**维护工具需求设计**

**目 录**

[1. 引言 3](#_Toc75790349)

[1.1 编写目的 3](#_Toc75790350)

[1.2 术语与定义 3](#_Toc75790351)

[1.3 参考资料 3](#_Toc75790352)

[2. 任务概述 3](#_Toc75790353)

[2.1 系统背景 3](#_Toc75790354)

[2.2 运行环境 3](#_Toc75790355)

[2.3 功能总体框架 4](#_Toc75790356)

[3. 功能设计 4](#_Toc75790357)

[3.1 客户端与服务端通信 4](#_Toc75790358)

[3.1.1 基于TCP协议的Socket通信 4](#_Toc75790359)

[3.1.2 基于UDP协议的Socket通信 6](#_Toc75790360)

[3.1.3 多客户端连接流程 7](#_Toc75790361)

[3.1.4 TCP客户与服务器实现信息回射 7](#_Toc75790362)

[3.2 所有主机分组 7](#_Toc75790363)

[3.2.1 主机分组编号存储 7](#_Toc75790364)

[3.2.2 主机分组通信流程 8](#_Toc75790365)

[3.2.3 主动注册 9](#_Toc75790366)

[3.3 远程调用 10](#_Toc75790367)

[3.3.1 Linux基本命令使用 10](#_Toc75790368)

[3.3.2 配置脚本 10](#_Toc75790369)

[3.4 远程升级 10](#_Toc75790370)

[3.5 状态查看 11](#_Toc75790371)

[3.5.1 服务器主动查看 11](#_Toc75790372)

[3.5.2 客户端上报 11](#_Toc75790373)

[3.6 界面设计 11](#_Toc75790374)

[3.6.1 vty虚拟终端 11](#_Toc75790375)

[3.6.2 http页面 11](#_Toc75790376)

[4. 非功能规格 12](#_Toc75790377)

[4.1 性能 12](#_Toc75790378)

# 引言

## 编写目的

由部门任务布置，利用libhv框架和vty工具，在此基础上进行改进，在服务器上对客户端进行远程维护管理，因此在具体代码开发前，撰写需求设计规格说明书，确认开发需求并对所需功能进行细化，以便更好地理解系统的基本情况以及各模块的功能。

## 术语与定义

客户端维护管理系统，遵从libhv中的API规范，常用脚本语句。

## 参考资料

# 任务概述

## 系统背景

libhv是由c/c++编写HTTP API服务端/客户端最简单的库，也是一个跨平台的类似libevent、libev、libuv的异步IO事件循环库，提供了更加简单的API接口和更加丰富的协议（包括http、ftp、smtp、dns、icmp等），具有如下特性：

* 跨平台(Windows, Linux, Mac)
* 支持https
* 支持RESTful API
* 支持application/json、application/x-www-form-urlencoded、multipart/form-data
* 内置web service文件服务和indexof service目录服务
* 可扩展多进程/多线程模型

vty(Virtual Teletype Terminal)是一个虚拟终端连接工具，属于telnet的虚拟端口，可用于远程登陆。

依据libhv框架和vty虚拟终端，按照需求增加一些功能：对远程客户端进行分组管理，按不同类型和所在区域进行分组；使用服务器对客户端进行远程调用；对客户端进行远程升级；查看客户端的当前状态；结合vty工具，实现远程登陆；设计一个简单的http界面，在界面上进行post请求操作。

## 运行环境

1. 设备环境：常规电脑配置
2. 硬件环境：常规硬件配置
3. 软件环境：开源框架libhv和vty
4. 网络环境：常规网络
5. 操作环境：Red Hat Linux服务器

## 2.3 功能总体框架

总的功能是要实现服务端对所有远程主机进行日常维护和管理，必要时还需以远程通信方式进行信息交换和数据传输等。为了方便后期管理，需要对所有主机编号以进行分组分域。

下图是整个需求的功能结构示意图。



图1 系统功能概述

# 功能设计

## 3.1 客户端与服务端通信

### 3.1.1 基于TCP协议的Socket通信

（1）多个客户与服务器通信：考虑到实际需求，由于可能存在多个客户在同一时间请求连接到同一服务器，因此需要用到并发服务器，这是用于提供并发性的一种常用Unix技术。每个客户连接都迫使服务器为它创建一个新的线程。

（2）客户和服务器通信概述：服务器首先启动，稍后某个时刻客户启动，它试图连接到服务器。假设客户给服务器发送一个请求，服务器处理该请求，并且给客户返回一个响应。这个过程一直持续下去，直到客户关闭连接的客户端，从而给服务器发送一个EOF（文件结束）通知为止。服务器跟着也关闭连接的服务器端，然后结束运行或者等待新的客户连接。

（3）客户与服务器实现连接所需要的主要功能：

1， 为了执行网络I/O，一个进程首先要做的就是调用socket函数，同时指定期望的通信协议类型（比如使用IPv4的TCP）。socket函数返回一个小的非负整数值，它被称为套接字描述符。

2， TCP客户用connect函数来建立与TCP服务器的连接，调用connect函数将激发TCP的三路握手过程，并且仅在连接建立成功或出错时才返回。

3，调用bind函数将本地协议地址与一个套接字进行绑定，协议地址一般是IP地址与端口号的组合。

4，TCP服务器调用listen函数将一个未连接的套接字转换为一个被动套接字，指示内核应接受指向该套接字的连接请求。

5，TCP服务器调用accept函数，用于从已完成连接队列队头返回下一个已完成连接。如果已完成连接队列为空，则进程进入睡眠。

6，当有多个客户请求连接时，服务器的对应进程为每一位客户建立一个线程，实现服务器的并行连接功能。

7，TCP服务器调用close函数来关闭套接字，并终止TCP连接。

（4）服务端实现步骤：

Step 1：创建套接字（socket）

Step 2：将套接字绑定到一个本地地址和端口上（bind）

Step 3：将套接字设为监听模式，准备接收客户端请求（listen）

Step 4：等待客户请求到来，当请求到来后，接收连接请求，返回一个新的对应于此次

连接的套接字（accept）

Step 5：用返回的套接字和客户端进行通信（send/recv）

Step 6：返回，等待另一客户请求

Step 7：关闭套接字（close）

（5）客户端实现步骤：

Step 1：创建套接字（socket）

Step 2：向服务器发出连接请求（connect）

Step 3：和服务器端进行通信（send/recv）

Step 4：关闭套接字（close）

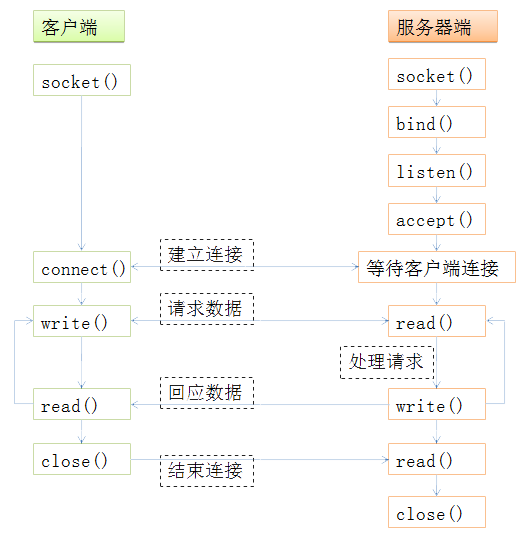


图2 服务器与客户端使用TCP协议通信流程图

### 3.1.2 基于UDP协议的Socket通信

客户不与服务器建立连接，而是只管使用sendto函数给服务器发送数据报，其中必须指定目的地（服务器）的地址作为参数。类似的，服务器不接受来自客户的连接，而是只管调用recvfrom函数，等待来自某个客户的数据到达。recvfrom将与所接受的数据报一道返回客户的协议地址，因此服务器可以把响应发给正确的客户。

（1）服务端实现步骤：

Step 1：创建UDP套接字（socket）

Step 2：将套接字绑定到一个本地地址和端口上（bind）

Step 3：等待接收客户端发送的数据信息（recvfrom）

Step 4：读取数据信息并回射发送者

Step 5：关闭套接字

（2）客户端实现步骤：

Step 1：定义发送信息

Step 2：创建套接字（socket），包含将要发送的信息

Step 3：把服务器地址填入套接字地址结构

Step 4：向服务器发送数据（sendto）

Step 5：关闭套接字

### 3.1.3 多客户端连接流程

一、多客户端连接：

当有多个客户端和服务器连接时，服务器就需要为每个客户端创建新连接。

二、连接完毕后，服务器要为每个客户端产生用于通讯的Socket与客户端进行通讯，把此Socket保存到集合中，使用远程终结点进行区分每个连接，可以保存在集合中（比如List，Dictionary..），可以实现与客户端单聊和群发的功能。

三、接收客户端消息：

服务端的Receive方法只能接收一个客户端的数据，并且需要循环接收，需要启用一个线程进行完成，而对于多客户端，就需要服务器为每个客户端启用一个单独的线程，对客户端的数据进行接收，可以把每个线程放到集合中，以便后面管理。为了保证线程的安全，需要为线程上锁；当有客户端在一定时间内不再与服务器进行通信（或者设置keepalive超时，自动断开连接，调用hevent.c中的hio\_set\_connect\_timeout函数来执行），需要使连接客户端的线程休眠，当再次通信时可自动唤醒。

四、服务器发送消息：

服务器发送消息时，会使用集合中的远程终结点对应的Socket(或者线程号)来区分每个客户端，群发：直接遍历集合中的每个客户端对应的套接字进行发送。

用TCP协议，需要监听固定的一个端口，建立连接时会自动分配另一个端口和套接字。用UDP协议，通常固定的一个端口接收信息，可以用这一个端口对所有客户端，根据需要也可以使用另外的端口和套接字。

### 3.1.4 TCP客户与服务器实现信息回射

一，在某些情况下，客户需要与服务器进行信息交互以实现某种功能，交互的大体流程如下：

1. 客户从标准输入读入一行文本，并写给服务器；
2. 服务器从网络输入读入这行文本，并回射给客户；
3. 客户从网络输入读入这行回射文本，并显示在标准输出上。
4. 在实现简单的信息回射后，后期再根据具体的业务需求，修改服务器对来自客户的输入的处理过程，并将处理结果以文本的形式返回给客户。

## 3.2 所有主机分组

### 3.2.1 主机分组编号存储

假设按不同主机的性能类别来分组则共有3大类，再按不同主机所在的地域大区来分则共有5小类，那么预计总的类别为3\*5=15类。使用此15种类别就可以对所有主机进行分组分域，然后再各自进行编号。

1. 对各主机编号以进行分组分域，编号后各主机都得到一个关键码值key，按组号来构造多个存储结构（比如链表或者数组），通过某种函数(hash函数)使元素的存储位置与它的key之间能够建立一一映射的关系，那么在查找时通过该函数可以很快找到该元素。
2. 采用哈希函数，但它不是固定的，一般常用的有直接定制法，除留余数法，平方取中法，折叠法，随机数法，数学分析法，初步考虑选择使用除留余数法。
3. 当向该数据结构插入元素时，存入根据关键码以此函数计算出的位置，当搜索时，也是先要将给定的关键码用函数转换成存储位置进行查找，将得到位置处的元素进行比较，若关键码值相同，则搜索成功。

4. 但是通过一个哈希函数得到的位置，一定是会有冲突的。例如用除留余数法，哈希函数为key%100。在此情况下数字1与数字101得到的存储位置就是相同的，这样就是哈希冲突,哈希冲突一般有两种解决方式，一种是闭散列，另一种是开散列。

闭散列（开放地址法）：当发生哈希冲突时，如果哈希表未被装满，说明在哈希表中还有空位，那就可以把key值存放到了列表的下一个空位。

开散列（链地址法）：首先对关键码集合用哈希函数计算哈希表中的偏移位置，具有相同地址的关键码归于同一子集合，每一个子集合称为一个桶，各个桶中的元素通过一个单链

表链接起来，各链表的头结点存储在哈希表中。

对于每个客户端中所带的数据变量，可分为全局变量和局部变量，全局变量如客户端所在的区域和类型等，局部变量如ip地址、port、标识id等。全局变量区域命名为所在区域的首字母大写英文名，类型为sdr、utrace和probe，放在枚举类型中。客户端的结构信息如下：

{

enum 区域{

区域一

区域二

区域三

……..

};

enum 类型{

sdr

utrace

probe

};

ip

hostname

端口号

id(唯一，客户端编号)

…..  
}

### 3.2.2 主机分组通信流程

1. 当客户端与服务器使用TCP连接成功后，判断客户端把本身的标识信息：所在区域和类别等信息是否已经传入到服务器上进行存储。在服务器上定义一个数组，长度为客户端的数量，数组中的每个元素对应客户端的id，包含客户端的结构信息。如果在服务器上没有找到，则进行存储；否则，显示该客户端已被标识，把信息发送给客户端。当服务器要显示客户端所在的区域或类别的数量时，服务器对命令行参数进行解析，找到对应的关键字find，执行程序从存储列表中进行查找统计。

2. 当客户端与服务器使用TCP连接失败时，表明服务器的ip地址已更改，则跳转使用UDP进行连接。当UDP连接成功后，服务器把本身的ip地址发送给客户端，客户端接收信息后，更新服务器的ip地址；更新成功后，客户端发送更新成功的信息给服务器，并把TCP连接失败、服务器更换ip地址的信息写入到日志中，返回步骤1。

对于TCP连接失败，使用UDP进行连接，可以把这两者封装在一起进行判断并调用。对于命令行输入的字符，可分为两种：命令和一般字符，可通过判断是否带命令头来区分。



图3 主机分组通信流程图

### 3.2.3 主动注册

1. 客户端发送ping服务器ip命令和指定的次数(参考protocol/icmp.c中的ping函数或者examples/consul\_cli.cpp中的注册参数)；
2. 如果客户端在设定的时间内(1000ms)没有接收到服务器返回的信息，则表明客户端与服务器已断开连接或者服务器的ip地址已变动；使用UDP使客户端与服务器重新建立连接，获取当前服务器的ip地址并传给客户端进行更新服务器的ip地址，把此次ping操作失败的记录写入到日志中(使用hlog中的hlogf宏调用logger\_print函数)，返回步骤1；此操作与3.2.2节相同。
3. 当客户端在设定的时间内接收到服务器发送的信息，则表明注册成功；否则，注册失败。



图4 主动注册流程图

## 远程调用

### Linux基本命令使用

在某些情况下，服务器需要了解某台或者某些远程主机的当前运行状况，那么就可能会需要查看此主机内的某些文件或者需要此主机执行某些操作，因此需要远程交互。

当服务器与客户端选择使用TCP连接进行远程交互时，服务器在终端输入Linux常用命令：ls、mkdir、rm、touch、vim、cat等(可在examples/hmain\_test中进行更改，加入一些自己需要的命令，并实现相关操作)，并把命令操作传给指定客户端或集群客户端，客户端从服务器接收到命令后执行相关操作。当操作完成后，把成功信息返回给服务器，否则把操作出错的信息发送给服务器,并调用libhv的hlog中的hlogw函数把操作出错的命令写入到日志中。

### 配置脚本

当遇到某个实际需求时，将需求设计成命令和函数等，并将对应的代码写进shell脚本。服务器把编写好的脚本通过curl/wget/nc工具传给指定客户端或集群客户端的指定目录，客户端运行编写好的脚本并把脚本执行完毕的信息返回到服务器上。

## 远程升级

由于客户端的文件会随着需求的改变而改变本身的功能，因此客户端需要从服务器上下载最新文件版本进行升级，升级涉及到的流程如下：

1. 服务器与客户端连接后，对于想要升级的文件，服务器通过线程集合向指定的客户端或集群客户端发送升级命令update；
2. 客户端接收到升级命令后，执行update程序获取服务器的文件版本最新日期号和本身的文件版本日期号(文件版本日期号可到指定目录下查找)，进行对比；
3. 如果日期号相同，则不升级；
4. 否则，服务器把文件传输给客户端并调用远程升级脚本进行升级；
5. 如果客户端升级成功，则把升级成功消息发送给服务器；否则发送升级失败消息，并且把此次操作状态信息写入到升级日志中。

对于远程升级，需要定义获取文件版本日期号程序。对于服务器，搭建tomcat环境，将升级包放上去，开启服务。当需要升级时，客户端利用curl/wget/nc工具，将升级包通过curl/wget/nc函数传输过来。

## 状态查看

### 服务器主动查看

服务器向客户端发送命令执行事先编写好的脚本，客户端对服务器发送来的命令进行解析，找到对应的命令程序操作，调用脚本并执行，把脚本的输出信息返回给服务器。如果服务器没有在设定的时间内接收到客户端发来的信息，则表明连接断开，把这次断开的记录写入状态日志中，并重新建立连接。具体实现如下：

服务器发送命令：sh 客户端脚本所在路径(该路径用来存放脚本，可从服务端传输过来)

客户端接收信息：对命令进行解析，调用脚本执行程序sh来执行脚本，根据返回值判断，0为正常，1为警告，2为不正常，并把信息发送给服务器。

服务器接收信息：服务器把包括客户端的id等状态信息写入到状态日志中。

### 客户端上报

在客户端上设置一个定时包每过一段时间来执行指定的脚本，来判断当前客户端的状态。可调用event/hevent.c中的hio\_set\_heartbeat函数，并自己定义一个函数来执行脚本，根据其返回值判断，0为正常，1为警告，2为不正常。并把相关信息记录到状态日志中。实现参照3.5.1节。

## 界面设计

1. **vty虚拟终端**

利用vty工具，用户可以通过Telnet或SSH方式登录设备实现本地或远程维护，这需要对vty进行配置。vty配置认证模式有三种：None认证模式、Password认证模式、AAA认证模式。

1. None认证模式

在服务器上设置一定数量的vty连接（数量为客户端的个数），来提供客户端远程连接服务器。

1. Password认证模式

在服务器上设置vty时的认证密码，以实现登录。即远程客户端连接服务器时需要输入设置好的密码才能远程登陆。

1. AAA认证模式

在服务器上建立AAA用户，然后设置vty的认证模式为aaa认证，客户端在用户模式下远程登陆服务器。

对于vty配置认证模式的选择，需要用户根据使用需求以及对设备安全的考虑来配置vty。

1. **http页面**

实现简单的http页面post请求操作，向指定资源提交数据进行处理请求（例如提交表单或者上传文件），数据被包含在请求体中。

该页面包含三个部分：命令输入框、发送按钮和返回信息框。

命令输入框：包含3.1—3.5节的功能实现。

返回信息框：即命令输入框输入操作在客户端上返回的信息。

# 非功能规格

## 性能

数据性能：数据多为整形和字符型

适应性能：只在Linux服务器上运行