Linux 之间的网络通信采用了基于面向连接的协议——TCP 协议, 并通过 socket 异步通信方式进行数据收发. 在 Windows 平台使用 C#编写的客户端, 该客户端采用异步回调方式和多线程编程方式进行设计. 而在 Linux 则使用 C 语言编写的服务器端, 该服务器端通过使用 epoll 接口实现异步操作. 本文简要分析 I/O 复用模型, 分别介绍了 Linux 和 Windows 的异步通

① 收稿时间:2014-10-27;收到修改稿时间:2014-12-17

信过程, 并举例说明 Windows 和 Linux 平台间的异步通信的软件设计过程、调试步骤和结果.

1 Socket异步通信

1.1 I/O 复用模型

Socket 为套接字, 是一种双向的通信端口[3]. I/O 复用模型可使用 select、poll、epoll 函数, 其中 epoll 是 Linux 下多路复用 I/O 接口 select/poll 的增强版本, epoll 中所监听的事件数没有限制, 它只与系统的资源有关[4].

如图 1 所示, 以 select 函数为例对 I/O 复用模型进行分析. 应用进程要执行读操作, 进行了系统调用, 系统调用被传递给了内核. 但此时在应用进程在系统调用之后, 并不等待内核返回响应信息(这也是其被称为异步 I/O 的重要原因). 而 I/O 复用模型之所以被称为阻塞, 主要是因为 select()函数会阻塞应用进程, 一直等到这个系统调用有结果返回, 再通知应用进程. 即在 I/O 复用模型中, 配置为非阻塞 I/O, 然后使用阻塞 select 系统调用来确定 I/O 描述符是否可读写.

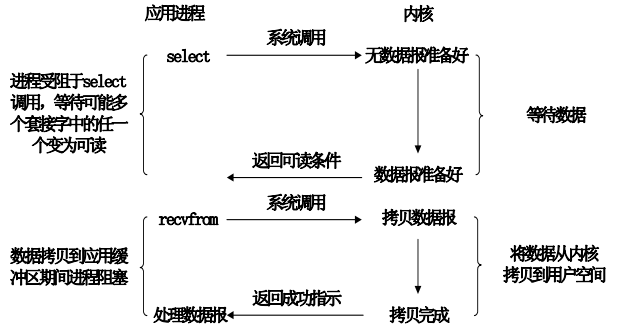

图 1 I/O 复用模型

1.2 Windows 与 Linux 平台间的异步通信过程

Linux 平台作为服务器端, Windows 平台作为客户端, 两个平台间的异步通信过程如图 2 所示.

服务器端首先创建套接字地址允许来自任何接口的连接, 然后将其绑定到服务器套接字上, 接下来就可以监听客户端的连接请求. 此时, 若有多个客户端与服务器端进行连接, 服务器端监听到有新的连接, 则接受客户端连接请求, 并将所有新的文件描述符添加至 epoll 的监听队列中.

当接收到数据时, 读取 socket 上的数据, 并修改标识符, 等待下一个循环时发送数据, 实现服务器端的异步操作. 客户端发送完消息后, 调用回调函数, 结束发送.

在服务器端接收数据的同时, 如果此时在键盘输入信息并敲击“Enter”键发送, 则对键入信息进行处理, 并将信息加入套接字发送队列. 客户端接收完消息后, 调用回调函数, 结束接收.

使用 epoll 后, 需要用 close()来关闭被创建的 epoll 句柄, 而套接字可以通过人为方式进行关闭. 如图 2 中 quit 和关闭连接的判断, 一旦条件符合则关闭套接字, 断开连接.

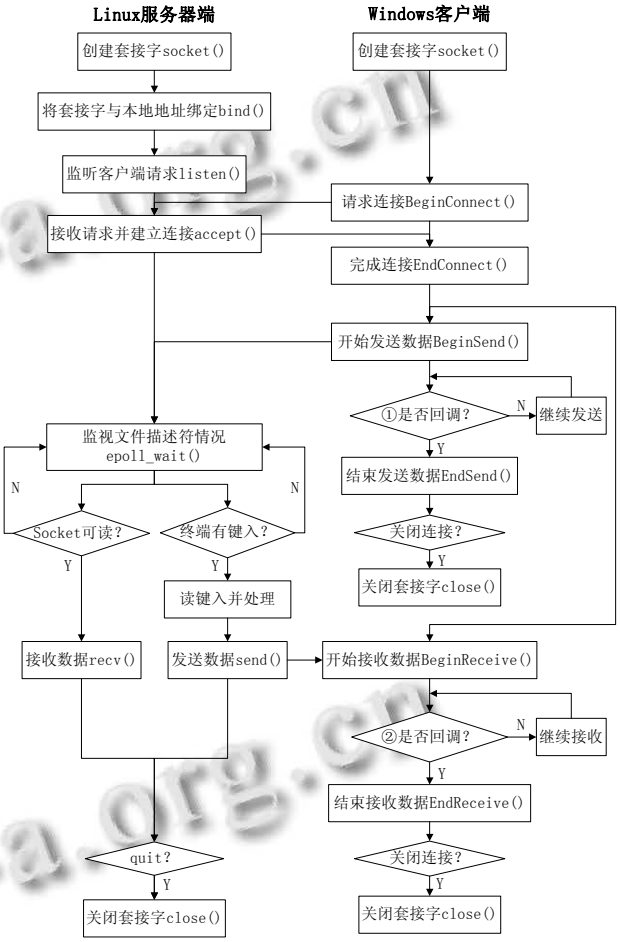

图 2 Windows 与 Linux 异步通信过程

① 如图 2 所示, 流程图的标号①中, 客户端发送数据异步回调情况如下:

客户端发送数据长度与服务器端接收 buffer 长度一致, 发送所有数据时回调.

客户端发送数据长度大于服务器端接收 buffer 长度, buffer 填满时回调, 数据发送完时回调.

客户端发送数据长度小于服务器端接收 buffer 长度, 数据发送完时回调.

② 如图 2 所示, 流程图的标号②中, 客户端接收数据异步回调情况如下:

服务器端发送数据长度与客户端接收 buffer 长度一致, 接收到所有数据时回调.

服务器端发送数据长度大于客户端接收 buffer 长度, buffer 填满时回调, 数据接收完时回调.

服务器端发送数据长度小于客户端接收 buffer 长度, 数据接收完时回调.

2 软件设计

2.1 Linux 平台软件设计

2.1.1 响应客户端连接请求

使用函数 socket、bind、listen、accept [5]监听客户端的连接请求并与之进行连接. 连接时, 服务器端响应客户端连接请求, 建立一个新的线程, 服务器端的套接字继续处于监听状态, 继续接受其他客户端套接字请求.

2.1.2epoll 接口的使用步骤

① 需要包含头文件#include <sys/epoll.h>.

② 调用 create_epoll 函数来创建一个 epoll 句柄.

③ 调用 epoll_ctl 函数将新的 socket 句柄加入到 epoll 的监听队列中.

④ 调用 epoll_wait 函数来监听所有 socket 句柄. 在给定的 timeout 时间内, 当监听的所有 socket 句柄中有事件发生时, 返回用户态进程, 执行数据发送与接收函数.

2.1.3 接收和发送数据

接收数据时, 使用 recv(socket, buffer, BUFFER_SIZE, 0) [6]将数据接收到数据缓冲区. socket 为客户端套接字描述符、buffer 和 BUFFER_SIZE 分别表示发送数据内容和长度.

发送数据时, 使用 fgets(buffer, BUFFER_SIZE, stdin)函数, stdin 表示从屏幕上输入字符串. 再使用 send(socket, buffer, strlen(buffer) - 1, 0)将屏幕上的数据发送到数据缓冲区.

2.1.4 终止监听

如果要终止监听, 使用 if (!strncasecmp(buffer, "quit", 4))判断输入内容是否为 quit, 如果是则使用 printf("服务器终止监听!\n")语句, 在屏幕上输出“服务器终止监听！”提示信息. 最后执行 break 跳出当前循环, 终止监听.

2.2 Windows 平台软件设计

2.2.1 界面设计

在 Windows 平台下, 使用 C#语言编写的 Socket 异步通信程序. 在 Microsoft Visual Studio 2010 中新建“Windows 窗体应用程序”, 在窗口添加 4 个 button 控件, 分别为连接服务器(tb_Conn)、关闭连接(tb_CloseConn)、清空(tb_Clear)、发送(tb_Send); 添加 1 个 RichTextBox 控件 rtb_Status(消息显示框); 添加 3 个 TextBox 控件, 分别为 tb_Port(端口号输入框)、tb_IP(IP 地址输入框)、tb_SendInfo(发送内容输入框); 最后添加3个Lable控件, 分别为端口号、IP 地址、发送内容.

2.2.2 请求连接和结束请求

传递 IP 地址和端口号创建 IPEndPoint 对象 remoteep, 创建异步回调 AsyncCallBack 对象 callback 以及套接字 Socket 对象 client, 并使用 client.BeginConnect(remoteep, callback, client)进行连接. 当服务器端接受连接请求事件发生后, 在异步回调函数中使用 client.EndConnect(ar)(其中 ar 为 IAsyncResult 对象)完成远程主机的异步连接请求.

2.2.3 发送和接收数据

① 发送数据时, 读取发送内容输入框中的数据, 首先创建字节数组:

```
byte[]msg=Encoding.UTF8.GetBytes(tb_SendInfo.Text);
```

并向 Socket 异步发送数据:

```
client.BeginSend(msg,0,msg.Length,SocketFlags.None,
callback, client)
```

此后判断是否进行异步回调, 执行异步发送回调函数.

② 接收数据时, 创建字节数组:

```
Rcvbuffer = new byte[client.SendBufferSize];
```

并从 Socket 中异步接收数据:

```
client.BeginReceive(Rcvbuffer,0,Rcvbuffer.Length,
SocketFlags.None, callback, client).
```

此后判断是否进行异步回调, 执行异步接收回调函数.

2.2.4 多线程方式进行消息显示

多线程程序作为一种多任务、并发的工作方式, 还可以提高应用程序响应时间[7]. 采用多线程方式进行显示, 将发送和接收信息、状态信息等显示在消息显示框 rtb_Status 中. 具体使用步骤如下:

① 调用委托类型和委托方法

```
delegate void AppendDelegate(string str);
AppendDelegate AppendString;
```

② 在消息显示框 rtb_Status 写字符串的委托方法

```
private void AppendMethod(string str){
rtb_Status.AppendText(str);}
```

③ 实例化委托对象, 与委托方法关联

```
AppendString=new AppendDelegate(AppendMethod);
```

④ 需要显示信息时, 使用下列语句将消息显示

在消息显示框 rtb_Status 中

rtb_Status.Invoke(AppendString, String.Format("消息显示内容！")).

3 调试步骤与结果分析

3.1 编译并执行服务器端程序

① 编译: gcc -o SocketServer SocketServer.c -Wall.

② 执行: ./ SocketServer.

3.2 运行客户端软件

在客户端中的IP地址输入框和端口输入框分别输入服务器端监听的 IP 地址和端口号, 并单击连接服务器. 至此, Windows 平台的客户端软件与 Linux 平台运行的程序就可以进行异步通信了.

3.3 运行结果分析

图 3 所示是异步通信 Windows 客户端, 在窗体界面的发送内容输入框中输入消息, 并点击“发送”. 图 4 所示是异步通信 Linux 服务器端, 在终端输入要发送的消息, 敲击键盘“Enter”键进行发送.

图 3 异步通信 Windows 客户端运行效果图

图 4 异步通信 Linux 服务器端运行效果图

如图 5 所示, 为了模拟服务器的高并发性, 还模拟了 10000 个以上的客户端同时连接 Linux 服务器端并向服务器端发送数据, 服务器端接收到客户端的数据后返回数据. 在只接收 Windows 客户端简短信息而未做其它通信的条件下, Linux 服务器总的处理时间不到 1s. Linux 服务器的高并发性与 epoll 接口红黑树、双链表数据结构、回调机制是密不可分的.

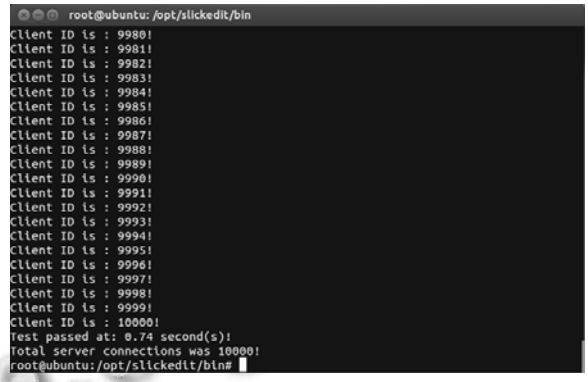

图 5 异步通信 Linux 服务器端高并发测试图

4 结语

经过研究与设计, Windows平台和Linux平台能够进行异步通信. 异步通信的最大优点是可以在一个操作没有完成之前同时进行其它的操作. socket为我们提供了发送和接收网络数据的接口, 是跨平台异步通信的前提. Windows 平台使用异步回调的委托方式, 而 Linux 平台则使用 epoll 接口. 异步回调和 epoll 接口使 Windows 和 Linux 平台实现全双工异步通信, 提高了跨平台通信的实时性和信息共享能力, 为 Linux 服务器处理数量巨大的 Windows 客户端通信请求提供了解决方案.

参考文献

1 孙旭,孙瑜,孙道层.基于 TCP/IP 协议的网络通信模式研究.商洛学院学报,2012,26(4):24–27.

2 周西峰,陆鹏,郭前岗.利用 Socket 实现 Windows 与 Linux 平台间的网络通信.微型机与应用,2013(18):49–51.

3 欧军,吴清秀,裴云,等.基于 socket 的网络通信技术研究.网络安全技术与应用,2011(7):19–21.

4 梁明刚,陈西曲.Linux 下基于epoll+线程池高并发服务器实现研究.武汉工业学院学报,2012,31(3):54–59.

5 张杰敏,王巍.基于 UNIX/Linux 的 C 系统编程.北京:清华大学出版社,2013.

6 Jones MT.GUN/Linux 环境编程(第 2 版).张元章,译.北京:清华大学出版社,2010.

7 戴景峰,潘松峰,薛兵.基于 Linux 的嵌入式数据采集装置的 SOCKET 通信.信息技术与信息化,2011(2):65–67.