通信软件基础B

多线程基础

姓名|职称|地址|联系方式

《通信软件基础B》教学内容

- 第二部分：通信中的数据传输
  - Socket基础
  - 多线程基础
  - 通信协议基础
  - 通信软件开发过程

《通信软件基础B》教学内容

- 第二部分：通信中的数据传输
  - Socket基础
  - **多线程基础**
  - 通信协议基础
  - 通信软件开发过程

信息领域的发展概况

提供资源访问虚拟化操作(文件、硬件、网络等)

一切基于Web应用

Microsoft

Google

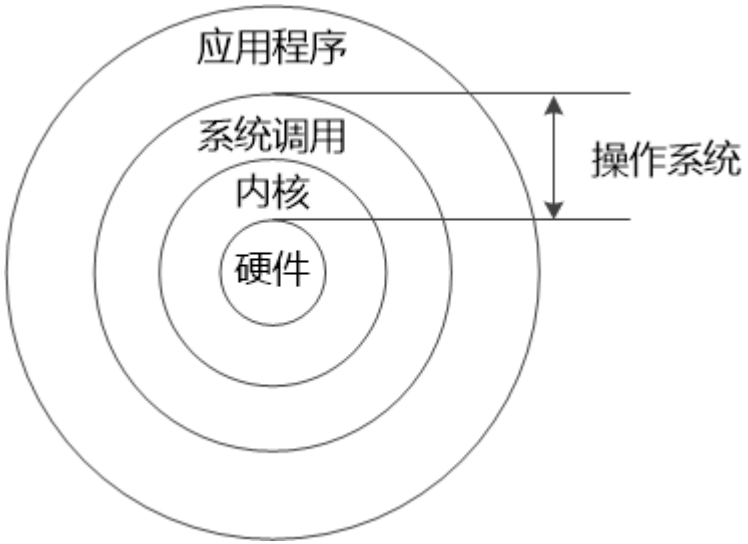
vmware

Chrome OS

Chromebook

●【操作系统】

- ●是***管理***计算机系统的***软、硬件资源***，使之正常运行，为用户***提供人机操作界面***的系统软件；
- ●是建立在裸机上的第一层软件；



- **内核**：负责管理硬件资源分配
  - ✓ 让CPU可以运算；
  - ✓ 让硬盘可以读写数据等；
- 如果只有内核
  - ✓ 只能让计算机硬件运行;
  - ✓ 不能有任何功能;
- **系统调用**是操作系统提供给开发者的调用接口，从而可以开发应用程序，丰富计算机系统的功能；
- 任何硬件都不会默认就被操作系统控制，需要开发商根据操作系统提供的接口开发**驱动程序**；

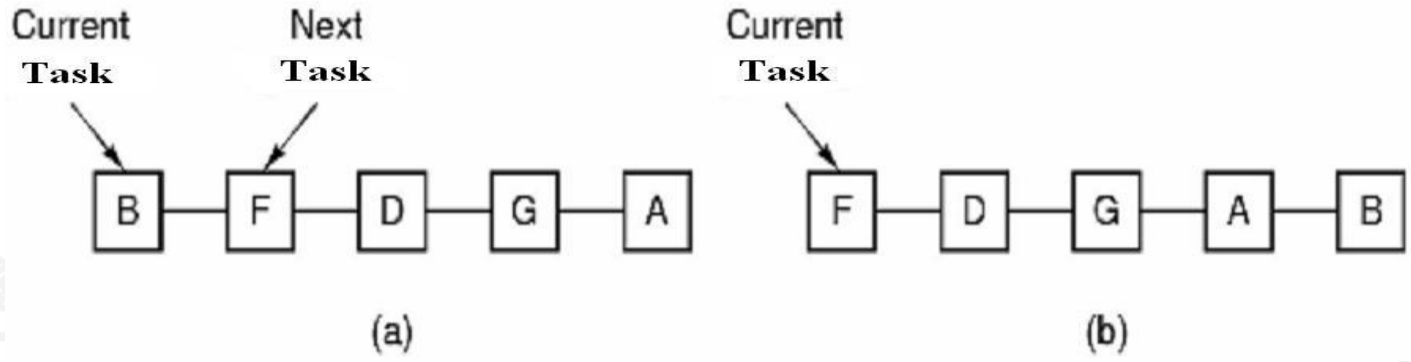
- 【进程的基本概念】
  - 什么是进程？
    - 一个进程是一个程序的一次执行的过程；
    - 进程是程序执行时的一个实例；
  - 进程的两个重要特性
    - 独立性
      - 是系统申请系统资源和调度的基本单位；
      - 是系统中独立存在的实体，可以拥有自己独立的资源；
        - 比如CPU 时间、存储器、文件和设备描述符等；
      - 没有经过进程本身允许，其他进程不能访问到这些资源；

- 【进程的基本概念】
  - 进程的两个重要特性
    - 动态性（进程与程序的区别）
      - 程序只是一个静态的指令集合
        - 是保存在磁盘上的可执行代码和数据的集合；
      - 进程是一个正在系统中活动的指令集合
        - 在进程中加入了时间的概念；
        - 是程序执行的过程，包括动态创建、调度和消亡的整个过程；
        - 进程具有自己的生命周期和各种不同的状态；
        - 一个程序可以对应多个进程；

●【进程的基本概念】

●时间片（time slice）

- ●在时间片轮转算法中，将所有的就绪任务按照FCFS（First Come First Served）原则，排成一个队列；
- ● 每次调度时将处理器分派给队首任务，让其执行一小段CPU时间（时间片，time slice）；
- ● 在一个时间片结束时，如任务还没有执行完的话，将发生时钟中断，在时钟中断中，调度程序将暂停当前任务的执行，并将其送到就绪队列的末尾，然后执行当前的队首任务；

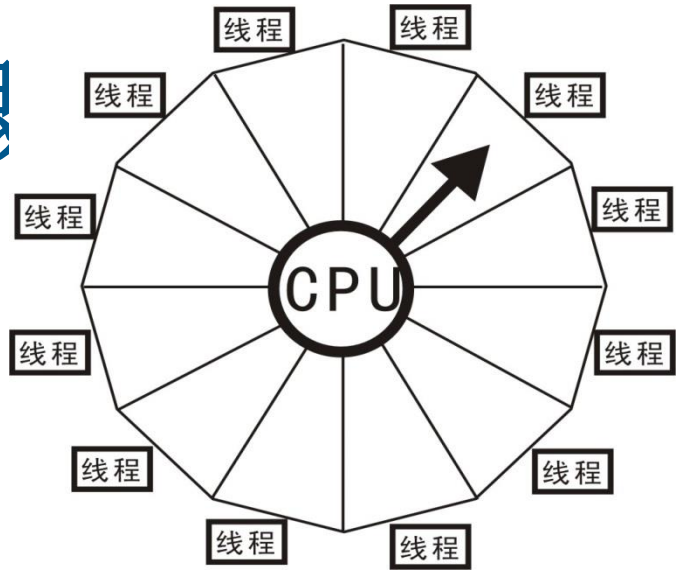


- 【进程的基本概念】

网络编程相关的基本概念

- 网络编程与进程通信
  - 进程与线程的基本概念
    - 进程是处于运行过程中的程序实例，是操作系统调度和分配资源的基本单位。
    - 一个进程实体由程序代码、数据和进程控制块三部分构成。
    - 各种计算机应用程序在运行时，都以进程的形式存在。网络应用程序也不例外。

网络编程相关的基本概



- 网络编程与进程通信
  - 进程与线程的基本概念
    - Windows系统不但支持多进程，还支持多线程。
    - 当创建一个进程时，系统会自动创建它的第一个线程，称为主线程。然后，该线程可以创建其他的线程，而这些线程又能创建更多的线程。

网间进程的标识

- IP地址--在网络中标识主机
  - IP地址(网络号+主机号)
- 传输层端口--标识进程
  - 端口是TCP/IP协议族中，应用层进程与传输层协议实体间的通信接口；
  - 从实现的角度讲，端口是一种抽象的软件机制，包括一些数据结构和I/O缓冲区；
  - 应用程序需要与端口建立**绑定关系**；
  - 每个端口都拥有一个叫作端口号（port number）的整数型标识符；

一个比喻

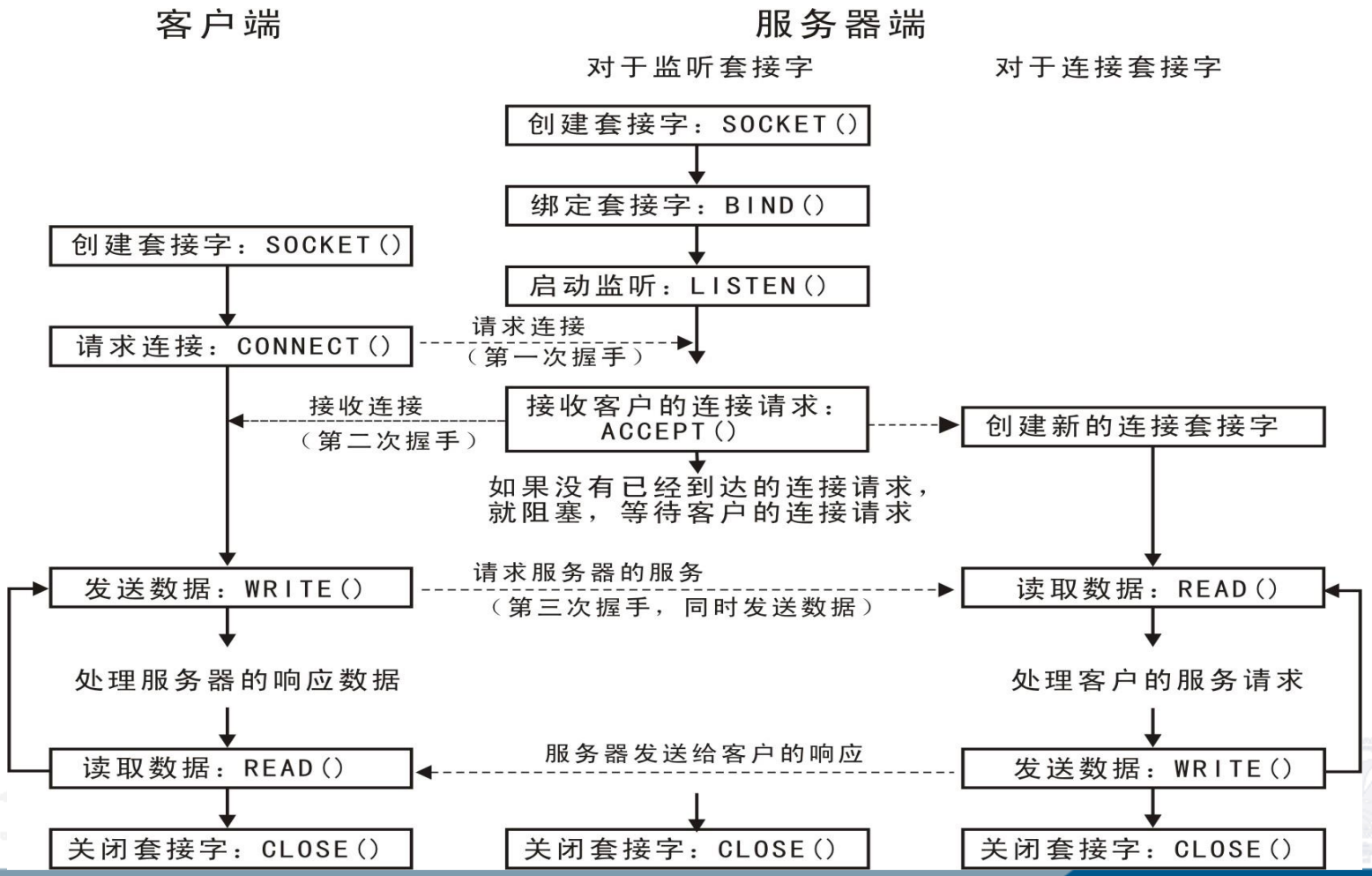
- 如果把IP数据包的投递过程看成是给远方的一位朋友寄一封信，那么
- IP地址就是这位朋友的所在位置（依靠此信息进行路由）
- 端口号就是这位朋友的名字（依靠这个信息最终把这封信交付给这位收信者）

三类网络编程

- 基于TCP/IP协议栈的网络编程
  - 基于TCP/IP协议栈的网络编程是最经典的网络编程方式，主要是使用各种编程语言，利用操作系统提供的套接字网络编程接口，直接开发各种网络应用程序。
- 基于WWW应用的网络编程
  - JAVA，HTML，ASP，PHP
- 基于.NET框架的Web Services网络编程

面向连接的套接字编程

- 套接字的工作过程



无连接的套接字编程

- 无连接的套接字编程的两种模式
  - 1. 对等模式

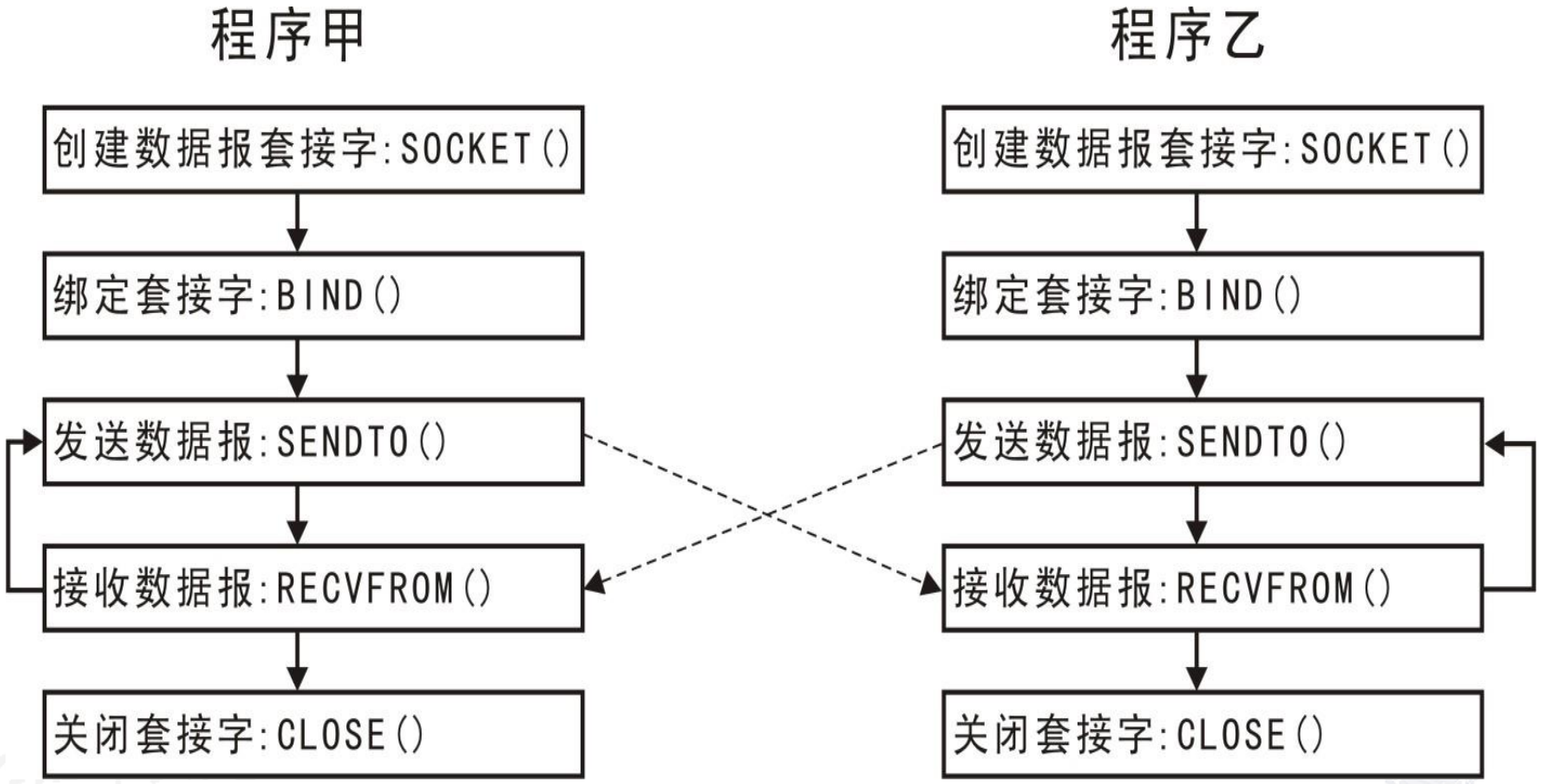


图2.10 对等模式的数据报套接字的编程模型

无连接的套接字编程

- 无连接的套接字编程的两种模式
  - 2. 客户/服务器模式

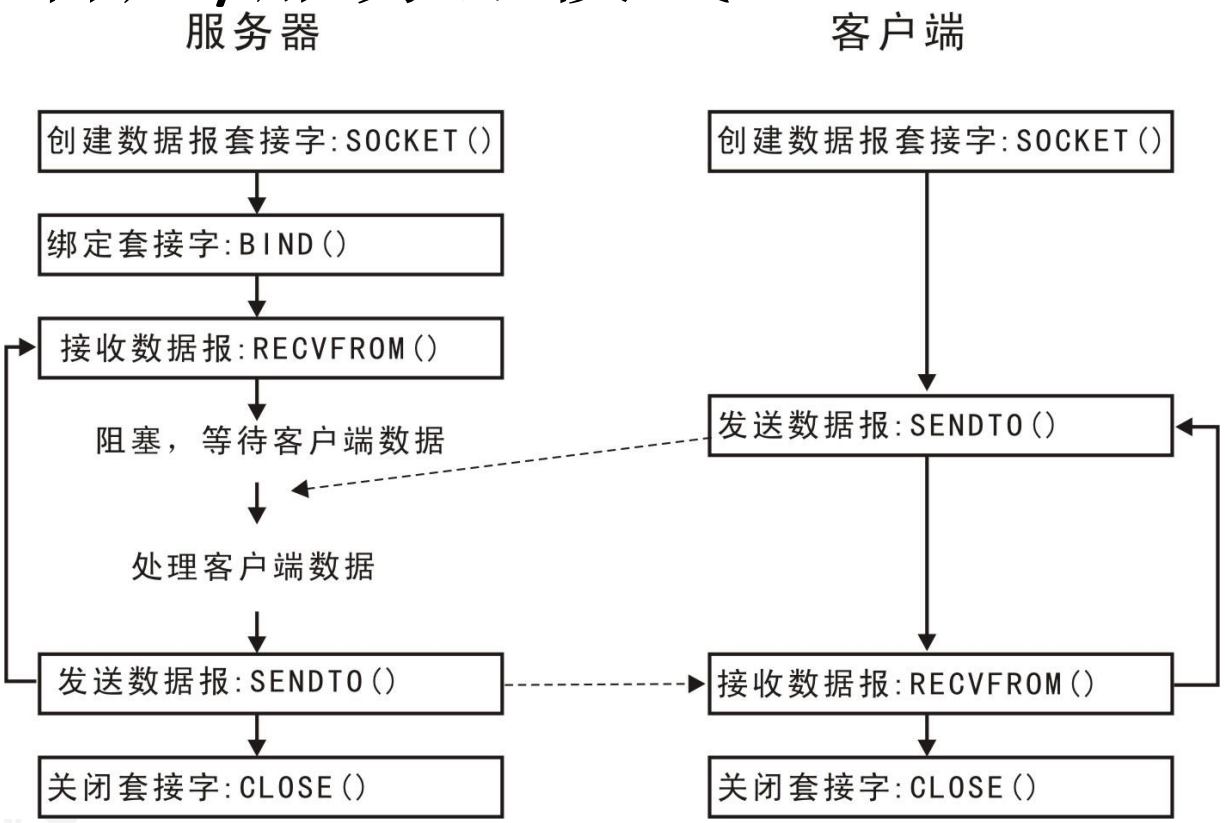
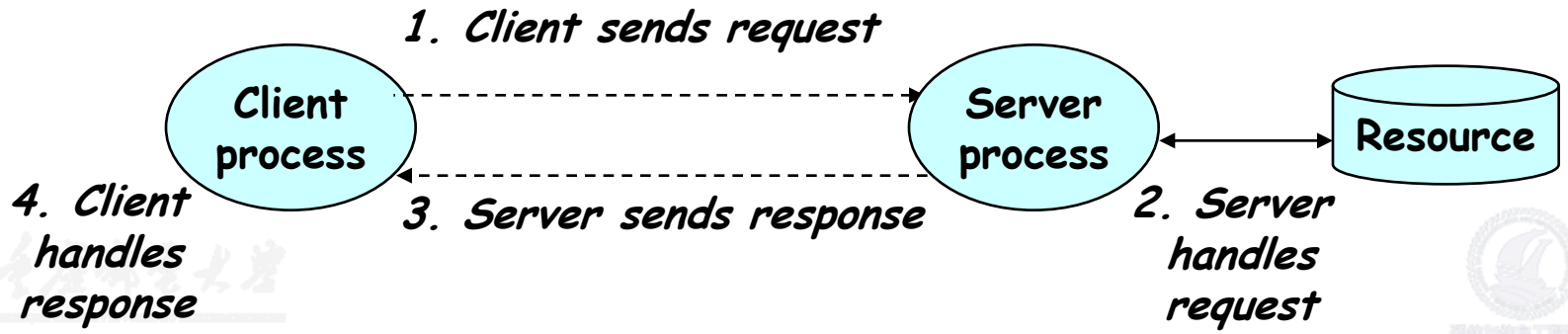


图2.11 C/S模式的数据报套接字的编程模型

C/S模式

- Client asks (*request*) – server provides (*response*)
- Typically: single server - multiple clients
- The server does not need to know *anything* about the client
  - even that it exists
- The client should always know *something* about the server
  - at least where it is located



C/S模式

- C/S模式，因特网上应用程序最常用的通信模式。
- 客户方采取的是主动请求方式，其工作过程是：

(1) 打开一通信通道，并连接到服务器所在主机的特定监听端口。

(2)向服务器发送请求报文，等待并接收应答；继续提出请求，与服务器的会话按照应用协议进行。

(3)请求结束后，关闭通信通道并终止。

C/S交互模式

- 服务器的工作过程：

(1) 打开一通信通道，并告知服务器所在的主机，它愿意在某一公认的地址上（熟知端口，如FTP为21）接收客户请求。

(2) 等待客户的请求到达该端口。

(3) 服务器接收到服务请求，处理该请求并发送应答信号。

(4) 返回第二步，等待并处理另一客户请求。

(5) 在特定的情况下，关闭服务器。

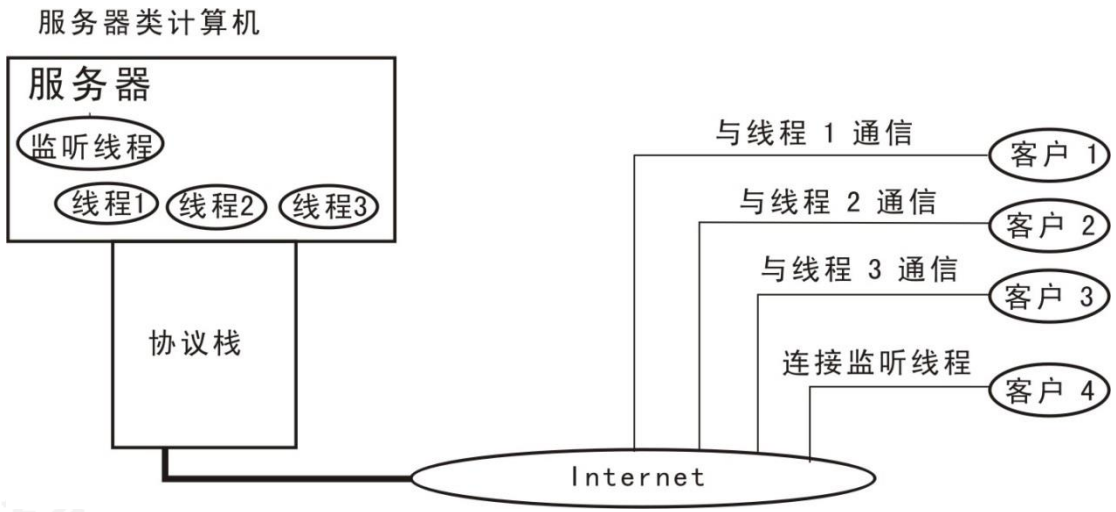
服务器的工作方式

若有多个客户端同时请求，服务器如何处理？

- 循环方式(iterative mode)
  - 在计算机中一次只运行一个服务器进程。当有多个客户进程请求服务时，服务器进程就按请求的先后顺序依次做出响应。
- 并发方式(concurrent mode)
  - 在计算机中同时运行多个服务器进程，而每一个服务器进程都对某个特定的客户进程做出响应。

服务端的并发性

- 并发性是C/S模式的基础，并发允许多个客户获得同一种服务，而不必等待服务器完成对上一个请求的处理。这样才能很好地同时为多个客户提供服务。
- 操作系统支持并发性程序开发
  - Unix系统: fork()，创建新进程
  - Windows系统: CreateThread()，创建新线程

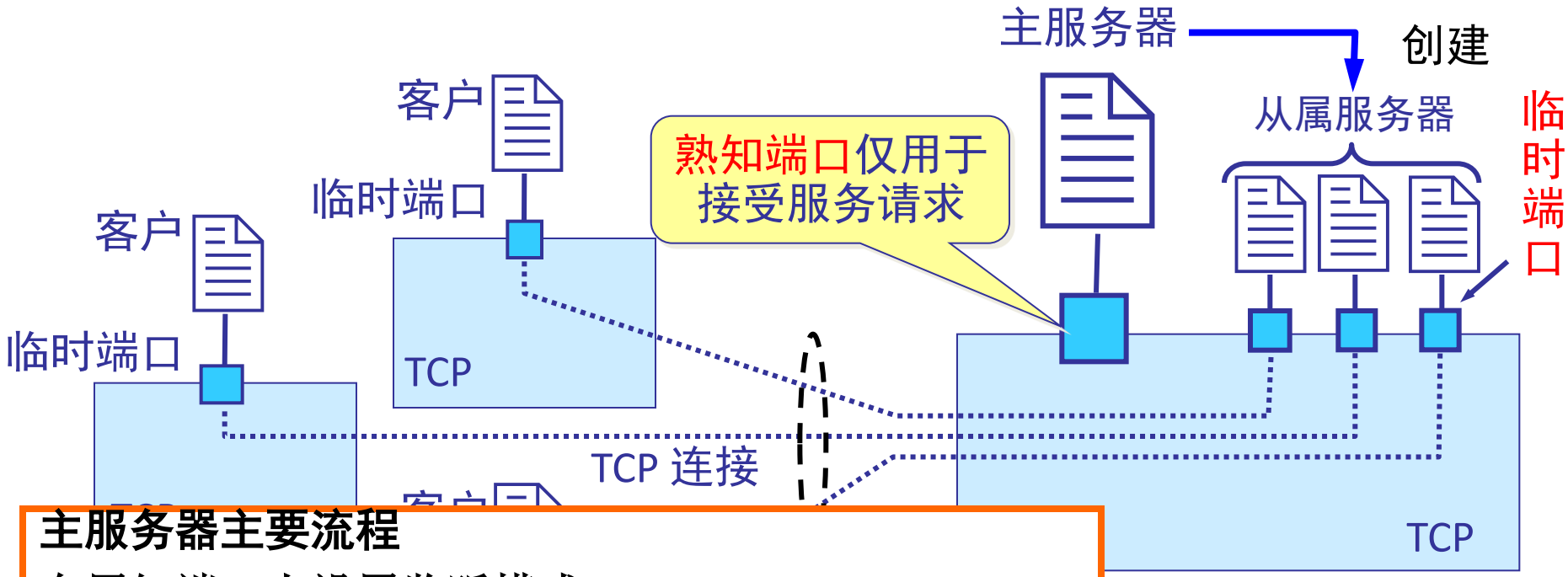


服务器的设计方式

- 服务器设计方式由采用的传输层协议和工作方式决定；
  - (TCP/UDP)+(循环/并发)

- 四种设计方式：
  - **面向连接的并发**
  - 面向连接的循环
  - 无连接的并发
  - **无连接的循环**

面向连接的并发服务器



主服务器主要流程

在周知端口上设置监听模式

While(true){

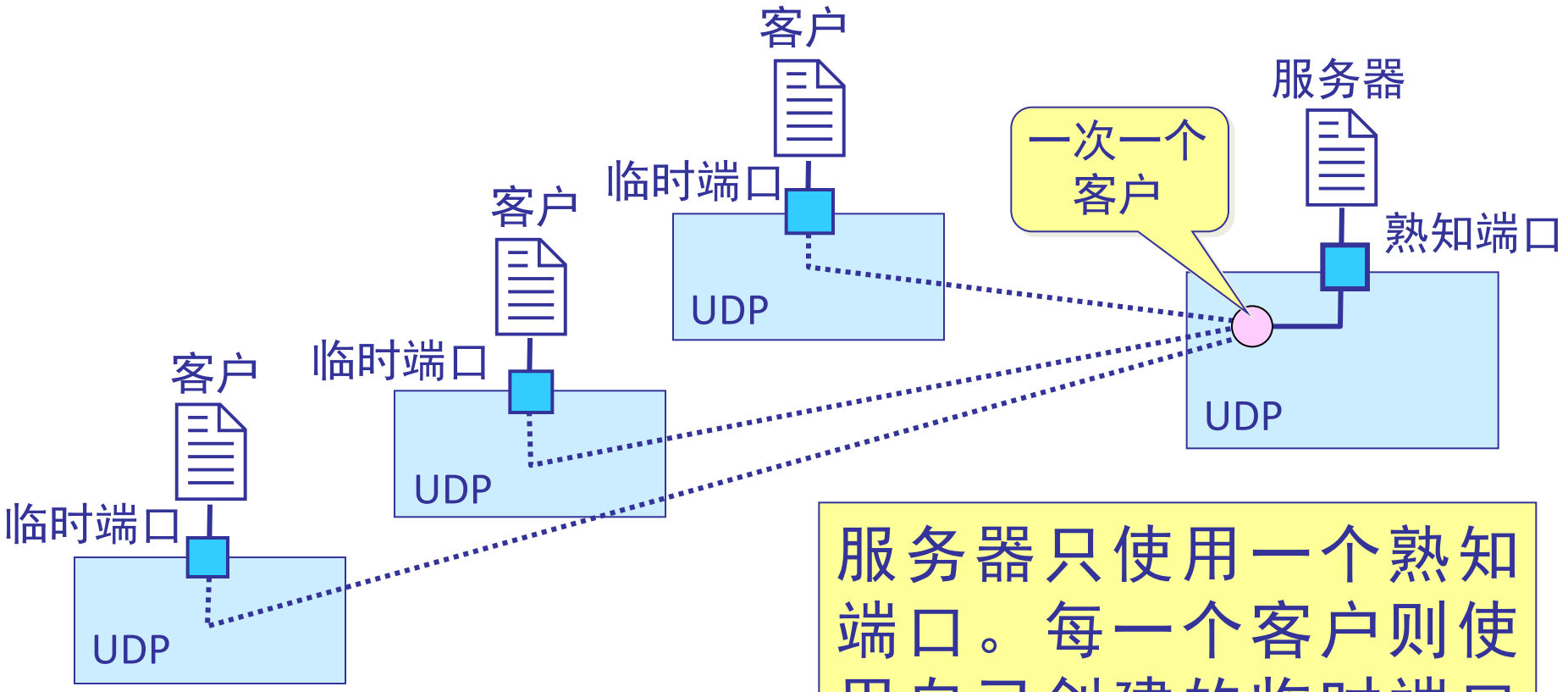
若在周知端口上有客户端的TCP连接请求;

进行三次握手建立TCP连接;

创建从属服务器(用临时端口与客户端进行通信);

}

无连接循环服务器



服务器只使用一个熟知端口。每一个客户则使用自己创建的临时端口（端口号自己设定）。

客户 / 服务器交互模式

- 容易混淆的术语
  - 客户与用户
    - “客户”(client)和服务器都指的是应用进程，即计算机软件。
    - “用户”(user)指的是使用计算机的人。

客户 / 服务器交互模式

- 容易混淆的术语
  - 图示区别

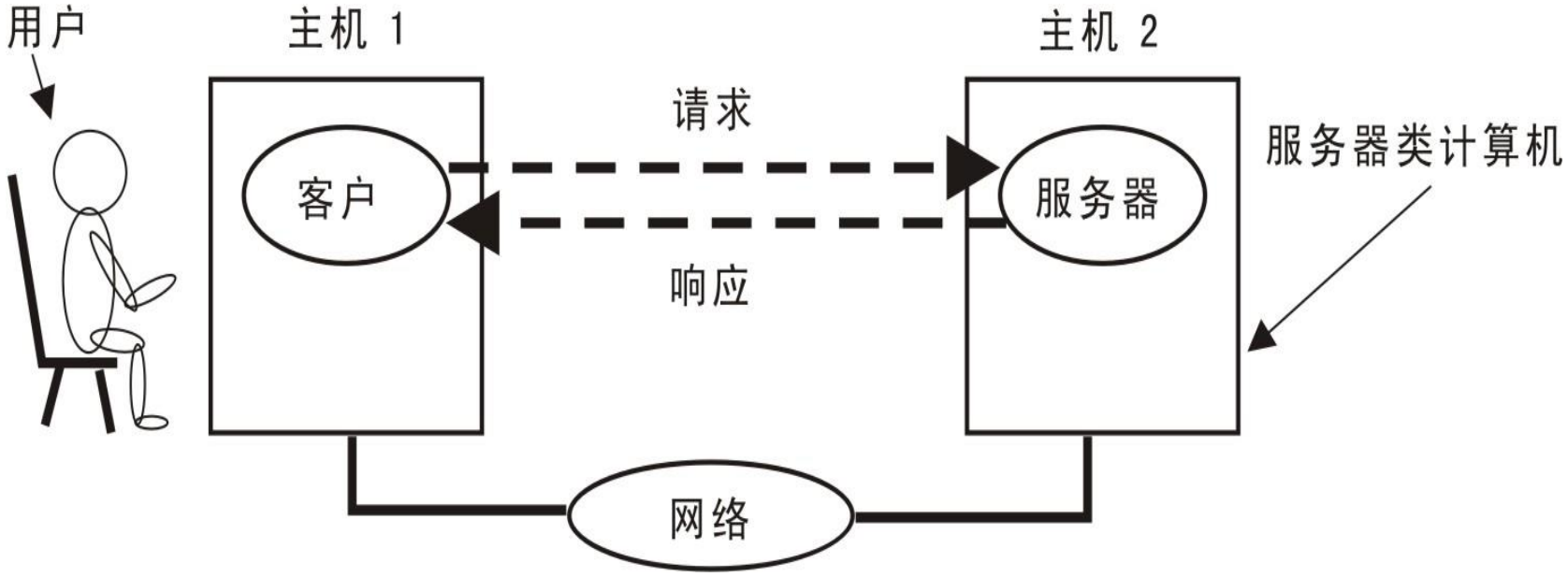


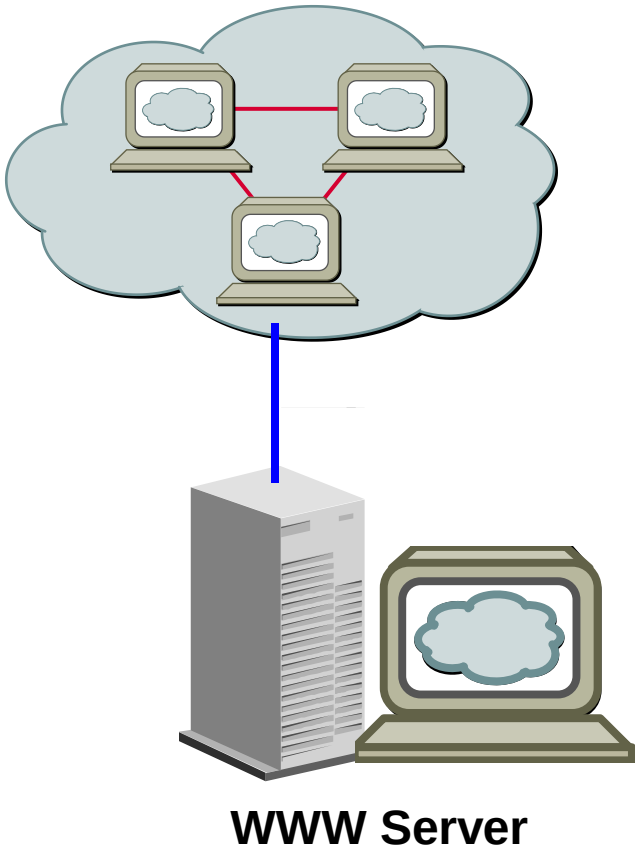
图1.10 用户、客户、服务器、服务器类计算机

客户 / 服务器交互模式

- 错综复杂的客户 / 服务器交互
  - C/S模式中，存在着三种一个与多个的关系：
    - （1）一个服务器同时为多个客户服务；
    - （2）一个用户的计算机上同时运行多个连接不同服务器的客户
    - （3）一个服务器类的计算机同时运行多个服务器；

工作模式的变迁：B/S模式

- 特点：
  - 改进信息表示方式
  - 三层构架，将商业逻辑放到Server
  - 减轻安装维护工作



面向连接的套接字编程

- 面向连接的套接字编程实例
  - 实例的功能
    - 服务器对来访的客户计数，并向客户报告这个计数值。
    - 客户建立与服务器的一个连接并等待它的输出。
    - 每当连接请求到达时，服务器生成一个可打印的ASCII串信息，将它在连接上发回，然后关闭连接;
    - 客户显示收到的信息，然后退出。

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 关于阻塞的问题

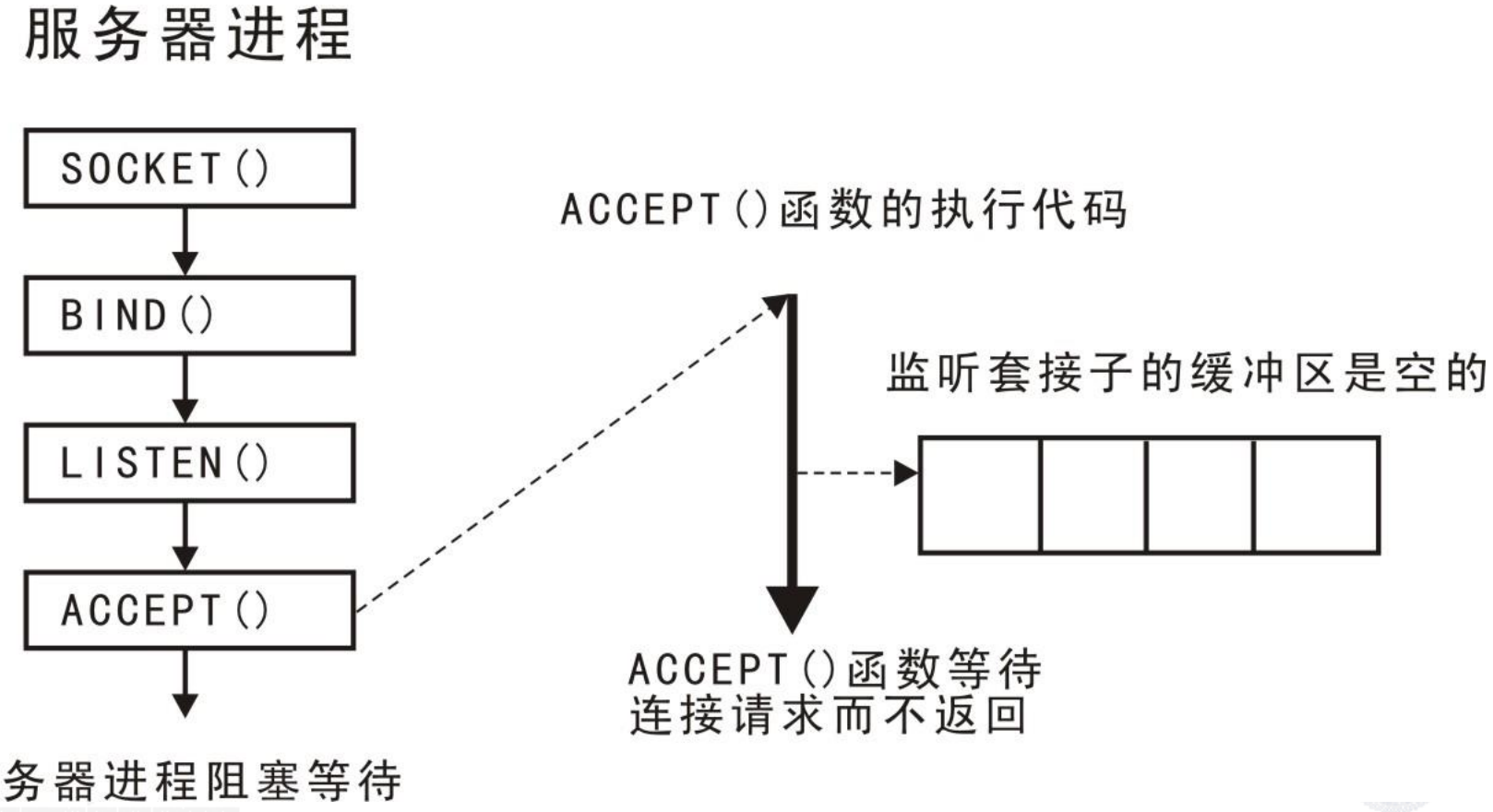


图2.7 服务器进程因调用ACCEPT()而被阻塞

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 什么是阻塞
    - 阻塞是指一个进程执行了一个函数或者系统调用，该函数由于某种原因不能立即完成，因而不能返回调用它的进程，导致进程受控于这个函数而处于等待的状态，进程的这种状态称为阻塞。

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 什么是阻塞

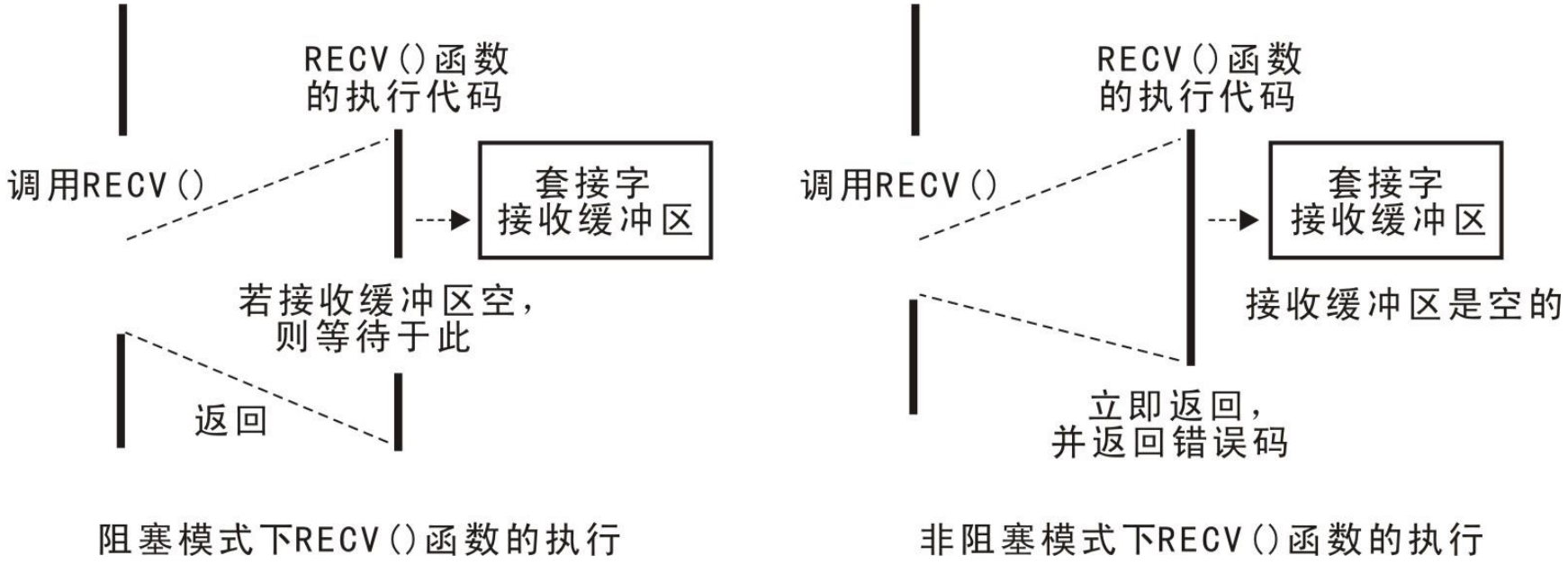


图2.8 RECV()函数的两种执行方式

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 能引起阻塞的套接字调用
    - 在Berkeley套接字网络编程接口的模型中，套接字的默认行为是阻塞的，具体地说，在一定情况下，有多个操作套接字的系统调用会引起进程阻塞。
      - （1）ACCEPT()
      - （2）READ()、RECV()和READFORM()
      - （3）WRITE()、SEND()和SENDTO()
      - （4）CONNECT()
      - （5）SELECT()
      - （6）CLOSESOCKET()什么是阻塞

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题
    - 采用阻塞工作模式的单进程服务器是不能很好地同时为多个客户服务

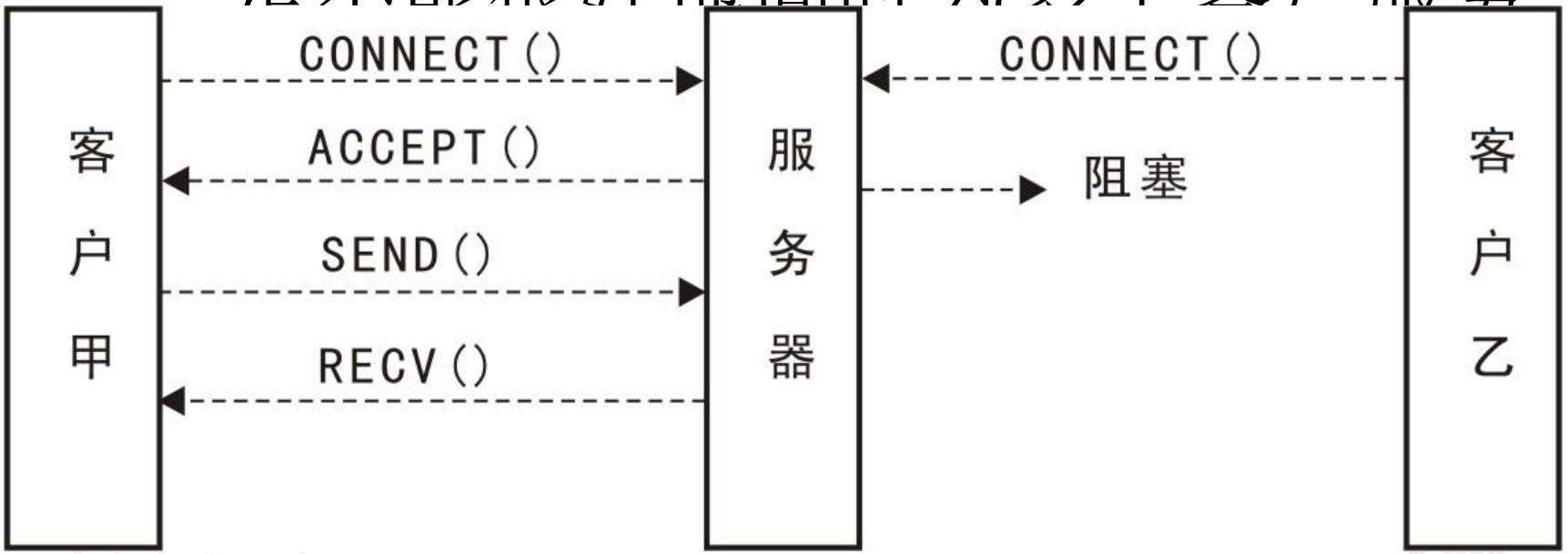


图2.9 采用阻塞工作模式的服务器不能很好地为多个客户服务

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题
    - 利用UNIX操作系统的FORK()系统调用，编制多进程并发执行的服务器程序。可以创建子进程。对于每一个客户端，用一个专门的进程为它服务，通过进程的并发执行，来实现对多个客户的并发服务。基本的编程框架是：

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题

```
父进程代码
If ((pid = FORK()) == 0) {
.....
子进程代码
.....
} else if (pid<0) {
报错信息
}
父进程代码
```

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题
    - 举例：

```
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <arpa/inet.h>

void main(int argc, char** argv)
{
int listenfd,clientfd,pid;
struct sockaddr_in ssockaddr, csockaddr;
char buffer[1024];
int addrlen,n;
```

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题

```
/* 创建监听套接字 */
listenfd = socket(AF_INET,SOCK_STREAM,0);
if  (listenfd < 0) {
fprintf(stderr, "socket error!\n");
exit(1);
}
```

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题

```
/* 为监听套接字绑定网络地址 */
memset(&ssockaddr,0,sizeof(struct sockaddr_in));
ssockaddr.sin_family = AF_INET;
ssockaddr.sin_addr.s_addr = htonl(INADDR_ANY);
ssockaddr.sin_port = htons(8080);
if  (bind(listenfd,&ssockaddr,sizeof(struct sockaddr_in)) < 0) {
fprintf(stderr, "bind error!\n");
exit(2);
}
```

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题

```
/* 启动套接字的监听 */
listen(listenfd,5);
addrlen = sizeof(sockaddr);

/* 服务器进入循环，接受并处理来自不同客户端的连接请求 */
while (1) {
clientfd = accept(listenfd,(sockaddr*)(&csockaddr),&addrlen);
/* accept 调用返回时，表明有客户端请求连接，创建子进程处理连
   接*/
If ((pid = FORK()) == 0) {
```

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题

```
/* 显示客户端的网络地址 */
printf("Client Addr: %s%d\n",inet_ntoa(csockaddr.sin_addr),
ntohs(csockaddr.sin_port));
/* 读取客户端发送来的数据，在将它们返回到客户端 */
while ((n = read(clientfd,buffer,1024)) > 0) {
buffer[n] = 0;
printf("Client Send: %s",buffer);
write( clientfd, buffer, n);
}
if (n < 0) {
fprintf( stderr, "read error!\n");
exit(3);
}
```

面向连接的套接字编程

- 进程的阻塞问题和对策
  - 阻塞工作模式带来的问题

```
/* 通信完毕，关闭与这个客户连接的套接字 */
printf("clent %s closed!\n", inet_ntoa(csockaddr.sin_addr));
close(clientfd);
it(1);
} else if (pid < 0) printf("fork failed!\n");
close(clientfd);
}
close(listenfd);      /* 关闭监听套接字 */
}
```

- CreateThread:
  - 提供操作系统级别的创建线程的操作，且仅限于工作者线程。线程函数中不调用MFC和RTL的函数时，可以用CreateThread，其它情况不要轻易使用，以免内存泄漏。
  - 但它没有考虑：
    （1）C Runtime库里面有一些函数使用了全局量，如果使用CreateThread 的情况下使用这些C++ 运行库的函数，就会出现不安全的问题。而 _beginthreadex 为这些全局变量做了处理（典型的例子是strtok函数）
    （2）MFC也需要知道新线程的创建，也需要做一些初始化工作。

线程函数原型介绍

- CreateThread —— creates a worker thread

```
HANDLE CreateThread(
  LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
  DWORD dwStackSize,                        // initial stack size
  LPTHREAD_START_ROUTINE lpStartAddress,    // thread function
  LPVOID lpParameter,                       // thread argument
  DWORD dwCreationFlags,                    // creation option
  LPDWORD lpThreadId                        // thread identifier
);
```

[举例]

实例练习：

- 编程实现一个简单的聊天程序，实现两台计算机间的信息自由交互；

- 不限操作系统
- 不限语言

Questions & Answers

谢谢您！ Thank you！