

《计算机网络与通信》课程

实验三 Socket编程

2023年秋季学期

实验介绍

◆实验目的

- ✓ 理解TCP和UDP协议主要特点
- ✓ 掌握socket的基本概念和工作原理，编程实现socket通信

◆实验任务

- ✓ 任务1：完善socket客户机
- ✓ 任务2：课程表查询服务器（TCP迭代）
- ✓ 任务3：课程表查询服务器（TCP并发）
- ✓ 任务4：获取网页内容（Python自学）

运输层协议

◆运输层为应用层提供两种服务

- ✓ TCP协议：面向连接，可靠
- ✓ UDP协议：无连接，不可靠
- ✓ 使用<IP地址，端口号>标识应用程序的服务接入点

◆运输层提供端到端的通信能力

- ✓ 为通信双方提供一条逻辑通信管道——socket编程对象
- ✓ 使用[协议] + [双方<IP地址，端口号>]来标识管道

什么是socket

◆socket是个多义词

- ✓ 通信双方的逻辑通信管道
- ✓ 围绕这种管道设计的一套应用编程接口(API)—— 位于应用层和运输层之间，包含一系列相关函数和数据结构
- ✓ 用于标识管道的数据结构 —— 通常以该结构的描述符/句柄形式出现
- ✓ Unix/Linux系统中用于创建管道的函数——socket()

常见socket的类型

◆按协议类型划分：

***流式socket (SOCK_STREAM)：** 基于TCP协议。

面向连接，可靠

***数据报socket (SOCK_DGRAM)：** 基于UDP协议。

无连接，不可靠

***原始socket (SOCK_RAW)：** 基于IP协议。

无连接，不可靠，用于直接访问IP层协议

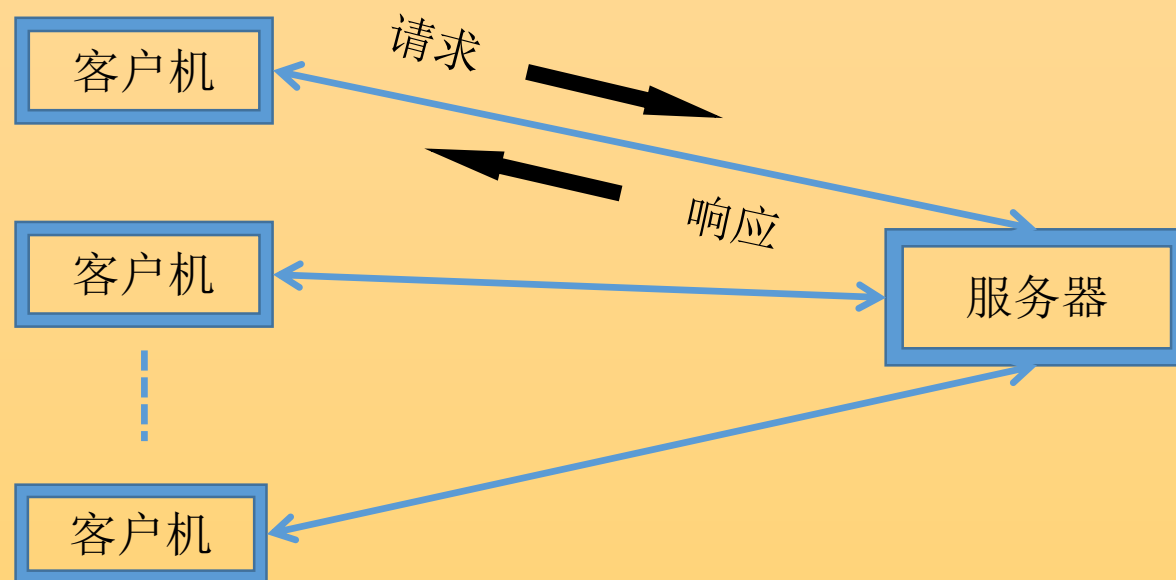
◆按I/O性能划分：

***阻塞式socket：** 函数调用将阻塞至有结果返回

***非阻塞式socket：** 函数调用立即返回，结果以消息/事件或异步回调方式告知应用程序

网络通信模型

- ◆网络通信最常见的模型，是客户机/服务器“Client/Server”模型，简称C/S模型(vs B/S)
- ◆发起通信请求的一方称为客户机，接受并处理通信请求的一方称为服务器



服务器类型

◆迭代服务器(iterative server)

- *同一时刻只能服务一个客户机。

在此过程中，下一个客户机处于等待状态

- *消耗资源少，效率低，适用于处理耗时较短的服务

- *实现简单：单进程循环

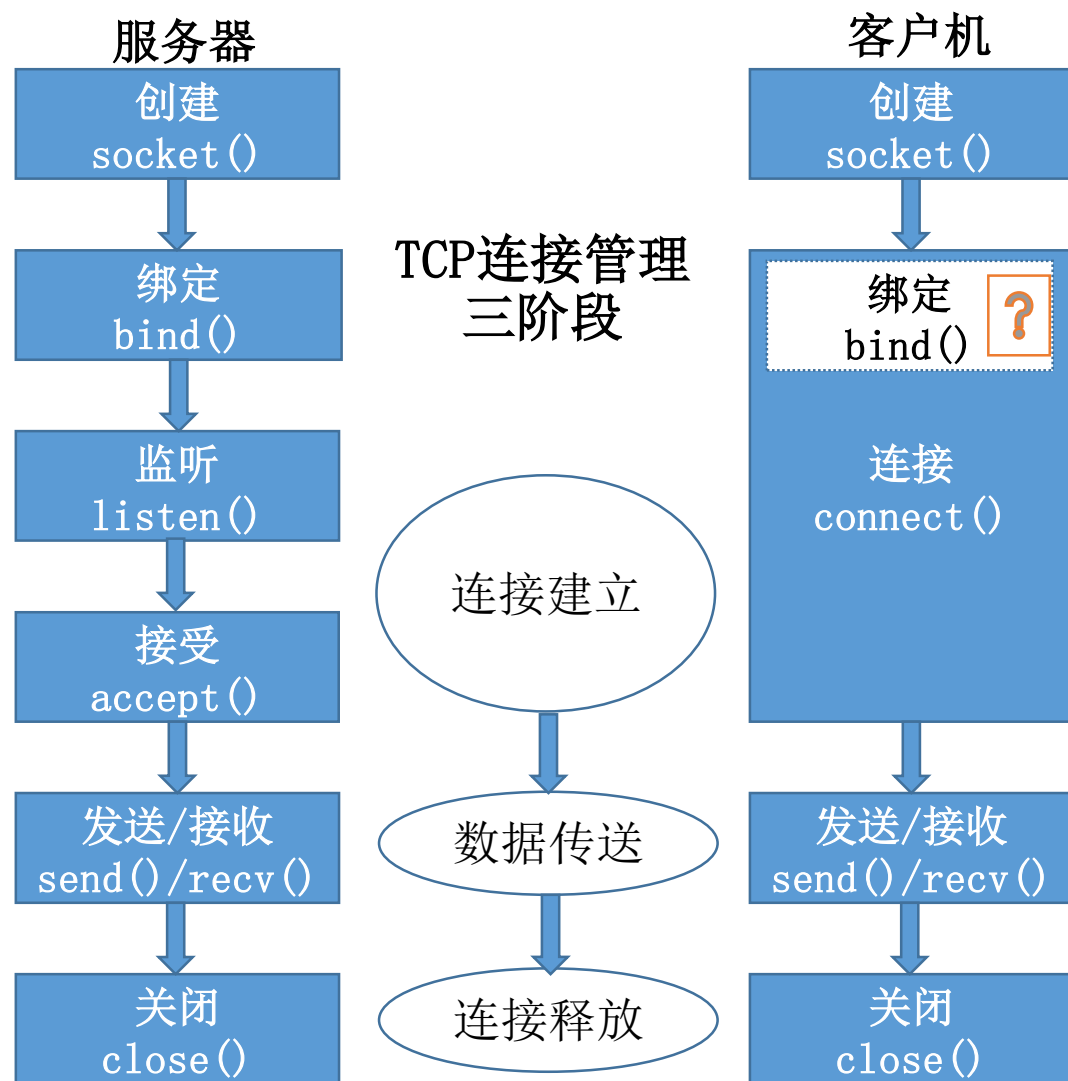
◆并发服务器(concurrent server)

- *同一时刻可以服务多个客户机

- *消耗资源多，效率高，适用于对服务响应时间要求较高的场合

- *实现复杂：多进程/线程、轮询选择、消息通知、事件响应、
I/O复用、异步I/O.....

流式socket的C/S通信模型



client_example.c

- `int main(int argc, char *argv[])`
- `//创建socket`
`if((client_sock = socket(AF_INET, SOCK_STREAM, 0)) == -1)`
`{ perror(“Client Socket Error”); //错误提示`
`exit(1);} //出错后，退出整个程序`
- `//指定服务器地址`
`server_addr.sin_port = htons(12345);`
`server_addr.sin_addr.s_addr = inet_addr(“127.0.0.1”); //指向本机`
`memset(server_addr.sin_zero, 0, sizeof(server_addr.sin_zero)); //零填充`
- `//连接服务器`
`connect(client_sock, (struct sockaddr *)&server_addr, sizeof(server_addr));`
- `//发送消息`
`printf(“Send: %s”, send_msg);`
`send(client_sock, send_msg, strlen(send_msg), 0);`
- `//接收并显示消息`
`memset(recv_msg, 0, sizeof(recv_msg)); //接收数组置零`
`recv(client_sock, recv_msg, sizeof(recv_msg), 0);`
`printf(“Recv: %s”, recv_msg);`
- `//关闭socket`
`close(client_sock);`

server_example.c

- `server_addr.sin_port = htons(12345);`
- `server_addr.sin_addr.s_addr = htonl(INADDR_ANY);` //INADDR_ANY表示本机所有IP地址
- `memset(&server_addr.sin_zero, 0, sizeof(server_addr.sin_zero));` //零填充
- //绑定socket与地址
- `bind(server_sock_listen, (struct sockaddr *)&server_addr, sizeof(server_addr));` //监听socket
- `listen(server_sock_listen, 0);`
- `server_sock_data = accept(server_sock_listen, NULL, NULL);`
- //接收并显示消息
- `memset(recv_msg, 0, sizeof(recv_msg));` //接收数组置零
- `recv(server_sock_data, recv_msg, sizeof(recv_msg), 0);`
- `printf("Recv: %s", recv_msg);`
- //发送消息
- `printf("Send: %s", recv_msg);`
- `send(server_sock_data, recv_msg, strlen(recv_msg), 0);`
- //关闭数据socket
- `close(server_sock_data);` //关闭监听socket
- `close(server_sock_listen);`

准备工作(实验环境: Ubuntu + GCC)

- 1.使用终端命令行安装本地echo服务, 监听tcp 7号端口:

`sudo apt update`

`sudo apt install xinetd`

`sudo nano /etc/xinetd.d/echo` #修改disable=no, Ctrl+O保存、Ctrl+X退出

`sudo service xinetd reload`

`netstat -an | grep :7`

```
[root@localhost ~]# netstat -an | grep :7
tcp6      0      0 :::7          :::*           LISTEN
```

- 2.输入`telnet localhost 7`, 连接本机echo服务器, 输入任意文本, 观察响应;

输入`Ctrl+]`结束echo服务, 输入`close`或`quit`退出telnet

```
cn@lab1:~$ telnet localhost 7
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello,network
hello,network
^]
telnet> quit
Connection closed.
cn@lab1:~$
```

任务1：完善socket客户机

- 1. 开启两个终端窗口，分别编译、运行server_example.c和client_example.c，观察它们实现的功能：

```
[root@localhost 桌面]# gcc server_example.c -o serv0
[root@localhost 桌面]# ./serv0

[root@localhost 桌面]# gcc client_example.c -o clnt0
[root@localhost 桌面]# ./clnt0
```

- 2. 按以下要求，修改范例client_example.c，实现类似telnet连接echo服务器的效果：

- ① 为所有socket函数调用添加**错误处理代码**；
- ② 范例中服务器**地址**和**端口**是固定值，请将它们改成允许用户以**命令行参数形式输入**；
- ③ 范例中客户机发送的是固定文本“Hello Network!”，请改成允许**用户输入**字符串，按回车发送；
- ④ 实现**循环**，直至客户机输入“bye”退出。

```
                                argv[0] argv[1] argv[2]
                                |       |       |
[root@localhost 桌面]# ./clnt0 localhost 7
Myself: Hello,11.11
Server: Hello,11.11
Myself: buy,buy,buy
Server: buy,buy,buy
Myself: no.enough.money
Server: no.enough.money
Myself: ^C
[root@localhost 桌面]# clear
```

命令行参数

Ctrl+C退出

任务2：课程表查询服务器（TCP迭代）

功能示例：服务器接收客户机发来的学号等信息，查询后发送给客户机，依次循环……

```
aaa@Ubuntu1804:~$ ./serv1 2000
Server is listening....
Accept a client
client: 003
server: 003 chris 大学物理 高等数学 通信基础
client: 003,2
server: 003 chris 高等数学
```

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 003
server: 003 chris 大学物理 高等数学 通信基础
client: 003,2
server: 003 chris 高等数学
client:
```

服务器地址 服务端口号

第一个客户机可以正常收发

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 003,0
```

第二个客户机发送后无响应

```
aaa@Ubuntu1804:~$ ./serv1 2000
Server is listening....
Accept a client
client: 003
server: 003 chris 大学物理 高等数学 通信基础
client: 003,2
server: 003 chris 高等数学
client: bye
```

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 003
server: 003 chris 大学物理 高等数学 通信基础
client: 003,2
server: 003 chris 高等数学
client: bye
aaa@Ubuntu1804:~$
```

第一个客户机bye退出

```
Server is listening....
Accept a client
client: 003,0
server: 003 chris 大学物理 高等数学 通信基础
client: 003,5
server: 不存在!
```

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 003,0
server: 003 chris 大学物理 高等数学 通信基础
client: 003,5
server: 不存在!
client:
```

第二个客户机收到回复

课程表查询服务器（TCP迭代）

按以下要求，修改范例server_example.c：

- ① 为所有socket函数调用添加错误处理代码；
- ② 范例中服务器地址和端口是固定值，请将它们改成允许用户通过命令行参数形式输入；
- ③ 实现循环，直至客户机输入“bye”退出；
- ④ 服务器迭代地处理客户机请求：查询给定的testlist，将结果回复客户机；一个客户机退出后、继续接受下一个，按Ctrl+C可以终止服务器程序。

实验报告(任务1+任务2)

- ① 说明客户机和服务器的运行情况，附上截图；
源代码要有充分的注释，随报告一起打包。
- ② 设置服务器listen()的backlog为0或1，同时打开多个终端窗口、让4个客户机连接服务器，使用netstat命令观察socket状态变化，对照TCP连接状态图(5-28/5-29/5-30)，说明变化过程以及backlog与客户机完成队列的数量关系。
`netstat -anp | grep “服务器端口号”`
- ③ (选做) 端口为什么要进行字节顺序转换？不转换会有什么情况？
提示：运行不转换代码进行对比。
- ④ (选做) 试验客户机也像服务器一样bind固定端口，看看结果如何？
提示：先在终端中运行命令：`sudo sysctl net.ipv4.tcp_timestamps=0`
让客户机bind固定端口，运行客户机连接服务器，
客户机主动退出、再次运行客户机，netstat观察。

任务3：课程表查询服务器(TCP并发)

功能示例：在任务2的基础上修改服务器代码，以多进程的方式提供并发服务，实现“同时”与多个客户机交替对话

```
aaa@Ubuntu1804:~$ ./serv2 2000
Server is listening....
Accept 59.77.8.101:50366
  From 59.77.8.101:50366: 001,1
  To 59.77.8.101:50366: 001 andy 大学英语
Accept 59.77.8.102:50367
  From 59.77.8.102:50367: 002,2
  To 59.77.8.102:50367: 002 bob C++程序设计
Accept 59.77.8.103:50369
  From 59.77.8.102:50367: 004,2
  To 59.77.8.102:50367: 004 david Java程序设计
  From 59.77.8.103:50369: 003,3
  To 59.77.8.103:50369: 003 chris 通信基础
  From 59.77.8.101:50366: 001,2
  To 59.77.8.101:50366: 001 andy 高等数学
  From 59.77.8.102:50367: bye
Close 59.77.8.102:50367
  From 59.77.8.103:50369: 006,3
  To 59.77.8.103:50369: 006 fox 计算方法
  From 59.77.8.101:50366: bye
Close 59.77.8.101:50366
  From 59.77.8.103:50369: bye
Close 59.77.8.103:50369
```

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 001,1
server: 001 andy 大学英语
client: 001,2
server: 001 andy 高等数学
client: bye
```

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 002,2
server: 002 bob C++程序设计
client: 004,2
server: 004 david Java程序设计
client: bye
```

```
aaa@Ubuntu1804:~$ ./clnt localhost 2000
client: 003,3
server: 003 chris 通信基础
client: 006,3
server: 006 fox 计算方法
client: bye
```


多进程实现并发服务

创建子进程——fork()

*返回值:

成功: 父进程返回子进程的进程号 (pid), 子进程返回0

失败: 返回-1

*说明:

fork创建的子进程与父进程共享代码正文, 可以通过判断pid来区分

*示例:

```
pid = fork();  
if (pid == -1)  
{ return 1;}  
else if (pid == 0)  
{ /* 子进程执行功能 */}  
else  
{ /* 父进程执行功能 */}
```

*提示:

```
#include <signal.h>
```

在创建子进程前加入signal(SIGCHLD,SIG_IGN)可以清除僵尸进程

多进程实现并发服务

- 子进程与父进程示例：

子进程

```
.....  
listen;  
while (1) {  
    xxx=accept;  
    pid = fork();  
    if (pid == -1)  
        { return 1;}  
    else if (pid == 0)  
        { while循环收发数据  
          close(xxx);  
          exit(0);  
        }  
    else  
        {.....}  
    .....  
}
```

父进程

```
.....  
listen;  
while (1) {  
    xxx=accept;  
    pid = fork();  
    if (pid == -1)  
        { return 1;}  
    else if (pid == 0)  
        { while循环收发数据  
          close(xxx);  
          exit(0);  
        }  
    else  
        {.....}  
    .....  
}
```

实验报告(任务3)

- ① 给出客户机和服务器运行时的截图，要求至少3个并发客户机，源代码随报告一起打包。
- ② 服务器accept之后会返回一个用于传输数据的socket，调用fork()会使父子进程同时拥有此socket描述符，父进程分支中是否需要关闭该socket？

提示：分别试验其关闭和不关闭的代码，看运行是否正常；
netstat观察多个客户机退出后的连接状态，考虑系统资源分配.....

任务4：获取网页内容（Python自学）

编写程序，建立与www.people.com.cn的tcp连接，请求网页的内容并保存。

- 步骤：使用Socket建立对指定URL的连接，使用HTTP的GET指令，获取网页的内容。
- 任务：① 将网页的内容以字符串形式保存在txt文件中。
②（选做）将网页中的图片保存到本地文件夹

- 推荐语言：Python

- * 提示：

使用python的socket时，可能需要用到的api：

`socket.connect()`

`socket.send()`

`socket.recv()` 请注意recv()为阻塞接收，编码时可通过循环接收、直到收到的串长度为0停止。

获取网页内容的GET请求：`"GET / HTTP/1.1\r\nHost: 网址\r\n\r\n"`

实验报告(任务4)

描述解决思路，展示运行情况及关键环节，
源代码随报告一起打包。

Python参考资料

Python环境配置(PyCharm IDE):

https://blog.csdn.net/ling_mochen/article/details/79314118

Python基础语法:

<https://www.runoob.com/python/python-basic-syntax.html>

Python socket编程:

<https://www.runoob.com/python/python-socket.html>

Python文件读写:

<https://www.runoob.com/python/python-files-io.html>