
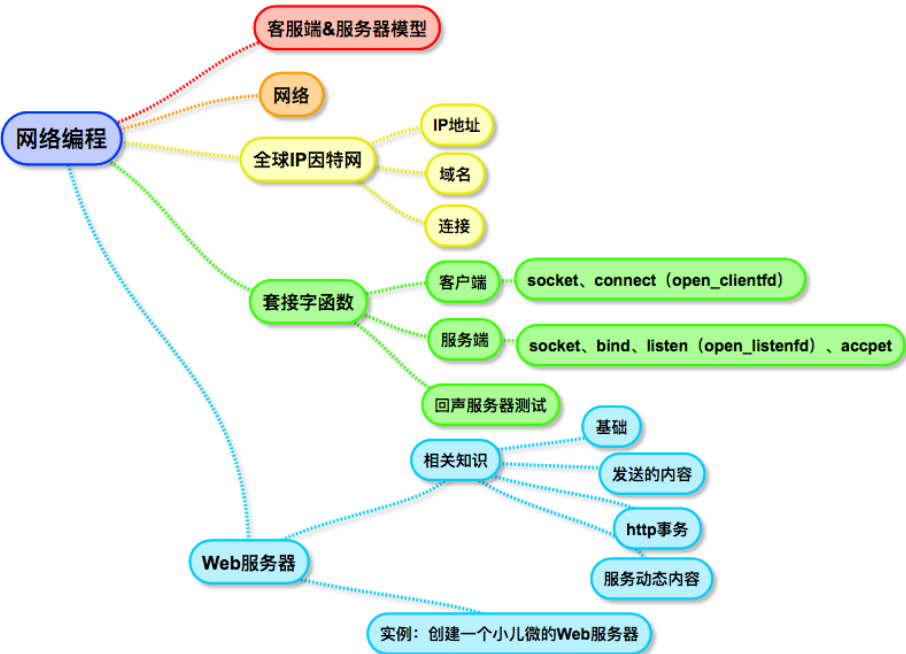


《深入理解计算机系统》| 网络编程

 唐鱼的学习探索 关注

2018.05.08 05:01:02 字数 3,482 阅读 802




 小鹅通

小鹅通商家数突破100万

— 在线直播课堂系统,最高直减5888元 —

立即注册

广告

 唐鱼的学习探索 关注

总资产29 (约2.81元)

如何高效的准备一次考试

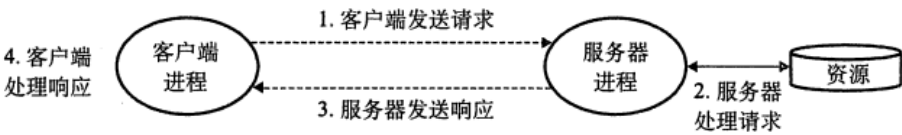
阅读 1,737

都9102年了, 你还不知道anki是什么

阅读 102

网络应用集成了我们已经学到的很多概念：进程、信号、字节顺序、存储器映射、动态分配等，同时客服端-服务器模型是一个新的知识，我们将所有的这些结合起来，创建一个微小的Web服务器，提供浏览器静态和动态的访问。

1.1 客户端--服务器模型



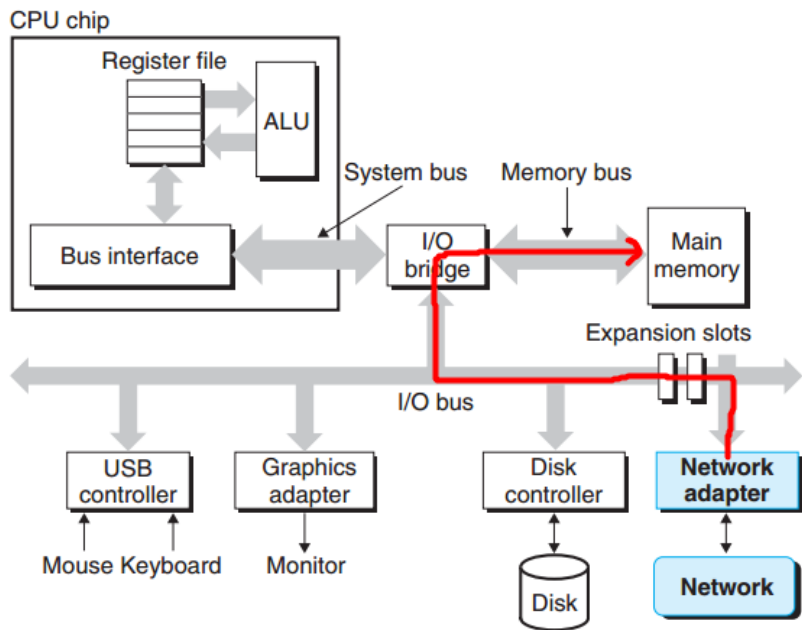
每一个网络应用都是基于一个客户端进程和一个服务器进程建立起来的，服务器管理资源，客服端请求某种服务，客户端和服务端都是一个运行中的程序。典型的示意图如下：

我们以一个邮件客服端访问文件为例：

- ① 客户端发送一个浏览文件的请求给Web服务器；
- ② 服务器接收请求以后，解释它，从资源中取出相应的文件；
- ③ 服务器将文件发送到请求的客户端；

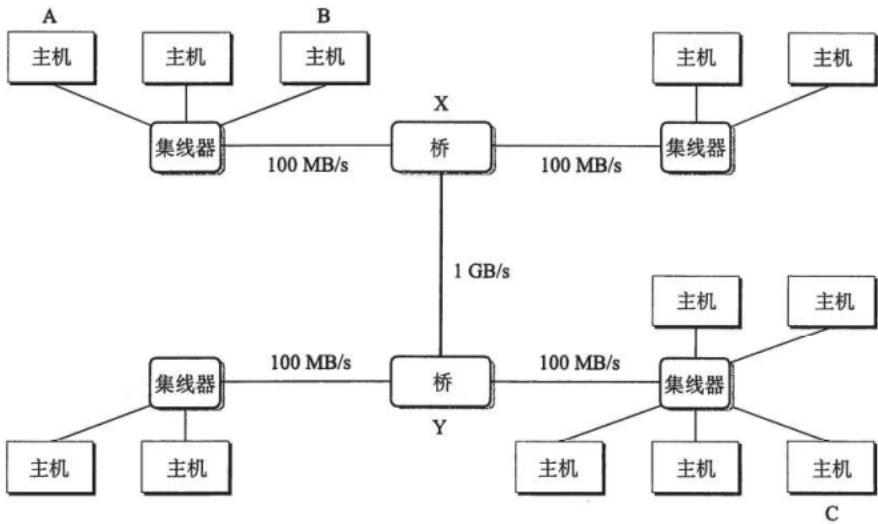
④ 客户端接收并显示在屏幕上。

1.2 网络



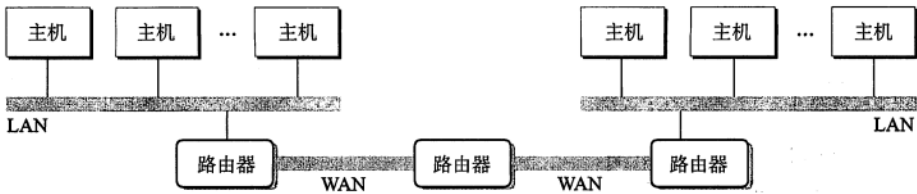
网络是连接客户端与服务端之间的通路，对于单个的主机而言，网络只是插在IO总线上的的一种，从网络上接收的数据经过适配器-IO总线-桥-存储器总线-主存：

上图只是一个简图，我们来简单的讨论一下具体的实现



在网络建立的初期，是通过一个集线器，实现无差别的所有主机之间的数据交换。同交换机不同，集线器不是点到点，它的每次转发都是针对所有的主机，这就形成了一个局域网。后来又加入了桥，实现局域网直接的互相连接，总体的架构图如下：

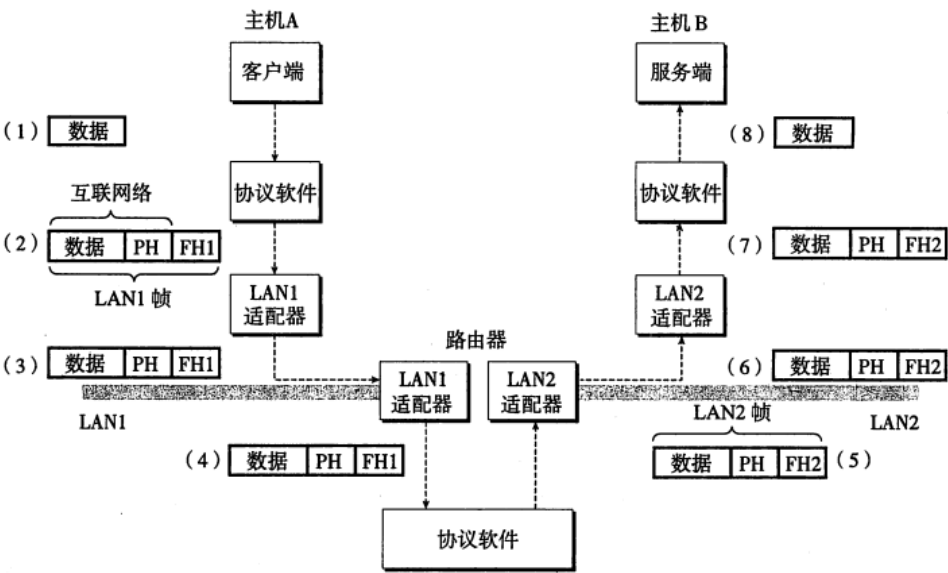
新加入的桥，可以实现自动学习，哪儿主机可以通过哪个端口方便到达，如果将A发送到B，当到达桥X的时候，桥发现A到B就是本局域网的事情，就会丢弃到这个帧。而当A发送到C的时候，桥X只会将帧发送到桥Y再由桥Y分发到C，而节省了带宽。



路由器有两个不同的端口，WAN（广域网）连接更大的局域网，LAN（本地网）连接本地主机，结构如下：

网络上的硬件已经全部连接好了，但是我们的主机各有不同，有不同的主机、不同的系统各种千差万别，是如何通过这个网络发送数据实现求同存异的。

通过协议发送，我们人类社会也是一样的，为了达成某种一致性，不至于以后扯皮，我们会实现签订一个协议，将需要调解的问题和责任全部列出，而且竟可能倾其所有，在以后的操作中不会有含糊不清或者界定不明的问题，也就能更好的合作，计算机网络也是一样通过建立一套TCP/IP协议族，实现不同主机之间的数据传输：



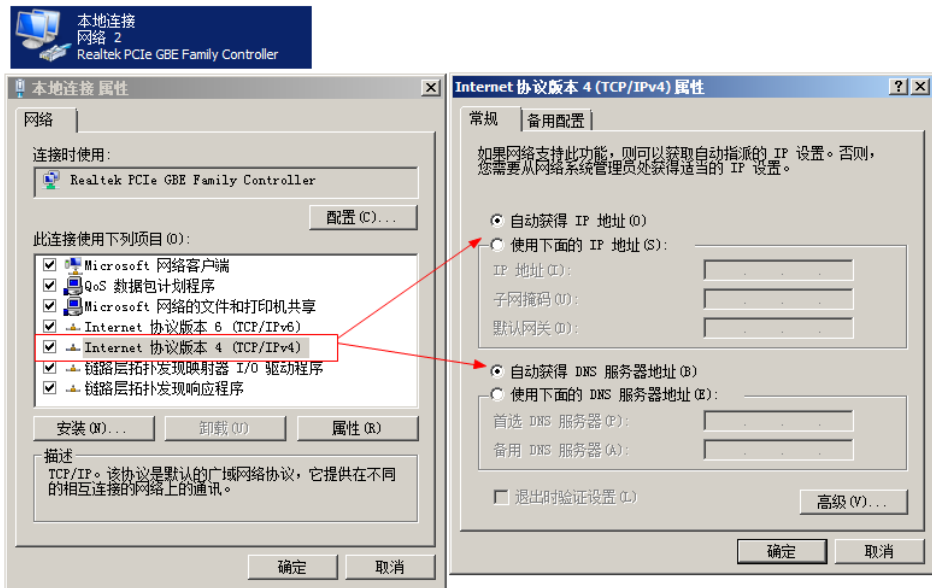
下图是客户端A发送数据到服务端B的过程：

- ① 客户端将需要发送的数据拷贝到缓冲区；
- ② 主机A上运行的协议软件（TCP/IP）把实际要发送的数据加入PH（互联网包头）和FH1（LAN1帧头），创建一个LAN1的帧发送到网络适配器中；
- ③ 主机上的网络适配器（连接在IO总线）收到了这个LAN1帧以后通过网线发送到路由器的LAN1端口；
- ④ 收到了这个LAN1帧的时候，将其发送到协议软件上；
- ⑤ 路由器的协议软件将LAN1帧的旧的LAN1帧头剥落，加入到实际发送的主机LAN2帧头，并将其发送到路由器LAN2端口上；
- ⑥ 路由器LAN2将数据拷贝到网线上；

⑦ 主机B的适配器从网线上收到了这个帧的时候，将其传送到协议软件；

⑧ 协议软件帧头和包头吧实际的数据拷贝出来；

1.3 全球IP因特网



每台主机都支持TCP/IP协议，并运行着这个软件，IP协议提供基本的命名方法和传送机制，我们来看看windows上运行的这个协议：

① IP地址

```
/* Internet address structure */
struct in_addr {
    unsigned int s_addr; /* Network byte order (big-endian) */
};
```

IP地址是一个无符号的32位整数，存储在如下的结构中：

```
32 unsigned long int htonl(unsigned long int hostlong); → 将本机字节顺序转网络字节顺序
16 unsigned short int htons(unsigned short int hostshort);

32 unsigned long int ntohl(unsigned long int netlong); → 将网络字节顺序转主机字节顺序
16 unsigned short int ntohs(unsigned short int netshort);

      返回：按照网络字节顺序的值。
      返回：按照主机字节顺序的值。
```

统一的网络字节顺序是以大端的字节顺序，IP地址也是使用大端法存放的，如果不同的话需要使用一些函数进行转换：

IP地址有时候是使用点分的十进制表示的如：192.168.1.1，而下面的函数可以实现将点分十进制的IP地址转成网络字节顺序的IP地址：

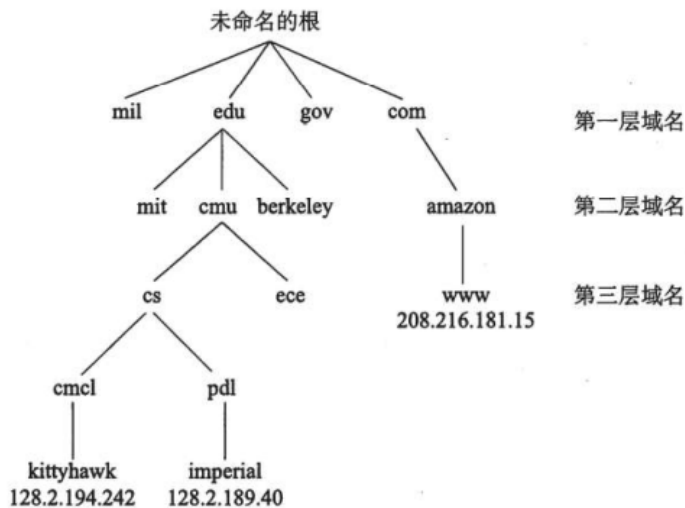
```
int inet_aton(const char *cp, struct in_addr *inp);

char *inet_ntoa(struct in_addr in);

      a : application      返回：若成功则为 1，若出错则为 0。
      n : network

      返回：指向点分十进制字符串的指针。
```

② 域名



域名是一种将IP地址映射为一组人性化的字符串的机制，而域名的集合是一个层次结构：

第一层域名是一个非营利性组织定义的，常见的有.com、edu、gov等

第二层cmu.edu是按照先后顺序获得的，一旦获得了cmu，就可以向下生成cs\ece等

```
/* DNS host entry structure */
struct hostent {
    char    *h_name;           /* Official domain name of host */
    char    **h_aliases;       /* Null-terminated array of domain names */
    int     h_addrtype;        /* Host address type (AF_INET) */
    int     h_length;          /* Length of an address, in bytes */
    char    **h_addr_list;     /* Null-terminated array of in_addr structs */
};
```

每一组由字符串映射到IP地址的数据结构如下图：

```
struct hostent *gethostbyname(const char *name);
    返回：若成功则为非 NULL 指针，若出错则为 NULL 指针，同时设置 h_errno。
struct hostent *gethostbyaddr(const char *addr, int len, 0);
    返回：若成功则为非 NULL，若出错则为 NULL 指针，同时设置 h_errno。
```

而负责解析这种映射的称为DNS数据库，保存着成千上万的上图中的条目结构。应用程序可以通过gethostbyname和gethostbyaddr函数显示的从DNS数据库中检索条目：

```
GNU nano 2.2.6                                File: hostinfo.c

#include "csapp.h"

int main(int argc, char **argv)
{
    char **pp;
    struct in_addr addr;
    struct hostent *hostp;

    if(argc != 2)
    {
        fprintf(stderr, "usage: %s < domain name or dotted-decimal>\n", argv[0]);
        exit(0);
    }
}
```

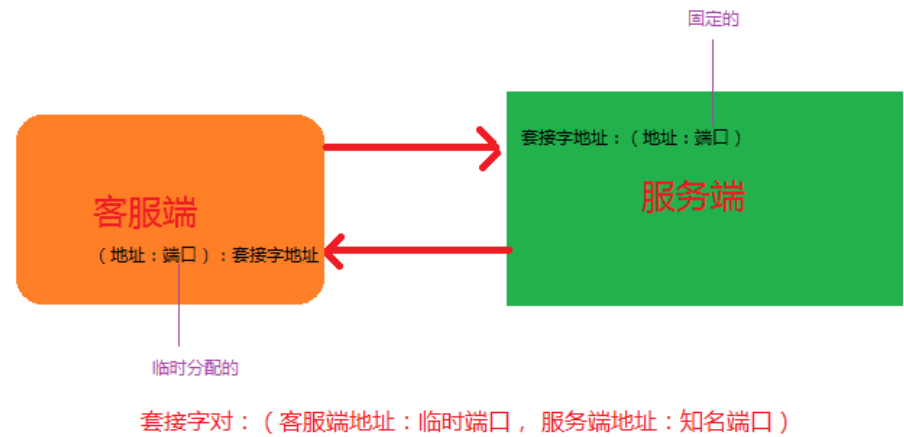
我们来看一个程序：

```
pi@raspberrypi:~/code $ ./hostinfo www.baidu.com
official hostname : www.a.shifen.com
alias : www.baidu.com
address: 119.75.217.109
address: 119.75.218.70
pi@raspberrypi:~/code $ ./hostinfo raspberrypi
official hostname : raspberrypi
address: 127.0.1.1
pi@raspberrypi:~/code $ ./hostinfo 127.0.0.1
official hostname : localhost
address: 127.0.0.1
pi@raspberrypi:~/code $
```

从给定的域名或者IP地址中打印出相应的主机名、别名和地址列表：

可以看出，多个域名可以映射多个IP地址

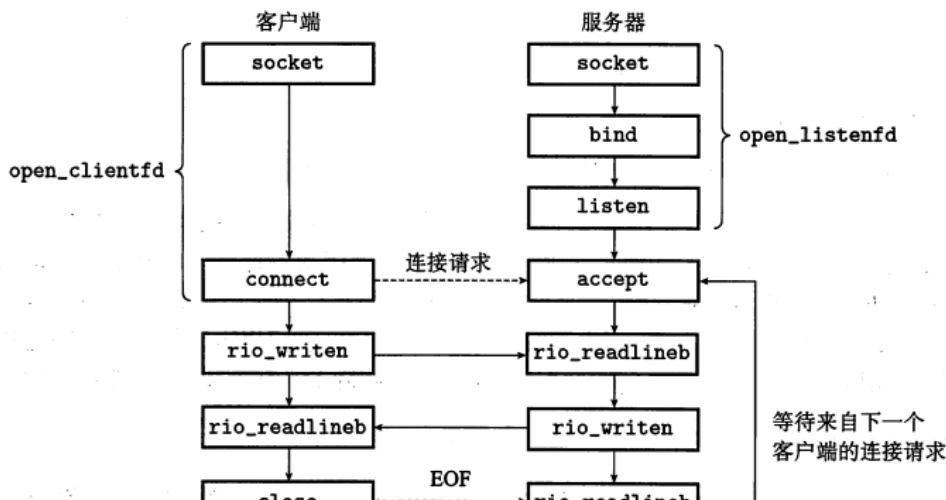
③ 因特网的连接



我们通常使用的是一个套接字完成一个客户端与服务端之间的双向连接的

每个端是由该端的地址：端口组成，其中服务端的端口通常是固定的，而客户端的端口是临时分配的，当形成了一个套接字对的时候，就可以开始双向通信了。

1.4 套接字函数



当我们进行客户端与服务端之间的双向通信的时候，我们使用的是一组套接字函数，来看一张总的图：

客户端：

① socket函数：创建一个套接字描述符

我们通常使用clientfd = socket (AF_INET, SOCK_STREAM, 0) ；来描述，其中AF_INET表示使用互联网，SOCK_STREAM表示使用套接字连接的一个端点；

```
int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

成功指向套接字对 连接的服务端地址 返回：若成功则为 0，若出错则为 -1。

② connect函数：客户端建立同服务器的连接

```
1 int open_clientfd(char *hostname, int port)
2 {
3     int clientfd;
4     struct hostent *hp;
5     struct sockaddr_in serveraddr;
6
7     if((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
8         return -1;
9     if((hp = gethostbyname(hostname)) == NULL)
10        return -2;
11    bzero((char *)&serveraddr, sizeof(serveraddr));
12    serveraddr.sin_family = AF_INET;
13    bcopy((char *)hp->h_addr_list[0], (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
14    serveraddr.sin_port = htons(port);
15
16    if(connect(clientfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
17        return -1;
18    return clientfd;
19 }
20
21 /* 我们将typedef struct sockaddr SA 再将soerveraddr强制转换 */
```

③ 客户端封装socket和connect函数：open_clientfd函数

我们首先创建了套接字的描述（行7），然后检索DNS主机条目，并拷贝第一个IP地址到serveraddr中，发起一个connect（行16）请求，成功时返回clientfd给调用函数。

服务端：

① bind函数：高数内核将my_addr中的服务器套接字地址和套接字描述符sockfd联系起来：

原型是：int bind(int sockfd, struct sockaddr *my_addr, int addrlen);

② listen函数：被动的监听，告诉内核被服务端使用

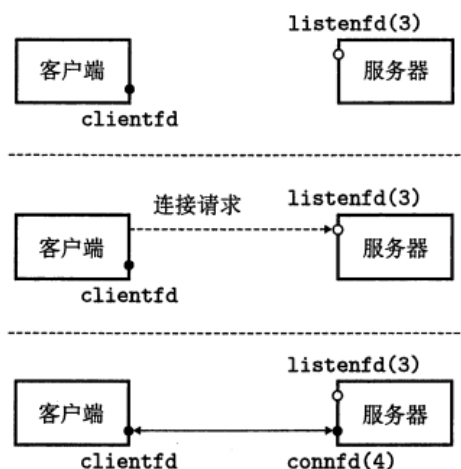
原型是：int listen(int sockfd, int backlog);将sockfd转化为监听套接字

```
1  int pen_listenfd(int port)
2  {
3      int listenfd, optval=1;
4      struct sockaddr_in serveraddr;
5      // 创建套接字描述listenfd
6      if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
7          return -1;
8      // 配置服务器
9      if(setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
10                     (const void *)&optval, sizeof(int)) < 0)
11          return -1;
12      // 初始化套接字地址结构serveraddr
13      bzero((char *) &serveraddr, sizeof(serveraddr));
14      serveraddr.sin_family = AF_INET;
15      serveraddr.sin_addr.s_addr = htonl(INADDR_ANY); // 任何IP地址
16      serveraddr.sin_port = htons((unsigned short)port);
17      // 调用bind函数做好准备
18      if(bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
19          return -1;
20      // 转化为监听描述符
21      if(listen(listenfd, LISTENQ) < 0)
22          return -1;
23
24      return listenfd;
25 }
```

③ 封装：bind和listen成为open_listenfd函数

④ accept函数：等待客户端的连接，创建已连接描述符

监听描述符创建一次存在于整个生命周期，已连接描述符只存在于一个客户端，可以创建并发服务器。



第一步：服务器调用accept，等待连接到达监听描述符；

第二步：客户端调用connect函数，发送一个连接请求到监听描述符；

第三步：打开一个已连接描述符connfd（4）通过connfd和clientfd交换数据。

实例：回声（echo）客户端与服务端

```
GNU nano 2.2.6 File: echoclient.c

#include "csapp.h"

int main(int argc, char **argv)
{
    int clientfd, port;
    char *host, buf[MAXLINE];
    rio_t rio;

    if(argc != 3)
    {
        fprintf(stderr, "usage: %s <host> <port>\n", argv[0]);
        exit(0);
    }
    host = argv[1];
    port = atoi(argv[2]);

    clientfd = Open_clientfd(host, port); ← 建立连接
    Rio_readinitb(&rio, clientfd);

    while(Fgets(buf, MAXLINE, stdin) != NULL)
    {
        Rio_writen(clientfd, buf, strlen(buf)); ← 发送到服务器一个字符串
        Rio_readlineb(&rio, buf, MAXLINE);
        Fputs(buf, stdout); ← 接收返回的字符串
    }

    Close(clientfd); 显示的关闭，一个好的习惯
    exit(0);
}
```

客户端：echoclient.c

```
GNU nano 2.2.6 File: echoserver.c

#include "csapp.h"

void echo(int connfd);

int main(int argc, char **argv)
{
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;
    struct hostent *hp;

    char *haddrp;
    if(argc != 2)
    {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(0);
    }
    port = atoi(argv[1]);

    listenfd = Open_listenfd(port); 创建监听描述符

    while(1)
    {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen); ← 已连接描述符
        hp = Gethostbyaddr((const char *)&clientaddr.sin_addr.s_addr,
                           sizeof(clientaddr.sin_addr.s_addr), AF_INET);
        haddrp = inet_ntoa(clientaddr.sin_addr);
        printf("server connected to %s (%s)\n", hp->h_name, haddrp);

        echo(connfd);
        Close(connfd); ← 显示关闭
    }

    exit(0);
}
```

```
#include "csapp.h"

void echo(int connfd)
{
    size_t n;
    char buf[MAXLINE];
    rio_t rio;

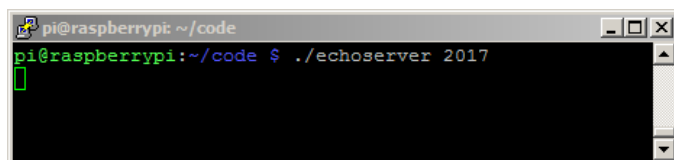
    Rio_readinitb(&rio, connfd);
    while((n = Rio_readlineb(&rio, buf, MAXLINE)) != 0)
    {
        printf("server received %d bytes \n", n);
        Rio_writen(connfd, buf, n);
    }
}
```

发送给客户端

服务端：echoserveri.c

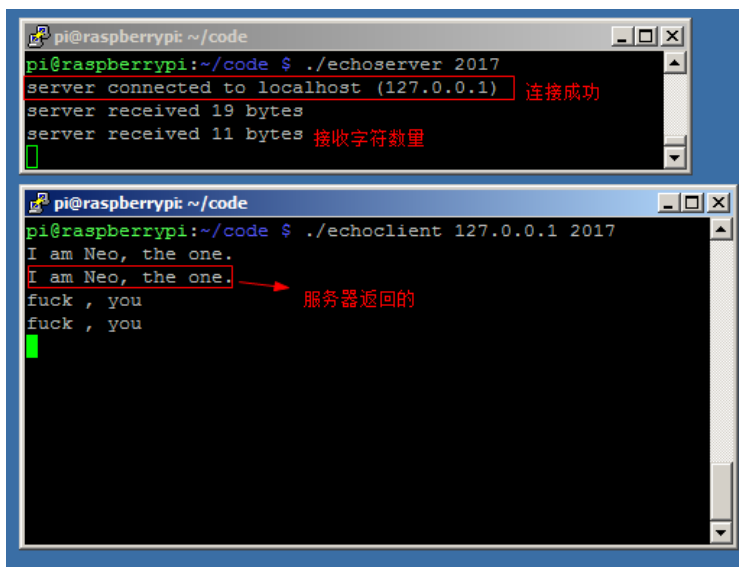
运行效果：

第一步：启动服务器2017端口



```
pi@raspberrypi: ~/code
pi@raspberrypi:~/code $ ./echoserver 2017
```

第二步：客服端开始连接本机127.0.0.1 2017端口，并发送字符串



```
pi@raspberrypi: ~/code
pi@raspberrypi:~/code $ ./echoserver 2017
server connected to localhost (127.0.0.1) 连接成功
server received 19 bytes
server received 11 bytes 接收字符数量

pi@raspberrypi: ~/code
pi@raspberrypi:~/code $ ./echoclient 127.0.0.1 2017
I am Neo, the one.
I am Neo, the one. 服务器返回的
fuck , you
fuck , you
```

1.5 Web服务器相关知识

① 基础：

Web客户端（浏览器）与服务器之间的交互是基于一个HTTP协议，同常规的FTP协议不同，传输的是超文本标记语言（HTML）

② Web服务器发送的内容

是一个MIME类型的序列

MIME 类型	描述
text/html	HTML 页面
text/plain	无格式文本
application/postscript	Postscript 文档
image/gif	GIF 格式编码的二进制图像
image/jpeg	JPEG 格式编码的二进制图像

以两种不同的方式发送到浏览器：

- 1> 取一个磁盘文件，返回给浏览器（静态内容）；
- 2> 取一个可执行文件，返回给浏览器（动态内容）。

URL：通用资源定位符（Universal Resource Locator）

URL是为每个文件定位使用的，其实也就是每个文件的名字，如：

访问：`http://www.google.com:80/index.html`请求/index.html文件静态内容

访问：`http://www.google.com:8000/cgi-bin/adder?15000&213`请求可执行文件cgi-bin/adder

注：其中1500&213是执行文件的两个传入参数

注：后缀中的“/”不代表根目录，请求的时候所有服务器默认为主页，解释为/index.html

③ HTTP事务

```
MatrixIMac:~ Neo$ telnet www.aol.com 80
Trying 54.148.112.228...
Connected to www.aol.com.
Escape character is '^['.
```

1> 请求

我们使用telnet 对www.aol.com 80端口进行连接，然后输入我们的请求：由两部分组成

请求行：GET / HTTP/1.1要求获取/index.html文件的内容；

请求头：host:www.aol.com附加额外信息；

最后我们以一个换行符号将我们的请求发送给服务器。

2> 响应

响应由三部分组成

响应行：HTTP/1.1 301 Moved Permanently ;

多个响应头和响应主体构成

Status code	Status message	Description
200	OK	Request was handled without error.
301	Moved permanently	Content has moved to the hostname in the Location header.
400	Bad request	Request could not be understood by the server.
403	Forbidden	Server lacks permission to access the requested file.
404	Not found	Server could not find the requested file.
501	Not implemented	Server does not support the request method.
505	HTTP version not supported	Server does not support version in request.

其中状态码有以下几种：

④ 服务动态内容

1> 浏览器如何将参数传递给服务器

使用？符号分割文件名和参数，使用&符号分割不同的参数；

2> 服务器如何将参数传递给子进程

如果一个服务器收到一个浏览器（客户端）发送的一个类似请求：

GET /cgi-bin/adder?15000&213 HTTP/1.1

我们的adder程序遵照CGI标准编写，这样子进程将传入CGI的环境变量QUERY_STRING设置为15000&213，这样在adder程序运行的时候就可以调用getenv（获取环境变量）来获得参数

3> 服务器如何将其他信息传递给子进程

Environment variable	Description
QUERY_STRING	Program arguments
SERVER_PORT	Port that the parent is listening on
REQUEST_METHOD	GET or POST
REMOTE_HOST	Domain name of client
REMOTE_ADDR	Dotted-decimal IP address of client
CONTENT_TYPE	POST only: MIME type of the request body
CONTENT_LENGTH	POST only: Size in bytes of the request body

依照CGI标准定义的函数还可以设置环境变量，有下面这些可以使用

4> 子进程将输出发送到哪里

子进程加载并运行CGI程序以前，将CGI程序的标准输出从定位到了服务端已关联描述符（使用dup2函数），这样一来任何标准输出都会发送到服务端去了。同时子进程还要负责生成content-type和content-length两个响应头，解释所发送的内容，以及终止的空行。

```
GNU nano 2.2.6 File: adder.c

#include "csapp.h"

int main(void)
{
    char *buf, *p;
    char arg1[MAXLINE], arg2[MAXLINE], arg3[MAXLINE];
    int n1 = 0, n2 = 0;

    if((buf = getenv("QUERY_STRING")) != NULL)
    {
        p = strchr(buf, '&');
        *p = '\0';
        strcpy(arg1, buf);
        strcpy(arg2, p+1);
        n1 = atoi(arg1);
        n2 = atoi(arg2);
    }

    sprintf(content, "Welcome to add.com");
    sprintf(content, "%s THE Internet addition portal.\r\n<p>", content);
    sprintf(content, "%s The answer is : %d + %d = %d \r\n<p>", content, n1, n2, n1+n2);
    sprintf(content, "%s Thanks for visiting!\r\n", content);

    printf("Content-length: %d\r\n", (int)strlen(content));
    printf("Content-type: text/html\r\n\r\n");
    printf("%s", content);
    fflush(stdout);

    exit(0);
}
```

一个标准的CGI程序adder，只是简单的将传入的参数相加，并返回给客户端

1.6 实例：创建一个微小的Web服务器

我们汇总我们已经学到的所有内容，创建一个只有200多行代码的小型Web服务器，不过麻雀虽小，东西还是满多的，可以实现对静态和动态内容的访问，我们直接上完整的代码。

① 主程序main部分：

```
int main(int argc, char **argv)
{
    int listenfd, connfd, port, clientlen;
    struct sockaddr_in clientaddr;

    if(argc != 2)
    {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(1);
    }

    port = atoi(argv[1]);
    listenfd = Open_listenfd(port);
    while(1)
    {
        clientlen = sizeof(clientaddr);
        connfd = Accept(listenfd, (SA *)&clientaddr, &clientlen);
        doit(connfd);
        Close(connfd);
    }
}
```

首先通过特定的端口创建一个监听描述符，然后就进入了无限循环中，通过accept函数创建已连接的描述符connfd，执行doit事务。

② doit事务

```
void doit(int fd)
{
    int is_static;
    struct stat sbuf;
    char buf[MAXLINE], method[MAXLINE], uri[MAXLINE], version[MAXLINE];
    char filename[MAXLINE], cgiargs[MAXLINE];
```

简书

首页

下载APP

搜索

Q

Aa

beta

登录

注册

```
Rio_readline(&rio, buf, MAXLINE);
sscanf(buf, "%s %s %s", method, uri, version);
if(strcasecmp(method, "GET"))
{
    clienterror(fd, method, "501", "Not Implemented", "Tiny does not implement this method");
}
read_requesthdrs(&rio); // 忽略请求头，不做任何处理
// Parse URI form Get request
is_static = parse_uri(uri, filename, cgiargs);
if(stat(filename, &sbuf) < 0)
{
    clienterror(fd, filename, "404", "Not found", "Tiny couldn't find this file");
    return ;
}
if(is_static) // static content
{
    if(!(S_ISREG(sbuf.st_mode)) || !(S_IRUSR & sbuf.st_mode))
    {
        clienterror(fd, filename, "403", "Forbidden", "Tiny couldn't read the file");
        return ;
    }
    serve_static(fd, filename, sbuf.st_size); // 如果是静态内容，调用静态处理函数
}
else
{
    if(!(S_ISREG(sbuf.st_mode)) || !(S_IXUSR & sbuf.st_mode))
    {
        clienterror(fd, filename, "403", "Forbidden", "Tiny couldn't run the CGI program");
        return ;
    }
    serve_dynamic(fd, filename, cgiargs); // 如果是动态内容，调用动态处理函数
}
}
```

推荐阅读

我的新宠植物

阅读 707

昨天刚发现的挣钱小渠道 今天就有收入了

阅读 12,149

果然，我们都被周扬青骗了

阅读 18,916

枕上书续(二十三)银装素裹下的暗流涌动

阅读 7,240

他画一只鸟价值1000万，网友质疑是儿童水平，专家：你看鸟眼睛

阅读 20,866

③ 解析URI函数：parse_uri

首先，我们解析请求行，并且只处理GET方法，然后对URI进行解析，返回一个标识确定到底是静态页面还是动态页面，分别调用相应的函数处理。

```
int parse_uri(char *uri, char *filename, char *cgiargs)
{
    char *ptr;
    if(!strstr(uri, "cgi-bin")) // 静态内容:
    {
        strcpy(cgiargs, ""); // 清除CGI参数串
        strcpy(filename, ".");
        strcpy(filename, uri); // 转化为一个相对路径名，如果为/
        if(uri[strlen(uri)-1] == '/') // 则默认转向home.html页面
            strcat(filename, "home.html");
        return 1;
    }
    else // 是可执行文件: 动态内容
    {
        ptr = index(uri, '?'); // 定位到?处，抽取参数
        if(ptr)
        {
            strcpy(cgiargs, ptr+1);
            *ptr = '\0';
        }
        else
        {
            strcpy(cgiargs, "");
            strcpy(filename, ".");
            strcpy(filename, uri); // 转化为一个相对于Unix文件名
        }
        return 0;
    }
}
```

小鹅通

小鹅通商家数突破100万

在线直播课堂系统,最高直减5888元

立即注册

广告

首先是确定到底是静态内容还是动态内容，如果是静态内容，首先清除CGI参数串，然后将URI转化为一个相对路径名，如果没有指定，默认的转向到home.html页面，返回1；如果是动态的内容，定位到？后面，抽取相应的参数，将URI转化为一个Unix文件名，返回0；

④ 忽略请求头：read_requesthdrs函数

写下你的评论...

评论0 赞2 ...

```

Rio_readlineb(rp, buf, MAXLINE);
while(strcmp(buf, "\r\n"))
{
    Rio_readlineb(rp, buf, MAXLINE); 读取并忽略
    printf("%s", buf);
}
return ;
}

```

是否在终止报头处

只是简单的已报头结束的地方循环读取整个报头，然后不做任何处理，忽略它；

⑤ 处理静态页面：serve_static 和 get_filetype

```

void serve_static(int fd, char *filename, int filesize)
{
    int srcfd;
    char *srcp, filetype[MAXLINE], buf[MAXBUF];

    get_filetype(filename, filetype); 判断文件类型: html、gif、jpeg、无格式
    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    sprintf(buf, "%sServer: Tiny Web Server\r\n", buf);
    sprintf(buf, "%sContent-length: %d\r\n", buf, filesize);
    sprintf(buf, "%sContent-type: %s\r\n\r\n", buf, filetype); 响应报头
    Rio_writen(fd, buf, strlen(buf)); 终止报头

    srcfd = open(filename, O_RDONLY, 0); 已只读方式打开，相应的文件
    srcp = mmap(0, filesize, PROT_READ, MAP_PRIVATE, srcfd, 0); srcp指向创建的存储器映射
    close(srcfd); 关闭打开的文件，这时候存储器映射已经完成，打开的文件描述已经不需要了
    Rio_writen(fd, srcp, filesize); 传输到客户端
    munmap(srcp, filesize); 避免内存泄漏，显示的释放存储器映射区域
}

```

```

void get_filetype(char *filename, char *filetype)
{
    if(strstr(filename, ".html"))
        strcpy(filetype, "text/html");
    else if (strstr(filename, ".gif"))
        strcpy(filetype, "image/gif");
    else if (strstr(filename, ".jpg"))
        strcpy(filetype, "image/jpeg");
    else
        strcpy(filetype, "text/plain");
}

```

⑥ 处理动态内容：serve_dynamic

```

void serve_dynamic(int fd, char *filename, char *cgiargs)
{
    char buf[MAXLINE], *emptylist[] = {NULL};

    sprintf(buf, "HTTP/1.0 200 OK\r\n");
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Server: Tiny Web Server\r\n");
    Rio_writen(fd, buf, strlen(buf)); 发送响应报头，表示成功

    if(Fork() == 0) 创建子进程运行
    {
        setenv("QUERY_STRING", cgiargs, 1); 使用传递的参数初始化环境变量，QUERY_STRING
        Dup2(fd, STDOUT_FILENO); 重定位输出，使得直接指向到客户端
        Execve(filename, emptylist, environ); 运行具体的程序，在cgi-bin目录下
    }
    Wait(NULL); 主程序等待子进程返回，并回收资源
}

```

⑦ 错误处理函数：clienterror

```
void clienterror(int fd, char *cause, char *errnum, char *shortmsg, char *longmsg)
{
    char buf[MAXLINE], body[MAXLINE];

    /* Build the http response body */
    sprintf(body, "<html><title>Tiny Error</title>");
    sprintf(body, "%s<body bgcolor=\"ffffff\">\r\n", body);
    sprintf(body, "%s%s: %s\r\n", body, errnum, shortmsg);
    sprintf(body, "%s<p>%s: %s\r\n", body, longmsg, cause);
    sprintf(body, "%s<hr><em>The Tiny Web server</em>\r\n", body);

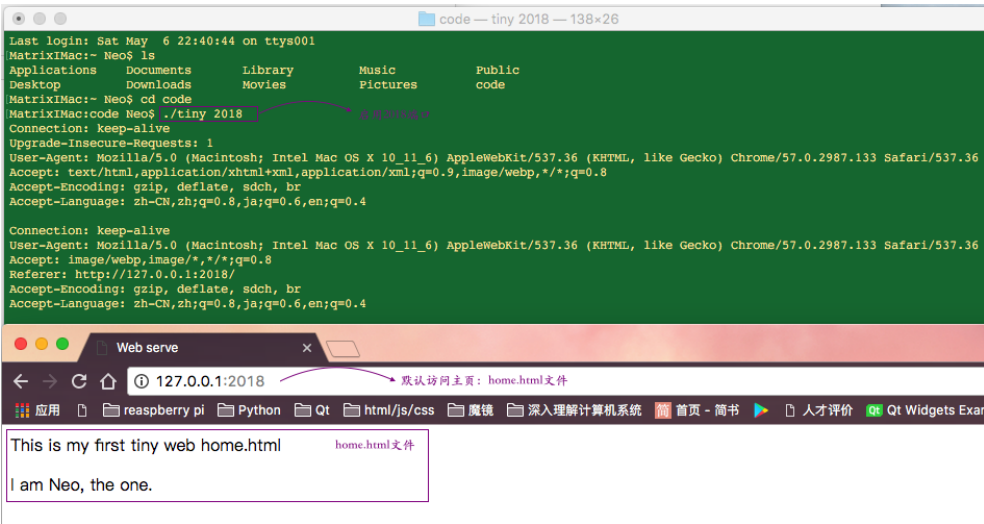
    /*printf the http response */
    sprintf(buf, "HTTP/1.0 %s %s \r\n", errnum, shortmsg);
    Rio_writen(fd, buf, strlen(buf));
    sprintf(buf, "Content-type: text/html\r\n");
    Rio_writen(fd, buf, strlen(buf));
    Rio_writen(fd, body, strlen(body));
}
```

创建相应的主体，显示的内容

响应行

响应头：type

我们来看看运行的效果，首先是解析动态内容，我们创建了一个主页：home.html文件，并且在服务端启用了



显示动态内容，我们使用之前的adder程序，将1243和12相加，并显示



2017年05月07日 完

"小礼物走一走，来简书关注我"

赞赏支持

还没有人赞赏，支持一下



唐鱼的学习探索 如果我像一般人一样读那么多书，我就跟他们一样愚蠢了。
总资产29 (约2.81元) 共写了10.4W字 获得530个赞 共463个粉丝

关注

ExpressVPN™ - Fast Premium VPN

Fast, Secure & Anonymous. 94 Countries. Easy Setup on any Device
Top-rated VPN

写下你的评论...

全部评论 0 只看作者

按时间倒序 按时间正序

被以下专题收入，发现更多相似内容

- 《深入理解计算...
- 程序员
- 网络

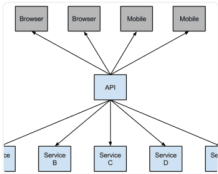
推荐阅读

Spring Cloud

Spring Cloud为开发人员提供了快速构建分布式系统中一些常见模式的工具（例如配置管理，服务发现，断路器，智...

卡卡罗2017 阅读 84,405 评论 14 赞 122

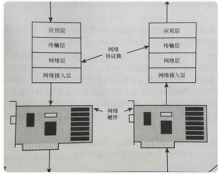
更多精彩内容>



计算机网络基础知识大总汇

谢谢大家喜欢我的文章!!! 我创建了一个技术公众号，为您提供一系列系统架构、数据结构、网络、C++、计算机底层等高...

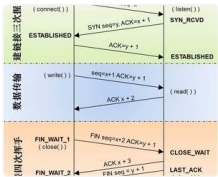
龙猫小爷 阅读 89,457 评论 40 赞 1,235



计算机基础相关知识

操作系统 进程和线程的区别进程是资源拥有的基本单位，线程是独立调度与独立运行的基本单位。进程是程序运行时的实例，每...

brucevanfdm 阅读 639 评论 1 赞 7



浅谈网络编程

引言 网络学习的核心内容就是网络协议的学习 网络协议：网络中进行数据交换而建立的规则、标准或者说是约定的集合因为不...

 _凉风_ 阅读 1,111 评论 9 赞 21

层	TCP/IP 五层模型	报文:		
	应用层	SMTP	FTP	TELNET
	传输层	传输协议:	TCP	
	网络层【IP层】	IP 数据报:	IP (
	数据链路层	帧:	网络接口层	
	物理层	以太网	令牌环	
	硬件【物理网络】			

Haskell 入门笔记(一)

一句话介绍：Haskell 是一门纯粹的函数式编程语言。安装Haskell 需要一个 Haskell 编译器。目...

 焉知非鱼 阅读 3,067 评论 9 赞 8