第11章 综合练习
一个小型远程访问FTP服务系统

实验目的

- 加深对进程、线程、进程互斥、同步、通信、文件系统及网络编程等概念的理解，能综合运用所学知识，并用来分析和解决有关理论问题和实际问题
- 理解基于客户/服务器计算模型，掌握支持并发用户访问的分布式软件系统的设计方法及核心技术
- 实现一个小型远程访问FTP服务系统，本实验需要综合运用前面几章介绍的概念、技术和方法，且有一定的程序量（约4000行），可作为操作系统的一个综合性实验或课程设计题。

主要内容

- 背景知识
  - 客户/服务器计算模型
  - 中间件
  - FTP技术简介
- 实验内容
  - 综合实验功能设计
  - 综合实验解决方案

客户/服务器计算模型

- 客户/服务器模型是一种基于局域网或广域网的分布式系统
  - 提供数据和服务的计算机称为服务器（Server），
  - 向服务器提出数据请求和服务要求的计算机称为客户机(Client)，
  - 这样的模型称为客户/服务器模型。
- 万维网基本上是一个客户/服务器模型。
  - 客户机通常为个人机或工作站，为终端用户提供友好的图形界面；
  - web站点是服务器，为用户提供各种数据和服务，最常见的是数据库服务、邮件服务、文件服务等。

客户/服务器计算模型

- 传统客户/服务器体系结构包括两层：
  - 客户层和服务器层
- 3层结构模型：
  - 用户层机器(客户)、中间层服务器(应用服务)及后端服务器(数据资源)。
  - 用户机器一般是瘦型客户机;
  - 中间层机器基本上是位于客户和很多后端数据库之间的网关;
  - 中间层机器介于两个层次之间而可充当桌面应用程序和后端应用程序之间的网关;
  - 中间层机器和后端服务器之间的交互也遵从客户/服务器模型，因此，中间层同时充当着客户机和服务器。

主要内容

- 背景知识
  - 客户/服务器计算模型
  - 中间件
  - FTP技术简介
- 实验内容
  - 综合实验功能设计
  - 综合实验解决方案

中间件

❖中间件概念

- 客户/服务器计算模型解决了单个系统的互通互联及独立的应用系统的开发和使用，存在问题：互操作性实现困难。
- 分布式计算的标准化。

❖解决方案

- 在上层应用程序和下层通信软件及操作系统之间使用标准编程接口和协议，建立支持在异构网络、异构计算机和不同操作系统中访问系统资源的工具，这种标准化的接口、协议和工具被称为“中间件”**(middleware)**。

中间件

❖中间件体系结构

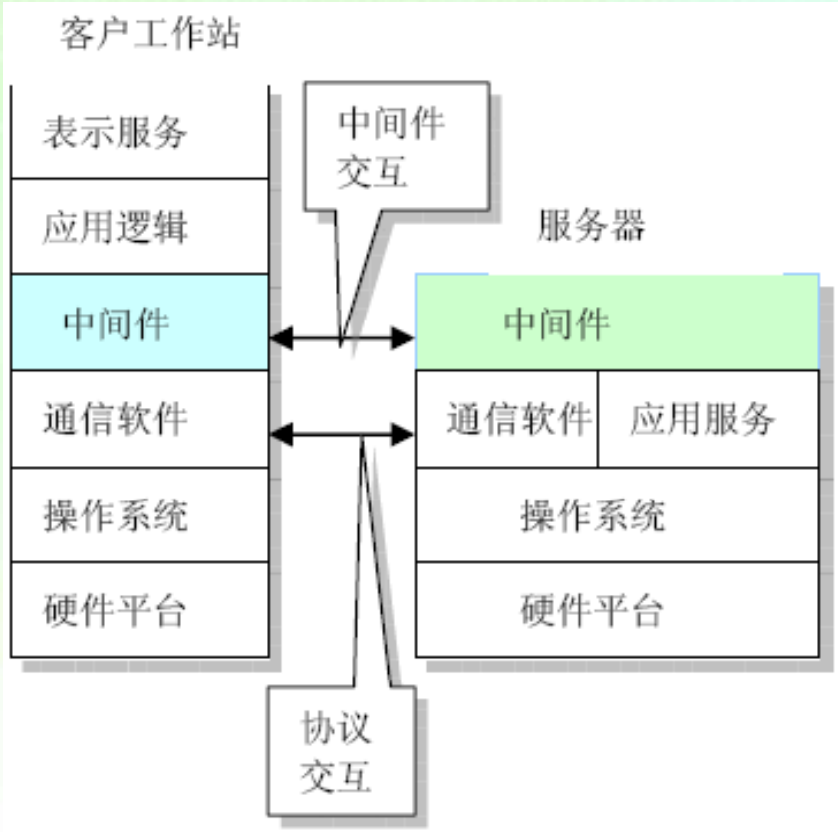


图 11-1 中间件在 C/S 结构中的作用

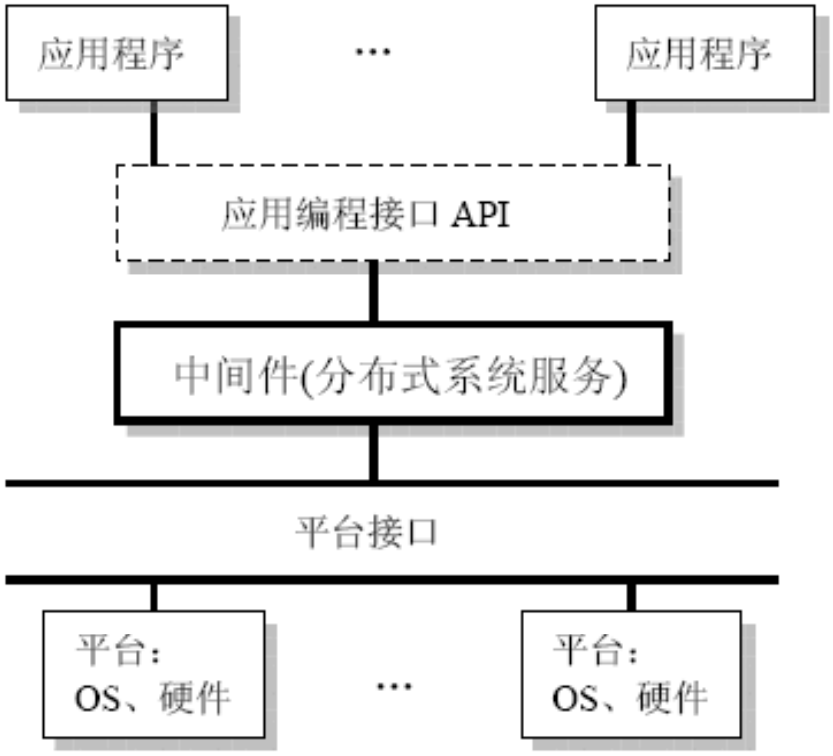


图 11-2 中间件的逻辑视图

主要内容

- 背景知识
  - 客户/服务器计算模型
  - 中间件
  - **FTP技术简介**
- 实验内容
  - 综合实验功能设计
  - 综合实验解决方案

FTP技术简介

❖FTP

- 文件传输协议FTP (File Transfer Protocol)，是TCP/IP协议集的协议之一，用于Internet上控制文件的双向传输。
- FTP也是一个分布式应用程序，Internet用户可以通过自己的个人计算机与分散于分布式网络环境下、运行FTP协议的任一服务器相连，进而访问服务器上的大量程序和信息。

❖FTP的主要作用

- 让用户通过FTP协议，连接到一个运行FTP服务器程序的远程计算机，察看远程计算机上的现有文件，然后将文件从远程计算机拷到本地计算机（下载（download）文件），或把本地计算机的文件传送到远程计算机去（上传（upload）文件）。
- 在TCP/IP协议集中，FTP标准命令TCP端口号为21，port方式数据端口为20。

FTP技术简介

❖FTP

- FTP也是一个客户/服务器系统。
- 客户通过一个客户机程序连接至在远程计算机上运行的服务器程序。依照 FTP协议提供服务，进行文件传送的计算机是FTP服务器；而连接FTP服务器、遵循FTP协议与服务器传送文件的计算机是FTP客户端。
- 在通过FTP从远程计算机拷贝文件时，实际上需要启动两个程序：
  - ✓一个是FTP客户端上的FTP客户程序，用于向FTP服务器提出拷贝文件的请求；
  - ✓另一个是FTP服务器上的FTP服务程序，响应客户请求并将指定文件传送到FTP客户计算机。

FTP技术简介

❖FTP

- 使用FTP协议实现文件传输时，FTP服务过程与FTP服务器及FTP客户机所处的位置、联接的方式、操作系统类型等无关。
- 使用FTP访问远程服务器的命令格式如下:
  - ✓ftp://客户名:密码@FTP服务器IP或域名：FTP命令端口/路径/文件名
- 从传输角度，FTP有两种传输模式，
  - ✓即ASCII传输模式和二进制数据传输模式:

主要内容

- 背景知识
  - 客户/服务器计算模型
  - 中间件
  - FTP技术简介
- 实验内容
  - 综合实验功能设计
  - 综合实验解决方案

综合实验功能设计

❖ 实验说明

- 借鉴Internet上FTP服务的功能设计，采用客户/服务器模型，实现一个支持远程文件传输与共享的小型FTP服务系统。
- 与标准FTP服务软件类似，该小型远程访问FTP服务系统需要由服务器端软件和客户端软件两部分组成。其中服务器软件在指定端口接受客户连接请求，并根据客户请求执行相应处理。客户端程序为客户提供访问此小型远程访问FTP服务系统的交互界面。

综合实验功能设计

- 该FTP服务系统具备以下几个基本功能。
  - 1）基于socket的客户/服务器通信模式。服务器在指定TCP端口sport启动FTP模拟系统的服务器软件，等待并接收客户请求。客户端通过TCP协议向服务器的端口sport发送连接请求，若通过服务器端的接入控制，将通过该连接与服务器实现交互。
  - 2）远程登录功能。为支持身份验证，服务器端软件还需集成Linux系统的客户管理功能，对客户身份信息予以验证。客户在发送连接请求时，必须提供FTP服务器软件所驻留的Linux主机上客户名及密码。客户发送服务请求的命令格式如下:
    - ftps://客户名:密码@FTP服务器IP或域名：FTP命令端口

综合实验功能设计

- 该FTP服务系统具备以下几个基本功能。
  - 3）并发执行及管理功能。为提高系统运行效率，服务器软件将采用多线程技术响应客户请求。当客户请求通过身份验证后，服务器将创建一个新线程来响应客户请求，为客户提供服务。
  - 4）活动客户计数功能。站点计数为客户提供统计当前系统中的活动客户个数的功能。当客户通过身份验证后，活动客户计数器加1。当客户断开连接后，活动客户计数器减1。

综合实验功能设计

- 该FTP服务系统具备以下几个基本功能。
  - 5）文件管理功能。客户请求被调度后，支持客户执行以下操作。A）在服务器端执行基本文件操作，包括创建/删除目录（mkdir/rmdir）、切换目录（cd）、查看当前目录下的所有文件（ls）；B）在客户端执行基本文件操作，包括创建/删除目录（lmkdir/lrmdir）、切换目录（lcd）、查看当前目录下的所有文件（ldir）；C）设定文件传输模式，包括文本模式（ascii）或二进制模式（bin）。D）文件传输功能，包括上传/下载文件（upload/download）到指定目录。

主要内容

- 背景知识
  - 客户/服务器计算模型
  - 中间件
  - FTP技术简介
- 实验内容
  - 综合实验功能设计
  - 综合实验解决方案

综合实验解决方案

- 服务器端接收客户请求的套接字结构
- 客户端发送套接字连接请求的核心代码
- 与线程处理相关的核心函数
- 接收客户请求与实现客户会话的线程
- 文件管理
- 套接字通信

综合实验解决方案

❖服务器端接收客户请求的套接字结构

- 启动服务函数

```
SOCKET FTP_RunServer(int nPort){
        SOCKET nServerSocket = INVALID_SOCKET;
        struct sockaddr_in tAddress;
        nServerSocket = socket(AF_INET,SOCK_STREAM,0);
        if (nServerSocket == INVALID_SOCKET){
                return INVALID_SOCKET;
        }
```

综合实验解决方案

❖服务器端接收客户请求的套接字结构

- 启动服务函数

```
tAddress.sin_port = htons((unsigned short)nPort);
tAddress.sin_family = AF_INET;
tAddress.sin_addr.s_addr = htonl(INADDR_ANY);
if (bind(nServerSocket,(const struct sockaddr*)&tAddress,sizeof(struct sockaddr_in)) ==
INVALID_SOCKET){
        return -1;
}
```

综合实验解决方案

❖服务器端接收客户请求的套接字结构

- 启动服务函数

```
    if (listen(nServerSocket,SOMAXCONN) == INVALID_SOCKET){
        return INVALID_SOCKET;
    }
    return nServerSocket;
}
```

综合实验解决方案

❖服务器端接收客户请求的套接字结构

➢ 接收客户请求函数

```
SOCKET FTP_Accept(SOCKET nSocket, in_addr_t *ptClientIP){
        struct sockaddr_in tAddr_in;
        socklen_t nLength = sizeof(struct sockaddr_in);
        SOCKET nAccSocket = accept(nSocket,(struct sockaddr*)&tAddr_in,&nLength);
        if (nAccSocket < 0){
                return INVALID_SOCKET;
        }
        if (ptClientIP)*ptClientIP = tAddr_in.sin_addr.s_addr;
        return nAccSocket;
}
```

综合实验解决方案

- 服务器端接收客户请求的套接字结构
- 客户端发送套接字连接请求的核心代码
- 与线程处理相关的核心函数
- 接收客户请求与实现客户会话的线程
- 文件管理
- 套接字通信

综合实验解决方案

❖客户端发送套接字连接请求的核心代码

- 客户端链接函数

```
SOCKET FTP_Connect(in_addr_t nConnectTo, in_port_t nPort){
        SOCKET nClientSocket;
        struct sockaddr_in tServerAddress;
        nClientSocket = socket(AF_INET,SOCK_STREAM,0);
        if (nClientSocket == -1){
                printf("Failed to create socket at FTP_Connect\n");
                return INVALID_SOCKET;
        }
```

综合实验解决方案

❖客户端发送套接字连接请求的核心代码

- 客户端链接函数

```
tServerAddress.sin_port = nPort;
tServerAddress.sin_addr.s_addr = nConnectTo;
tServerAddress.sin_family = AF_INET;
if (connect(nClientSocket,(struct sockaddr*)&tServerAddress,
        sizeof(struct sockaddr_in)) == INVALID_SOCKET){
        printf("failed to connect at FTP_Connect\n");
        return INVALID_SOCKET;
        }
        printf("connected to %s:%d\n",FTP_GetIP(nConnectTo),(int)ntohs(nPort));
        return nClientSocket;
}
```

综合实验解决方案

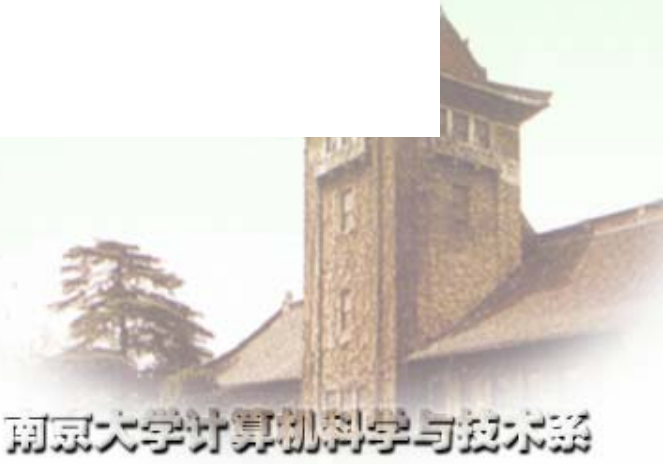
- 服务器端接收客户请求的套接字结构
- 客户端发送套接字连接请求的核心代码
- 与线程处理相关的核心函数
- 接收客户请求与实现客户会话的线程
- 文件管理
- 套接字通信

综合实验解决方案

❖ 与线程处理相关的核心函数

➢ 1.启动线程

```
int StartThread(tThread* pThread,ThreadFunc pFunc,void *pParam){
        if (pthread_create(pThread,NULL,(void*)pFunc,pParam) == 0)
                return 0;
        else
                return -1;
}
```



综合实验解决方案

❖ 与线程处理相关的核心函数

- 2.线程阻塞

```
int JoinThread(tThread hThread){
        if (pthread_join(hThread,NULL) != 0)
                return -1;
        else
                return 0;
}
```

综合实验解决方案

❖ 与线程处理相关的核心函数

➢ 3.释放进程占用资源

```
int DetachThread(tThread hThread){
        if (pthread_detach(hThread) != 0)
                return -1;
        else
                return 0;
}
```

综合实验解决方案

❖ 与线程处理相关的核心函数

➢ 4.初始化互斥变量

```
void InitializeMutex(tMutex* pMutex){
        if (pMutex){
                pthread_mutex_init(pMutex,NULL);
        }
}
```

综合实验解决方案

❖与线程处理相关的核心函数

➢ 5.销毁互斥变量

```
int DestroyMutex(tMutex* pMutex){

        pthread_mutex_destroy(pMutex);

        return 0;

}
```

综合实验解决方案

❖与线程处理相关的核心函数

➢ 6.互斥变量加锁

```
int LockMutex(tMutex *pMutex){
        if (pthread_mutex_lock(pMutex) != 0)
                return -1;
        else
                return 0;
}
```

综合实验解决方案

❖与线程处理相关的核心函数

➢ 7.互斥变量解锁

```
int LockMutex(tMutex *pMutex){
        if (pthread_mutex_lock(pMutex) != 0)
                return -1;
        else
                return 0;
}
```

综合实验解决方案

- 服务器端接收客户请求的套接字结构
- 客户端发送套接字连接请求的核心代码
- 与线程处理相关的核心函数
- 接收客户请求与实现客户会话的线程
- 文件管理
- 套接字通信

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 1.接入控制线程

```
unsigned int THREADCALL FTP_ServerThread(void *pParam){
        g_nServerSocket =FTP_RunServer(SPORT);
        if (g_nServerSocket == -1){
                printf("cannot start server. terminated.\n");
                return -1;
        }
        printf("server launched. waiting for users...\n");
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 1.接入控制线程

```
for (;;){
in_addr_t tClientAddress;

SOCKET nClientSocket = FTP_Accept(g_nServerSocket,&tClientAddress);

if (nClientSocket == -1){

        printf("cannot accept new user. terminated.\n");

        return -1;

}
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 1.接入控制线程

```
            if (checkuser(username, password) != -1){
                LockMutex(&g_tCounterMutex);
                Counter++;
                UnlockMutex(&g_tCounterMutex);
                 if (FTP_CreateNewClientThread(nClientSocket, tClientAddress) == -1){
                        printf("cannot add new thread. terminated.\n");
                        return -1;
                 }
            } else
                printf("Invalid username or password!\n");
        }
        return 0;
}
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 1.接入控制线程

```
int StartServer(){

        StartThread(&g_tFTP_ServerThread, FTP_ServerThread,NULL);

        DetachThread(g_tFTP_ServerThread);

        return 0;

}
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 2.创建客户会话线程的函数

```
int FTP_CreateNewClientThread(SOCKET nSocket,in_addr_t tClientAddress){
        tClientInfo *pNewInfo = NULL;
        pNewInfo = (tClientInfo *)malloc(sizeof(tClientInfo));
        if (!pNewInfo){
                printf("cannot alloc memory for tThreadInfo\n");
                return -1;
        }
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 2.创建客户会话线程的函数

```
    pNewInfo->m_nSocket = nSocket;
    pNewInfo->m_tClientAddress = tClientAddress;
    StartThread(&g_tFTP_ClientThread, FTP_ClientThread, pNewInfo);
    return 0;
}
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 2.创建客户会话线程的函数

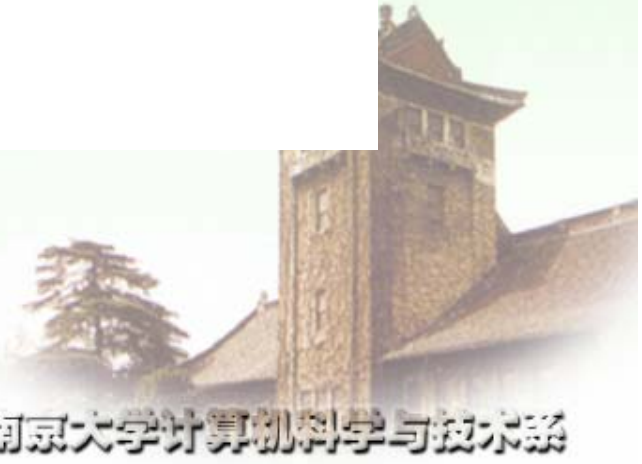
```
    pNewInfo->m_nSocket = nSocket;
    pNewInfo->m_tClientAddress = tClientAddress;
    StartThread(&g_tFTP_ClientThread, FTP_ClientThread, pNewInfo);
    return 0;
}
```

综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 3.客户会话线程

```
unsigned int THREADCALL FTP_ClientThread(void *pParam){
        tClientInfo *pClientInfo = (tClientInfo *)pParam;
        printf("Child Thread started.\n");
        if (FTP_RecvMessage(pClientInfo ->m_nSocket,&tRecvMessage) == -1){
                printf("failed to receive first message from client\n");
                return -1;
        }
```



综合实验解决方案

❖接收客户请求与实现客户会话的线程

➢ 3.客户会话线程

```
LockMutex(&g_tCounterMutex);
 counter++;
 UnlockMutex(&g_tCounterMutex);
for( ; ; ; ){ /*与客户的交互处理*/
      performRoutine(tRecvMessage, pClientInfo); //根据客户交互信息类型，做相应处理
}
FTP_CloseConnection(pClientInfo->m_nSocket);
printf("Child Thread ended.\n");
return 0;
}
```

综合实验解决方案

- 服务器端接收客户请求的套接字结构
- 客户端发送套接字连接请求的核心代码
- 与线程处理相关的核心函数
- 接收客户请求与实现客户会话的线程
- 文件管理
- 套接字通信

综合实验解决方案

❖文件管理

➢ 1.以ASCII模式传输文件

✓(1)读文本文件

```
if((fp=fopen(filename,"r"))==NULL) { /*  打开一个由 argv[1]所指的文件*/
    printf("not open");
    exit(0);
 }
while ((strlen(fgets(str,1024,fp1)))>0) {
       ……
}
 fclose(fp);
```

综合实验解决方案

- 文件管理
  - 1.以ASCII模式传输文件
    - ✓(2)写文本文件

```
if((fp=fopen(filename,"w"))==NULL)   { /*以只写方式打开文件 2 */
        printf("cannot open file\n");
        exit(0);
}
.......
fputs(str, fp );
```

综合实验解决方案

❖文件管理

➢ 2.以二进制模式传输文件

✓(1)读二进制文件

```
unsigned char buf[MAXLEN];
int rc;
if( (fp = fopen(filename, "rb" )) == NULL){
        printf("cannot open file\n");
         exit(0);
}
while( (rc = fread(buf,sizeof(unsigned char), MAXLEN,fp)) != 0 ) {
        …… /*通过网络发送文件*/
}
fclose(fp);
```

综合实验解决方案

❖文件管理

➢ 2.以二进制模式传输文件

✓(1)写二进制文件

```
unsigned char buf[MAXLEN];

int rc;

if( (fp = fopen(filename, "wb" )) == NULL){

        printf("cannot open file\n");

         exit(0);

}

  ……/*get file data over the network*/

fwrite( buf, sizeof(unsigned char), rc, outfile);
```

综合实验解决方案

- 服务器端接收客户请求的套接字结构
- 客户端发送套接字连接请求的核心代码
- 与线程处理相关的核心函数
- 接收客户请求与实现客户会话的线程
- 文件管理
- 套接字通信

综合实验解决方案

❖套接字通信

➢ 1.定义消息类型

```
typedef enum _tMsgType{
        FTP_USERAUTH,
        FTP_DOWNLOAD,
        FTP_UPLOAD,
        FTP_CD,
        FTP_MKDIR,
        FTP_RMDIR,
        FTP_LS,
        FTP_ASCII,
        FTP_BIN
} tPtclType;
```

综合实验解决方案

❖套接字通信

➢ 1.定义消息类型

```
typedef struct _tPtclHeader{
        uint32_t m_nVersion;
        tPtclType m_nType;
        uint32_t m_nBytes;
} tPtclHeader;
```

```
typedef struct _tPtclMsg{
        tPtclHeader m_tHeader;
        void* m_pData;
} tPtclMsg;
```

综合实验解决方案

❖套接字通信

➢ 2.发送元数据

```
int FTP_Send(SOCKET nSocket, const void* pBuffer, int nSize){

        return (int)send(nSocket, pBuffer, nSize, 0);

}
```

综合实验解决方案

❖套接字通信

➢ 3.阻塞式发送数据

```
int FTP_SendUntilAll(SOCKET nSocket,const void *pBuffer,int nSize){
    int nSent = 0;
    int nSentBytes;
    for (;;){
        nSentBytes = FTP_Send(nSocket,(void*)&((char*)pBuffer)[nSent],nSize - nSent);
        if (nSentBytes <= 0)return -1;
        nSent += nSentBytes;
        if (nSent == nSize)return 0;
    }
}
```

综合实验解决方案

❖套接字通信

➢ 4.发送消息头

```
int FTP_SendHeader(SOCKET nSocket, const tPtclHeader* pHeader){
        uint32_t bufHeader[4];
        bufHeader[0] = htonl(pHeader->m_nVersion);
        bufHeader[1] = htonl(pHeader->m_nType);
        bufHeader[3] = htonl(pHeader->m_nBytes);
        if (FTP_SendUntilAll(nSocket,bufHeader,sizeof(uint32_t)*4) == -1){
                printf("Connection lost at sending packet header:FTP_SendMessage\n");
                return -1;
        }
        return 0;
}
```

综合实验解决方案

❖套接字通信

➢ 5.发送消息体

```
int FTP_SendMessage(SOCKET nSocket, const tPtclMsg* pMsg){
    uint32_t bufEndMarker =     htonl(UMPP_END_MARKER);
    if (!pMsg || nSocket == -1 ||
        (pMsg->m_tHeader.m_nBytes != 0 && !pMsg->m_pData)){
        return -1;
    }
    if (FTP_SendHeader(nSocket,&pMsg->m_tHeader) == -1)return -1;
    if (pMsg->m_tHeader.m_nBytes != 0){
        if (FTP_SendUntilAll(nSocket,pMsg->m_pData,pMsg->m_tHeader.m_nBytes) == -1){
            printf("Connection Lost at sending packet message:FTP_SendMessage(%d)\n"
            ,nSocket);
            return -1;
        }
```

综合实验解决方案

❖套接字通信

➢ 5.发送消息体

```
        }
        if (FTP_SendUntilAll(nSocket,&bufEndMarker,sizeof(uint32_t)) == -1){
                printf("Connection Lost at sending endmarker:FTP_SendMessage(%d)\n",nSocket);
                return -1;
        }
        return 0;
}
```

综合实验解决方案

❖套接字通信

➢ 6.接收元数据

```
int FTP_Receive(SOCKET nSocket, void *pBuffer, int nSize){

        return (int)recv(nSocket, pBuffer,nSize,0);

}
```

综合实验解决方案

❖套接字通信

➢ 7.阻塞式接收元数据

```
static int ReceiveUntilFull(SOCKET nSocket,void *pBuffer,int nSize){

        int nReceived = 0;

        int nGotBytes;

        for (;;){

                nGotBytes = FTP_Receive(nSocket,(void*)&((char*)pBuffer)[nReceived],nSize - nReceived);

                if (nGotBytes <= 0)return -1;

                nReceived += nGotBytes;

                if (nReceived == nSize)return 0;

        }

}
```

综合实验解决方案

❖套接字通信

➢ 8.接收消息体

```
int FTP_RecvMessage(SOCKET nSocket, tPtclMsg* pMsg){
        uint32_t bufHeader[4];
        uint32_t bufEndMarker;
        if (!pMsg || nSocket == -1){
                return -1;
        }
        pMsg->m_pData = NULL;
        if (ReceiveUntilFull(nSocket,bufHeader,4*sizeof(uint32_t)) == -1){
                printf("Connection Lost at receiving packet header:FTP_RecvMessage(%d)\n",nSocket);
                return -1;
        }
```

综合实验解决方案

❖套接字通信

➢ 8.接收消息体

```
pMsg->m_tHeader.m_nVersion    = ntohl(bufHeader[0]);
pMsg->m_tHeader.m_nType       = ntohl(bufHeader[1]);
pMsg->m_tHeader.m_nBytes      = ntohl(bufHeader[3]);
if (pMsg->m_tHeader.m_nBytes == 0){
        pMsg->m_pData = NULL;
}
}
```

综合实验解决方案

❖套接字通信

➢ 8.接收消息体

```
else{
        pMsg->m_pData = malloc(pMsg->m_tHeader.m_nBytes);
        if (!pMsg->m_pData){
                printf("Cannot malloc %dbytes for getting data at
FTP_RecvMessage(%d)\n",pMsg->m_tHeader.m_nBytes,nSocket);
                return -1;
        }
        if (ReceiveUntilFull(nSocket,pMsg->m_pData,pMsg->m_tHeader.m_nBytes) == -1){
                printf("Connection Lost at receiving message:FTP_RecvMessage(%d)\n",nSocket);
                SAFE_RELEASE(pMsg->m_pData)
                return -1;
        }
}
```

综合实验解决方案

❖套接字通信

➢ 8.接收消息体

```
if (ReceiveUntilFull(nSocket,&bufEndMarker,sizeof(uint32_t)) == -1){
        printf("Connection Lost at receiving end marker:FTP_RecvMessage(%d)\n",nSocket);
        SAFE_RELEASE(pMsg->m_pData)
        return -1;
}
bufEndMarker = ntohl(bufEndMarker);
if (bufEndMarker != UMPP_END_MARKER){
        printf("Recieved non-EndMarker %x :FTP_RecvMessage(%d)\n",bufEndMarker,nSocket);
        SAFE_RELEASE(pMsg->m_pData)
        return -1;
}
return 0;
}
```

第11章 综合练习
一个小型远程访问FTP服务系统