

OSP通讯程序概要设计说明书

|  |  |  |
| --- | --- | --- |
| 版 本 号： | | V0.1 |
|  | |  |
|  | | |
| 编 制： | 陈上全 | |
| 审 核： | 肖楚然 | |
| 会 签： |  | |

**修订记录**

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 版本号 | 描述 | 作者 |
| 2018-08-20 | 0.1 | 初稿完成 | 陈上全 |
| 2018-10-18 | 0.2 | 修订 | 陈上全 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

OSP通讯程序概要设计说明书

1. **设计目的**

该OspDemo程序基于OSP通讯框架，实现了客户端与服务端之间的短消息通讯及文件传输。通过简单的OspDemo程序的编写，进一步掌握OSP的通讯框架及其实现原理。

1. **设计原理及流程**

OSP具有跨操作系统平台的编程界面，且OSP消息协议是基于TCP协议封装的通信协议，另外实现了有边界的完整消息传输，提供给OSP使用者一个可靠的实现网络间消息通讯的编程模型。

**基于OSP的逻辑模型：**

图（1）

**客户端与服务端通讯流程：**



图（2）

**需要完成的主要工作：**

1. **建立各个应用程序的监听结点，完成OSP应用程序之间的Node连接；**

客户端与服务端均可创建自己的监听结点，在创建之前必须先进行OspInit初始化操作，同时打开两端的telnet服务，便于信息获取及调试。这里，服务端暂选择telnet登陆端口号为2510，客户端选择telnet登陆端口号为2520。

由客户端向服务端的发起连接请求（前提：服务端已成功创建自己的TCP监听结点），同时客户端与服务端各自创建自己的APP与Instance。

备注：需考虑断链检测机制。

1. **继承CInstance基类，定义自己的实例数据（CDemoInstance），重载实现自己的InstanceEntry及DaemonInstanceEntry消息处理函数；**

定义各个消息类型，负责消息的分发处理；同时定义各个消息的处理函数，实现自定义的功能。

1. **定义并发送客户端到服务端的第一个消息类型及处理函数，实现Instance的合理分配及管理，并完成服务端对客户端的感知，实现对端之间的通讯；**

为了实现如上图红色路径点对点通讯形式，需要考虑Instance资源的合理分配及管理。可通过发送消息至DaemonInstanceEntry消息处理函数，由该函数负责寻找到空闲的Instance，并将其状态由IDLE\_STAT跳转到用户定义的工作状态，指定指令内容发送第一条消息。停止工作时，将Instance状态跳转为原来的IDLE\_STAT状态。

1. **实现短消息通讯及文件的传输（考虑分包管理设计）；**
2. 实现短消息通讯

例如上图中OSP应用程序2里面的APP1中的Instance 1，在发送数据时指定通过Node M结点发送，目的APP是1，目的Instance是2，这样数据就被发往指定目标（红色路径）；本demo程序，消息通讯固定Instance号为1。

1. 文件传输

考虑分包的管理形式进行文件的传输，如下图所示：



图（3）

1. 获取文件基本信息并分配空闲的Instance，同时创建文件发送list（含有开始、暂停及取消按键），用于控制文件发送流程。

每次选择待发送的文件时，收集文件的基本信息，用于具体的分包流程。同时，还需要分配空闲的Instance资源，创建文件发送的逻辑控制界面，负责控制文件发送的流程。

包的结构体定义如下：

// 包的结构体;

typedef struct tagFileMessage

{

s8 fileHead[4]; // 标志作用;

u32 fileStart; // 包中数据在整个文件中的起始位置标志;

u32 fileSize; // 包中数据的长度;

u8 filePacket[MAX\_FILE\_PACKET]; // 包中所传输的文件数据;

}TFileMessage;

文件信息收集内容如下：

// 文件信息结构体;

typedef struct tagFileInfo

{

u32 filePacketNum; // 文件分包数;

u32 fileLength; // 文件总长度;

s8 strFileName[MAX\_FILE\_NAME + 1]; // 文件名称;

}TFileInfo;

1. 响应开始、暂停、取消等逻辑控制

开始： 将文件基本信息发送至服务端，进入流程（3）；

暂停： 将Instance线程的暂停标志位置1，分包流程（7）中作暂停发送处理；

取消： 将Instance线程的取消标志位置1，分包流程（7）中作取消发送处理；

其它按键处理：例如一键发送、一键停止、一键取消等。

1. 服务端分配一个空闲的Instance，负责文件的接收处理流程

服务端响应客户端的文件发送事件，分配本端空闲的Instance，同时获取的文件基本信息，确定文件的存储位置。

1. 开始分包发送文件到服务端

定义包发送的处理函数：SendFileInfo(s32 fStart,s32 fSize,char \*fHead)

首要确定第一包的数据内容：包头标志位为“0”，起始位及数据长度均为0，第一包的发送主要负责进入到发包流程中，接下来的包发送，是基于服务端接收数据包的消息回复，及（7）。

1. 服务端检验收到的文件分包并回复
2. 解析包头，根据包头定义的标志内容，进行分包分发处理；
3. 正常包的接收，每次判定包的偏移位置，关注是否是最后一包，最后一包的回复事件决定文件发送流程的结束；
4. 取消或错误等包的响应，用于文件发送的逻辑控制。
5. 判断是否是最后一包，不是则进入（7），否则进入（8）
6. 发包流程的主要处理函数

接下来的包发送，是基于服务端接收数据包的消息回复：

正常确认包回复“OK！”，

错误包回复“ERR”，

取消包回复“CCL”，

暂停包回复“STP”，

包的大小暂定为10K，宏定义可修改，下一包的发送内容是以上一次包的偏移位置+10K的内容，同时更新本次发送包的偏移位置；

暂停、取消包发送操作，不进行文件的读取，只设置标志位并发送该特征包到服务端，服务端再检验包头信息，进行特定的消息处理（5）；

正常发送包的标志位为“FFF”。

1. 结束处理，Instance资源的释放处理

文件发送流程结束后，需要对客户端和服务端各自分配得到的Instance资源进行释放，便于资源的再次利用。

1. **文件覆盖处理**

考虑到文件传输时，服务端同名文件的处理，采取下流程图方案：



图（4）

文件选择时，即发送文件信息到服务端，服务端根据设定好的文件夹目录下，查找是否存在同名文件，若存在，即发送“文件已存在”事件到客户端，客户端即时弹出选择对话框，让用户确定是否进行文件覆盖选择；若确定覆盖，则删除服务端原有的同名文件，并开始申请空闲的Instance资源进行文件传输，若不覆盖，则以文件名加“- 副本（XX）”的形式，进行文件的存储。

1. **其它特性设计**
2. 限速

文件传输过程中设定发包时钟，控制发包速率。

1. 多端通信

支持多用户使用。

1. **设计结果**

实现客户端到服务端的短消息通讯及文件传输。

发送中：

