操作系统 实验四 socket编程

摘要

以教学 *PPT* 上的示例代码为基础，并依据上一次实验中读写文件与数据传输的经验，补充读写文件与多次收发的功能，使原示例代码从只能进行一次远程数据收发，变成能够读写并用 `TCP` 协议传输比较大的文本文件。本实验中使用了大小为 `3KB` 的文本文件对程序进行了测试，得到了正确的结果。

算法思想和概要设计

算法思想

Server

`server` 方使用 `socket()` 建立 `socket`，与对应套接字地址用 `bind()` 绑定。然后使用 `getsockname()` 函数获得一未用端口号， 并将其打印出来，以供 `client` 程序使用该端口号作为命令参数，与 `server` 建立 `socket` 连接。`server` 方使用 `listen()` 对该端口进行监听。

`server` 在调用 `accept()` 获得 `client` 方发来的连接请求后，产生子进程，使子进程使用一个新的 `socket` 句柄与 `client` 程序通信，而父进程继续在原地址上监听。

服务器进程的子程序调用 `recv()` 收取到 `client` 方发来的部分消息后，将其写入文件中，再收取 `client` 方发来的消息。这与上次实验中的数据通信并读写文件的过程是非常相似的，只不过从通过进程间管道数据传送变成了通过 `socket` 远程数据传输。所以代码实现也非常相似。

服务器进程的子程序接收到 `client` 方 `socket` 关闭后，打印相关信息并结束子进程。

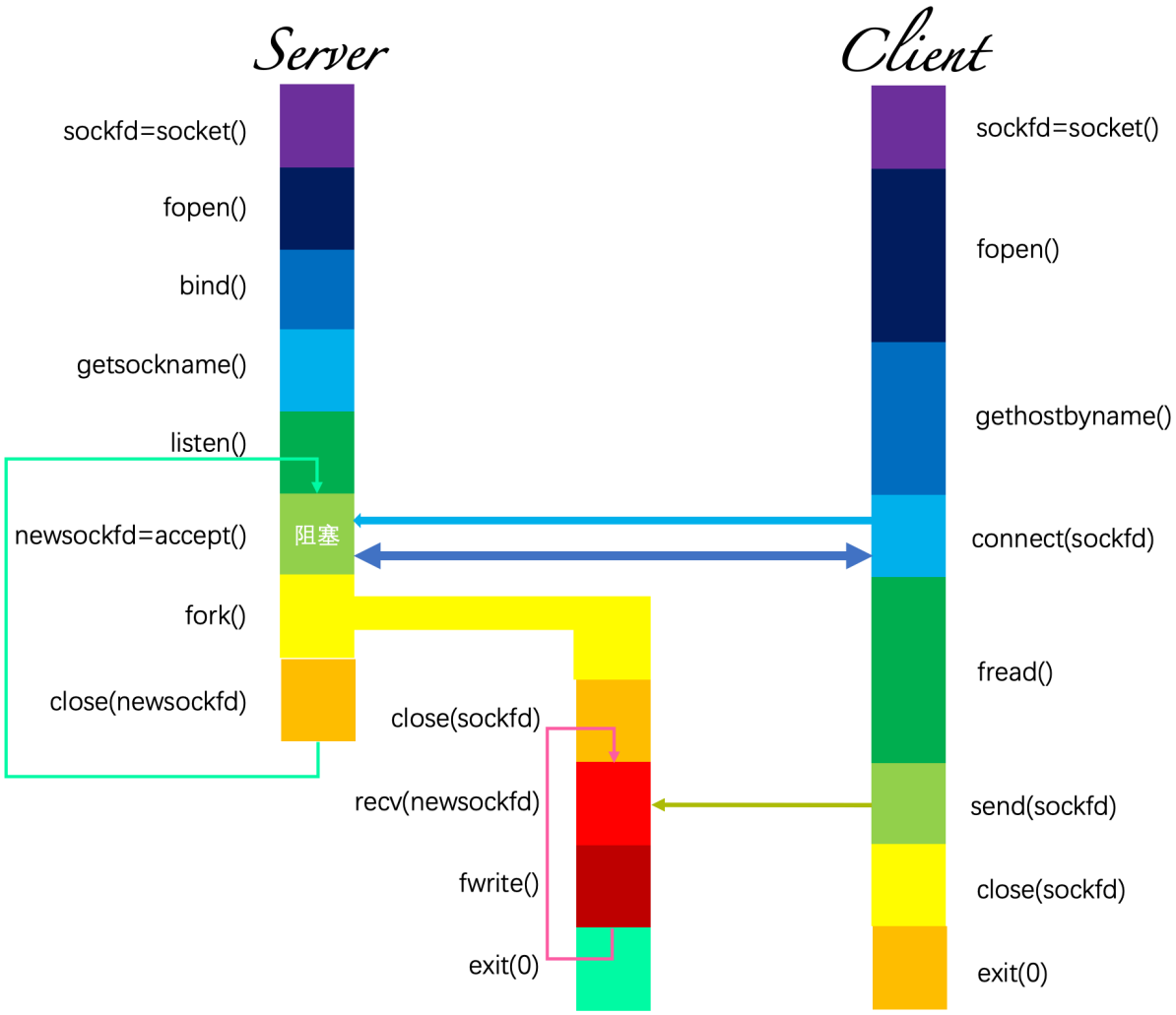
Client

`client` 使用 `socket()` 建立 `socket`。然后通过键入的服务器的名称与端口，调用 `gethostbyname()`，查找 `/etc/hosts` 文件，将服务器主机的 `IP` 地址填入 `hostent` 结构中，并返回指向该结构的指针，得以创建服务器的套接字地址。

通过 `connect()` 连接上述套接字地址对应的服务器。然后读取要发送的文件，并调用 `send()` 通过 `socket` 发送数据到客户端，知道文件被读取完，然后关闭 `socket`，结束进程。

概要设计

以下将以流程图来展示 `client` 和 `server` 概要设计。



数据结构与变量说明

套接字地址

```c
#include <netinet/in.h>
struct sockaddr_in{
     unsigned short          sin_family;   // 表示地址类型
     unsigned short int      sin_port;     // 表示端口号
     struct in_addr          sin_addr;     // 表示32位的IP地址
     unsigned char           sin_zero[8];  // 表示填充字节，一般情况下该值为0
};
struct sin_addr{
     unsigned long      s_addr;
};
```

协议簇

```
AF_UNIX      // (本机通信)
AF_INET      // (TCP/IP – IPv4)
AF_INET6     // (TCP/IP – IPv6)
```

套接字类型

```
SOCK_STREAM // (TCP流)
SOCK_DGRAM  // (UDP数据报)
SOCK_RAW    // (原始套接字)
```

主机地址

```c
struct hostent {
char *h_name;        /* 主机名 */
char **h_aliases;    /* 别名表 */
int h_addrtype;      /* 主机地址类型 */
int h_length;        /* 地址长度 */
char **h_addr_list; /* 域名服务器地址表，以NULL终止 */
};
```

源程序

头文件

```c
/* Client与Server头文件 */
#include <stdio.h>    // io操作
#include <sys/types.h>  // 套接字类型
#include <sys/socket.h> // 协议簇
#include <netinet/in.h> // 套接字地址
#include <netdb.h>      // hostnet
#include <string.h>   // 字符串操作
#include <sys/stat.h> // 文件读写操作
#include <sys/fcntl.h>  // 文件读写操作
#include <stdlib.h>

```

Server

主程序

```c
/* server */
#include "sockcom.h"

main()
{
  int sockfd, newsockfd, length, count;
  char c;
  char buf[1024];
  sockfd = socket(AF_INET, SOCK_STREAM, 0); // 建立 TCP socket, TCP/IP — IPv4
  if (sockfd < 0)
  {
    printf("TCP SOCKET ERROR!\n");
    exit(-1);
  }
  FILE *fp2;
  fp2=fopen("file2.txt","w");   // 打开要将收到数据写入的文件
  struct sockaddr_in server;
  server.sin_family=AF_INET;    // TCP/IP — IPv4
  server.sin_addr.s_addr = INADDR_ANY;  // 接收任意ip (0.0.0.0)
  server.sin_port=0;    // 系统选择一个已释放的端口号
  if (bind(sockfd,(struct sockaddr *)&server,sizeof(server)) < 0) // 绑定sockfd与套接字地址server
  {
    printf("bind stream socket ERROR!\n");
    exit(-1);
  }
  length=sizeof(server);
  if (getsockname(sockfd,(struct sockaddr *)&server,&length) < 0) // 获取套接字sockfd的名字
  {
    printf("getting socket name ERROR!\n");
```

```
    exit(-1);
  }
  printf ("socket port #%d\n", ntohs(server.sin_port));
  listen(sockfd,5); // 创建一个套接口并监听申请的连接，等待连接队列的最大长度为5
  while(1){
    newsockfd = accept(sockfd, (struct sockaddr *)0, (int *)0); // 从等待连接队列中抽取第一个连接，创建一个新的套接口并返回句柄
    if (!fork()){
      close(sockfd);
      bzero(buf,sizeof(buf));
      while ((count=recv(newsockfd,buf,sizeof(buf),0)) > 0){  // 接收数据
        fwrite(buf, 1, 1, fp2);
      }
      if(count < 0)
      {
        printf("Reading stream message ERROR!\n");
        exit(-1);
      }
      printf("\n one client closed.\n");
      exit(0);
    }
    close(newsockfd);
  }
}

```

Client

主程序

```
/* client */
#include "sockcom.h"

main(argc,argv)
int argc;
char **argv;  // 键入服务器地址与端口
{
  int sockfd;
  struct sockaddr_in server;
  struct hostent *hp, *gethostbyname();
  char msg[1024];
  sockfd=socket(AF_INET, SOCK_STREAM, 0); // 建立 TCP socket, TCP/IP — IPv4
  if (sockfd < 0)
  {
    printf("TCP SOCKET ERROR!\n");
    exit(-1);
  }
  FILE *fp1;
  fp1=fopen("file.txt","r");    // 打开要发送的文件
```

```
  if ((hp=gethostbyname(argv[1])) == NULL){ // 通过ip得到服务器地址
    fprintf(stderr,"%s:unknown host\n",argv[1]);
    exit(2);
  }
  server.sin_family=AF_INET;    // TCP/IP — IPv4
  bcopy((char *)hp->h_addr,(char *)&server.sin_addr.s_addr,hp->h_length);
// 服务器地址
  server.sin_port=htons(atoi(argv[2])); // 服务器端口
  if (connect(sockfd,(struct sockaddr *)&server,sizeof(server)) < 0)  //
建立与server的连接
  {
    printf("connecting stream socket ERROR!\n");
    exit(-1);
  }
  while((fread(msg, 1, 1, fp1)) > 0){
    if(!strlen(msg)) break;
    printf(msg);
    if(send(sockfd,msg,strlen(msg),0) < 0)  // 发送数据
    {
      printf("sendint message ERROR!\n");
      exit(-1);
    }
    bzero(msg, sizeof(msg));
  }
  printf("EOF...disconnect\n");
  close(sockfd);
  exit(0);
}

```

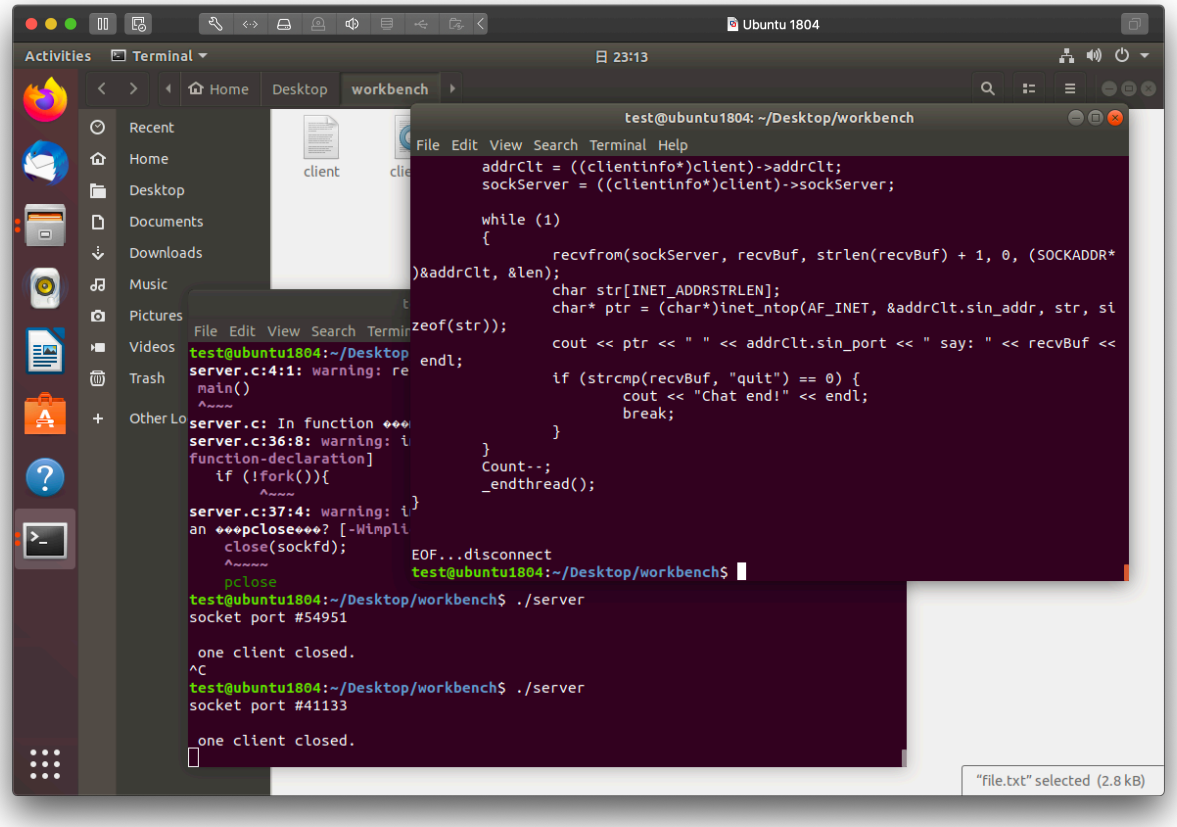
测试

本次用于测试传输的文件是一个大小为 `3KB` 的文本文件，命名为 `file.txt`，内容是 `Winsock` 编程实现的 `UDP` 多线程聊天程序代码。接收 `file.txt` 数据的文件命名为 `file2.txt`。

先打开 `Server`，可见系统选择了空闲端口 `41133`：

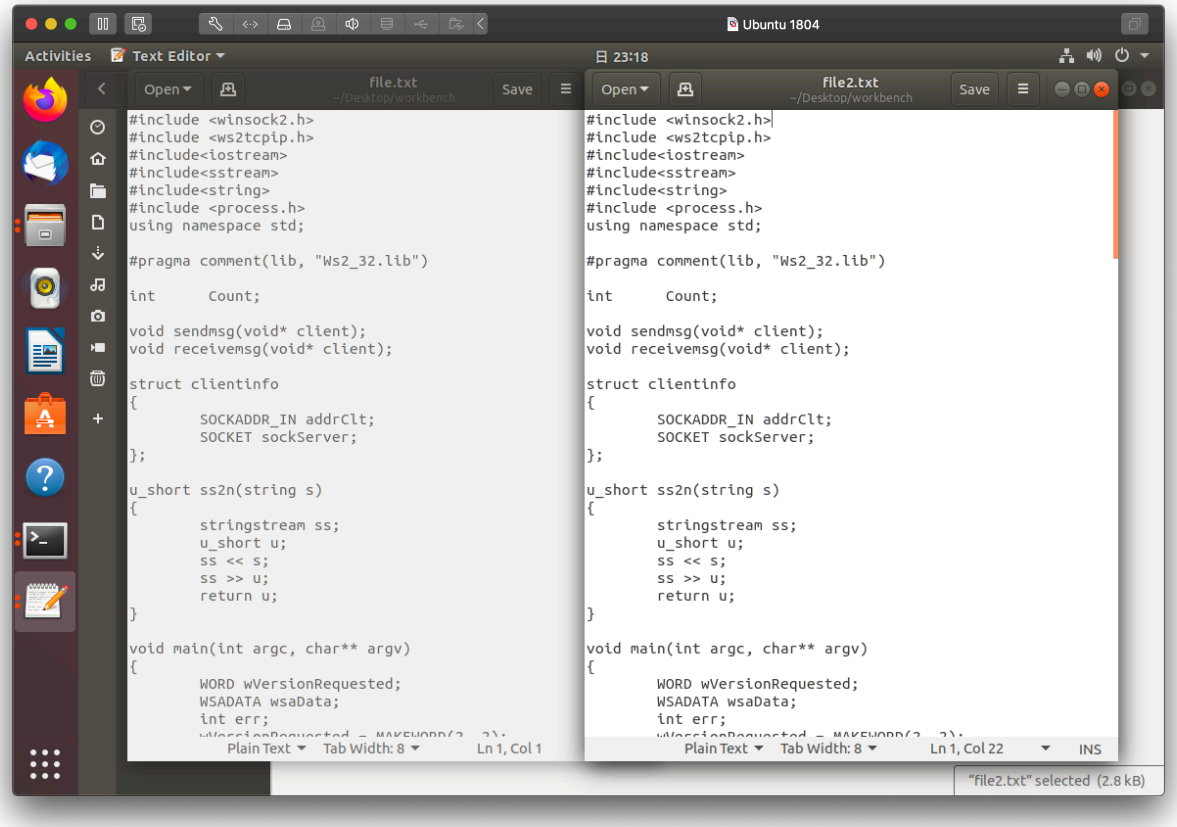
```
test@ubuntu1804: ~/Desktop/workbench
File  Edit  View  Search  Terminal  Help
 one client closed.
^C
test@ubuntu1804:~/Desktop/workbench$ gcc -o server server.c
server.c:4:1: warning: return type defaults to ◆◆◆int◆◆◆ [-Wimplicit-int]
 main()
 ^~~~
server.c: In function ◆◆◆main◆◆◆:
server.c:36:8: warning: implicit declaration of function ◆◆◆fork◆◆◆ [-Wimplicit-
function-declaration]
   if (!fork()){
        ^~~~
server.c:37:4: warning: implicit declaration of function ◆◆◆close◆◆◆; did you me
an ◆◆◆pclose◆◆◆? [-Wimplicit-function-declaration]
    close(sockfd);
    ^~~~~
    pclose
test@ubuntu1804:~/Desktop/workbench$ ./server
socket port #54951

 one client closed.
^C
test@ubuntu1804:~/Desktop/workbench$ ./server
socket port #41133
```

在打开 `Client`，使用命令 `./client 127.0.0.1 41133` 传递服务器参数。

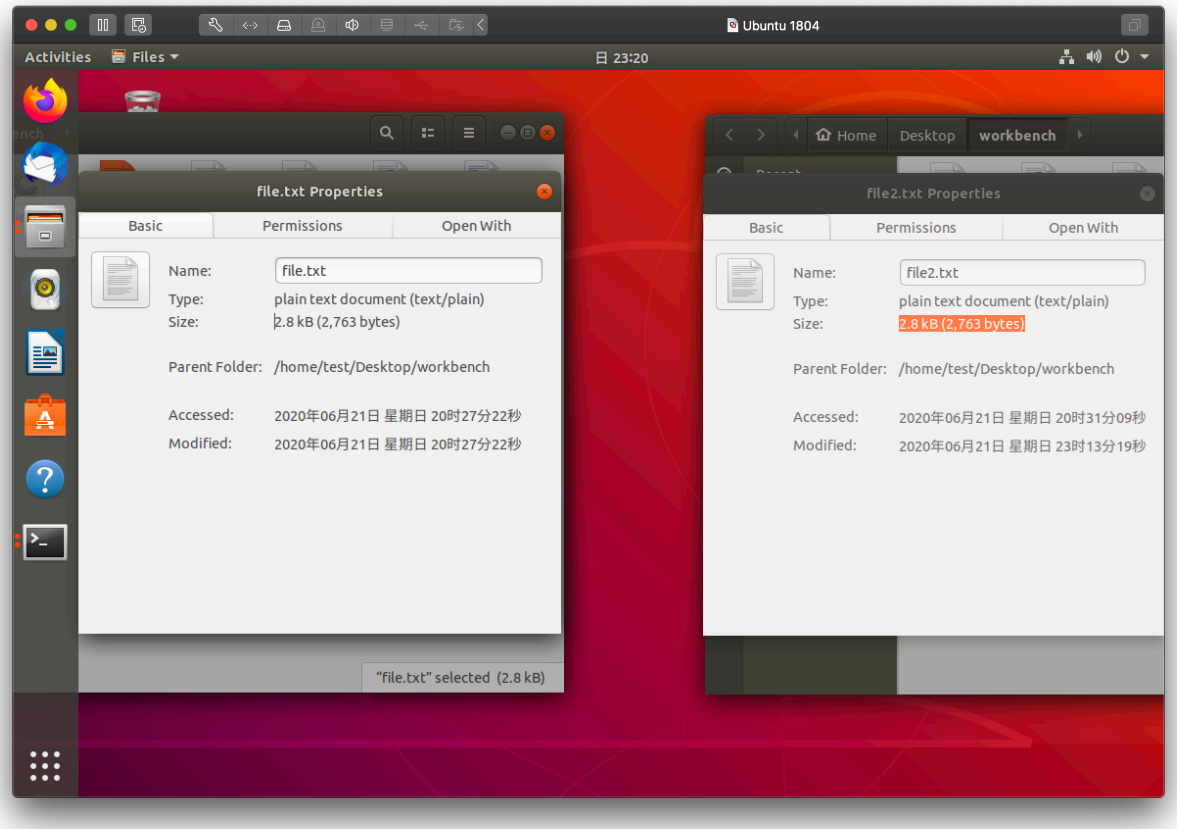


从 `Client` 的运行输出信息可以看出 `Client` 完整地传输了 `file.txt`，且传输完成后立即关闭。`Server` 也收到了 `Client` 的关闭信息，输出 `one client closed`，并继续监听端口等待下一个连接。

打开 `file.txt` 和 `file2.txt` 对比，完全一致，如下图。



打开文件信息查看，也完全一致，说明程序能完成在 `client` 和 `server` 间传输较大文本文件的功能：



改进与体会

改进

本次实验的代码是从 *PPT* 上的单次传输的示例代码的基础上改进而来，其中又用到了第三次实验读写文件的知识，使用和读写文件相似的流程完成 `client` 和 `server` 间的数据传输，从而能够读写并用 `TCP` 协议传输比较大的文本文件。由于在本学期的 **数据通信** 课上做过很多 *Winsock* 编程实验，所以对 *socket* 编程比较清楚，并未出现什么错误。

体会

由于在本学期的 **数据通信** 课上做过很多 *Winsock* 编程实验，所以对 *socket* 编程比较熟悉。这次在 *Linux* 上进行 *socket* 编程实验，发现其使用的基本函数，如 `socket()`、`bind()`、`listen()` 等的类型、用法，还有各种数据结构，以及实现流程，与 *Winsock* 编程相比没有很大的不同，所以实验做起来没有遇到什么困难（毕竟在 *Winsock* 编程实验中比这个难得多的功能，如多线程 `UDP` 聊天程序、`web` 服务器都实现过）。

这次实验中我综合运用了父子进程、文件读写、`socket` 等功能完成了 *socket* 编程，是对课堂上所学知识等一次很好的复习与锻炼。同时接触了类 **Unix** 系统的 *socket* 编程，扩充了 *socket* 编程知识面。

by 刘浩文 517021911065