目录

[1 程序简介 1](#_Toc48035296)

[1.1 应用背景 1](#_Toc48035297)

[1.2 程序目标 1](#_Toc48035298)

[1.2.1监控功能 1](#_Toc48035299)

[1.2.2更新功能 1](#_Toc48035300)

[1.2.3 回退功能 1](#_Toc48035301)

[1.3 环境配置 1](#_Toc48035302)

[2 程序设计 2](#_Toc48035303)

[2.1 流程设计 2](#_Toc48035304)

[2.1.1 网关守护进程流程图 2](#_Toc48035305)

[2.1.2 文件服务器流程图 3](#_Toc48035306)

[2.1.2 文件服务器流程图 4](#_Toc48035307)

[2.2 数据包设计 5](#_Toc48035308)

[2.2.1 网关守护进程与网关主程序的通信 5](#_Toc48035309)

[2.2.2 网关守护进程与文件服务器的通信 6](#_Toc48035310)

[2.3 具体设计 7](#_Toc48035311)

[2.3.1 声明设计 7](#_Toc48035312)

[2.3.2 结构体设计 7](#_Toc48035313)

[2.3.3 函数设计 8](#_Toc48035314)

[3 效果演示 9](#_Toc48035315)

[3.1 编译程序 9](#_Toc48035316)

[3.2 传输文件 9](#_Toc48035317)

[3.3 监控功能 9](#_Toc48035318)

[3.4 在线更新功能 10](#_Toc48035319)

[3.5 离线更新功能 12](#_Toc48035320)

[3.6 版本回退功能 13](#_Toc48035321)

[4 可拓展方面 16](#_Toc48035322)

[4.1 异常返回值 16](#_Toc48035323)

[4.2 日志功能 16](#_Toc48035324)

[5 实习总结 17](#_Toc48035325)

1 程序简介

# 1.1 应用背景

原网关Linux系统只能现场使用SD卡更新程序，不满足在线OTA的需求；而且工业客户对网关可靠性要求非常高，需要设计一个守护进程能够保证网关7\*24h持续稳定工作。

# 1.2 程序目标

完成以下功能：

## 1.2.1监控功能

能够监控网关主程序的运行状态，定时检查，在该进程未启动或意外崩溃时将其重新启动。

## 1.2.2更新功能

能够与网关主程序进行socket通信，实现网关主程序的替换更新。能够从本地或者远端文件服务器上获取升级文件，并执行替换原应用程序文件的操作。

## 1.2.3 回退功能

能根据网关主程序的要求，在必要时回退至更新前的版本。

# 1.3 环境配置

|  |  |
| --- | --- |
| 任务环境 | 配置详情 |
| 编程环境 | Visual Studio Code V1.47.3 |
| 编译环境 | Ubuntu 16.04.6 LTS |
| 服务器运行环境 | Ubuntu 16.04.6 LTS |
| 守护进程运行环境 | 米尔电子开发板（MYC-Y6ULY2-4E512D-50-I） |

2 程序设计

# 2.1 流程设计

## 2.1.1 网关守护进程流程图



## 2.1.2 文件服务器流程图



## 2.1.2 文件服务器流程图



# 2.2 数据包设计

在网关守护进程中存在3类大小的数据包，分别为小包（12字节），中包（116字节）和大包（1024字节），具体数据包设计如下所示。

## 2.2.1 网关守护进程与网关主程序的通信

1. 网关主程序向守护进程请求更新

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0x01（1字节） | flag（1字节） | 硬件编号长度（2字节） |
| 硬件编号（16字节） | | |
| 硬件编号（16字节） | | |
| 软件编号长度（2字节） | | 软件编号（2字节） |
| 软件编号（16字节） | | |
| 软件编号（12字节） | | |
| 软件编号（2字节） | | 厂家编号长度（2字节） |
| 厂家编号（16字节） | | |
| 厂家编号（16字节） | | |
| 异或校验码（4字节） | | |
| 0xBBBBBBBB | | |

flag:

0x01 在线停服务

0x02 在线不停服务

0x03 离线停服务

0x04 离线不停服务

1. 守护进程向网关主程序反馈更新文件下载结果

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0x02（1字节） | flag（1字节） | 备注（2字节） |
| 0xBBBBBBBB | | |

flag+备注：

0x00+0x0000 下载成功

0xFF+errno 下载失败

1. 网关主程序向守护进程反馈要求版本回退

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0x03（1字节） | 0x00（1字节） | 0x0000（2字节） |
| 0xBBBBBBBB | | |

1. 守护进程向网关主程序反馈要求版本回退结果

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0x04（1字节） | flag（1字节） | 0x0000（2字节） |
| 0xBBBBBBBB | | |

flag：

0x00 可回退

0xFF 不可回退

## 2.2.2 网关守护进程与文件服务器的通信

1. 守护进程告知服务器要取文件

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0x11（1字节） | 0x00（1字节） | 硬件编号长度（2字节） |
| 硬件编号（16字节） | | |
| 硬件编号（16字节） | | |
| 软件编号长度（2字节） | | 软件编号（2字节） |
| 软件编号（16字节） | | |
| 软件编号（12字节） | | |
| 软件编号（2字节） | | 厂家编号长度（2字节） |
| 厂家编号（16字节） | | |
| 厂家编号（16字节） | | |
| 异或校验码（4字节） | | |
| 0xBBBBBBBB | | |

1. 服务器反馈守护进程文件查找结果

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0xA2（1字节） | flag（1字节） | 备注（2字节） |
| 0xBBBBBBBB | | |

flag+备注：

0x00+0x0000 文件查找成功并准备传输

0xFF+errno 文件查找失败

1. 守护进程反馈服务器ACK

|  |  |
| --- | --- |
| 0xAAAAAAAA | |
| 0x12（1字节） | 0x000000（3字节） |
| 0xBBBBBBBB | |

1. 服务器向守护进程传输文件

|  |  |  |
| --- | --- | --- |
| 0xAAAAAAAA | | |
| 0xA3（1字节） | 0x00（1字节） | 文件长度（2字节） |
| 文件段（1008字节） | | |
| 异或校验码（4字节） | | |
| 0xBBBBBBBB | | |

1. 守护进程向服务器反馈ACK

|  |  |
| --- | --- |
| 0xAAAAAAAA | |
| 0x13（1字节） | 0x000000（3字节） |
| 0xBBBBBBBB | |

1. 服务器告知守护进程传输结束

|  |  |
| --- | --- |
| 0xAAAAAAAA | |
| 0xA4（1字节） | 0x000000（3字节） |
| 0xBBBBBBBB | |

1. 守护进程向服务器反馈ACK

|  |  |
| --- | --- |
| 0xAAAAAAAA | |
| 0x14（1字节） | 0x000000（3字节） |
| 0xBBBBBBBB | |

# 2.3 具体设计

## 2.3.1 声明设计

|  |  |
| --- | --- |
| 声明名 | 取值 |
| \_IP | 文件服务器的IP地址 |
| \_PORT | 文件服务器提供可连接的端口 |
| HEAD | 数据包的包头（0xAAAAAAAA） |
| TAIL | 数据包的包尾（0xBBBBBBBB） |
| PS\_SIZE | 小数据包的大小（12） |
| PM\_SIZE | 中数据包的大小（116） |
| PL\_SIZE | 大数据包的大小（1024） |
| PATCH\_SIZE | 大数据包中文件部分的大小（1008） |
| XML\_NAME | 版本回退中会检查的list.xml文件的相对路径 |
| EXE\_PATH | 守护进程所守护的网关主程序的相对路径 |
| FILE\_NAME | 更新文件包被本地存放或者下载的位置的相对路径 |
| TMP\_XML\_NAME | 更新文件包中的list.xml的相对路径 |
| FD\_BUF\_SIZE | 守护进程exec网关主程序所传递文件描述符的字符串缓冲数组的大小 |
| BUF\_SIZE | 通用缓冲区大小 |
| MAX\_FILE\_NUM | list.xml中最大的文件数量 |
| MAX\_NAME\_LEN | list.xml中最大的文件名长度 |
| SIM\_FILE | 服务器模拟更新文件的相对路径 |
| SERVER\_BUF\_SIZE | 服务器的通用缓冲区大小 |

## 2.3.2 结构体设计

1. Package\_S

|  |  |  |
| --- | --- | --- |
| 成员名 | 成员类型 | 备注 |
| head | uint32\_t | 小数据包的包头，通常为0xAAAAAAAA |
| cmd | uint8\_t | 小数据包的命令代码 |
| flag | uint8\_t | 小数据包的额外标志位，用于传输成功或失败信息 |
| info | uint8\_t [2] | 小数据包的备注部分，info[1]用于传递发送方失败时的errno |
| tail | uint32\_t | 小数据包的包尾，通常为0xBBBBBBBB |

1. Package\_M

|  |  |  |
| --- | --- | --- |
| 成员名 | 成员类型 | 备注 |
| head | uint32\_t | 中数据包的包头，通常为0xAAAAAAAA |
| cmd | uint8\_t | 中数据包的命令代码 |
| flag | uint8\_t | 中数据包的额外标志位，用于传输成功或失败信息 |
| hw\_len | uint16\_t | 中数据包的配置信息，代表硬件编号的有效长度 |
| hw\_dat | uint8\_t [32] | 中数据包的配置信息，代表硬件编号 |
| sw\_len | uint16\_t | 中数据包的配置信息，代表软件编号的有效长度 |
| sw\_dat | uint8\_t [32] | 中数据包的配置信息，代表软件编号 |
| fc\_len | uint16\_t | 中数据包的配置信息，代表厂家编号的有效长度 |
| fc\_dat | uint8\_t [32] | 中数据包的配置信息，代表厂家编号 |
| checksum | uint32\_t | 中数据包的校验位，对于数据包进行异或校验得出（包括包头和包尾） |
| tail | uint32\_t | 中数据包的包尾，通常为0xBBBBBBBB |

1. Package\_L

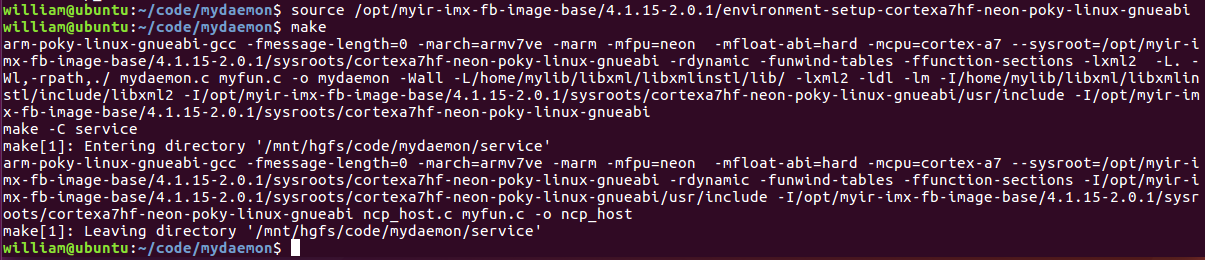
|  |  |  |
| --- | --- | --- |
| 成员名 | 成员类型 | 备注 |
| head | uint32\_t | 大数据包的包头，通常为0xAAAAAAAA |
| cmd | uint8\_t | 大数据包的命令代码 |
| flag | uint8\_t | 大数据包的额外标志位，用于传输成功或失败信息 |
| len | uint16\_t | 大数据包的配置信息，代表更新包文件的有效长度 |
| data | uint8\_t [1008] | 大数据包的配置信息，代表更新包文件 |
| checksum | uint32\_t | 大数据包的校验位，对于数据包进行异或校验得出（包括包头和包尾） |
| tail | uint32\_t | 大数据包的包尾，通常为0xBBBBBBBB |

## 2.3.3 函数设计

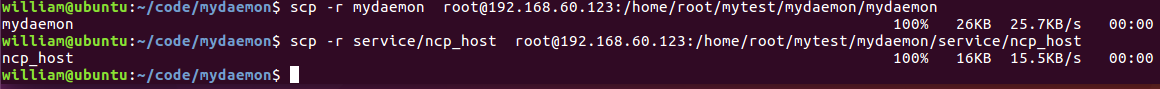
|  |  |  |
| --- | --- | --- |
| 函数名 | 函数类型 | 备注 |
| package\_init\_s | void | 根据传入参数初始化小数据包 |
| package\_init\_m | void | 根据传入参数初始化中数据包 |
| package\_init\_l | void | 根据传入参数初始化大数据包 |
| get\_sock | int | 文件服务器监听指定IP的指定端口，返回-1为监听失败 |
| seek\_sock | int | 网关守护进程连接文件服务器，返回-1为连接失败 |
| get\_checksum | void | 对整个包计算异或校验码，并将计算结果填入 |
| do\_checksum | int | 对整个包计算异或校验码，返回-1位校验失败 |
| transferd | int | 通过读文件描述符和写文件描述符，将数据逐批传输，返回总共传输的数据量 |
| writelog | void | 输出信息至日志文件中 |
| get\_list | int | 解析XML文件，获得需要更新的文件路径数组，返回该数组的有效大小 |
| update | int | 根据提供的路径，对于文件进行备份和替换（XML文件不执行备份），返回0 |
| recall | int | 根据提供的路径，对于文件进行备份回退并删除备份（XML只删除），返回0 |

3 效果演示

# 3.1 编译程序

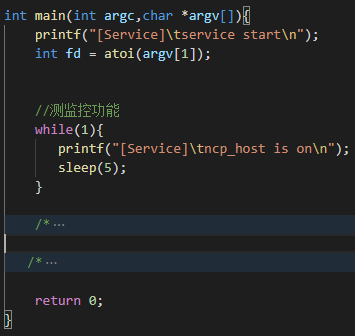


# 3.2 传输文件

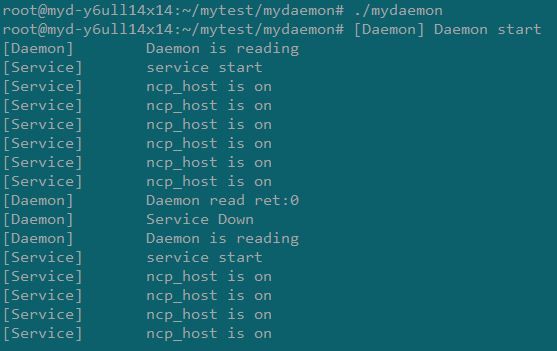


# 3.3 监控功能

编辑ncp\_host.c，编译后导入开发板，并在开发板上运行./daemon

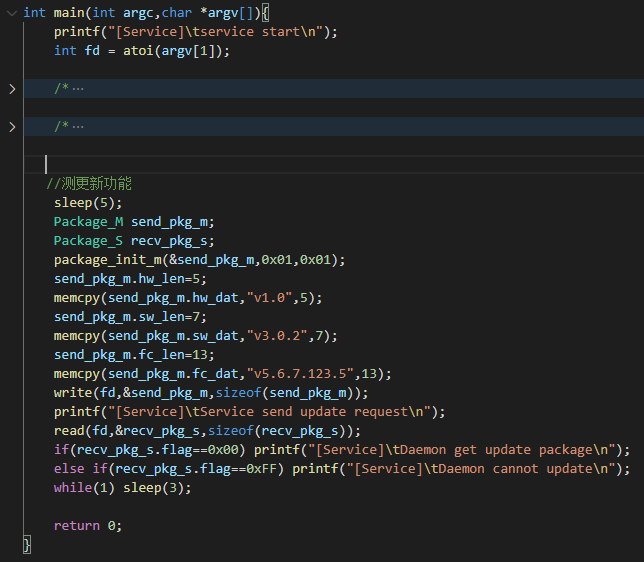


手动杀死网关主程序后，守护进程检测到后再次将网关主程序拉起

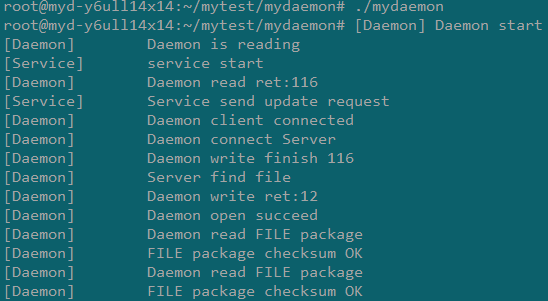


# 3.4 在线更新功能

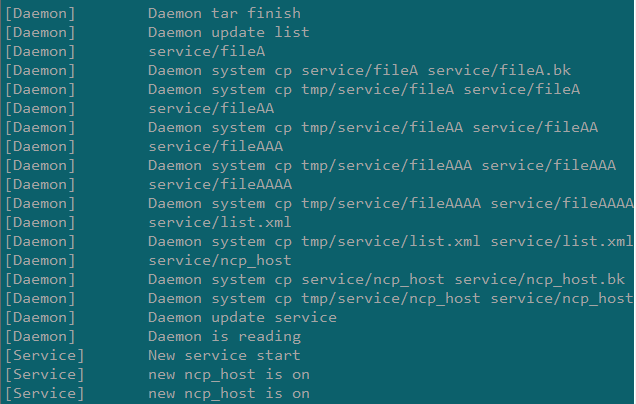
编辑ncp\_host.c，flag参数置为0x01，编译后导入开发板，并在开发板上运行./daemon



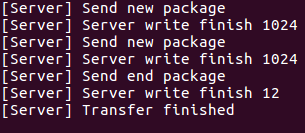
网关守护进程接收到网关主程序的更新需求，向文件服务器发出文件申请请求，在文件服务器成功找到文件后，即可开启文件包的传输



当文件传输完成后，网关守护进程解压文件，杀死网关主程序，替换文件，最后拉起网关主程序



服务端也显示传输完成

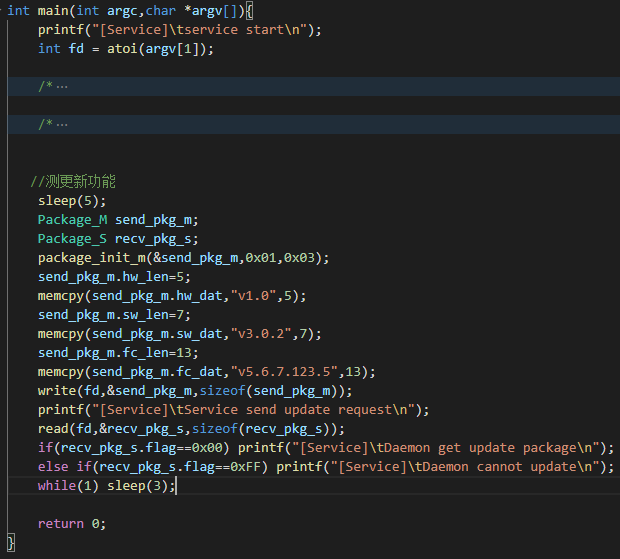


# 3.5 离线更新功能

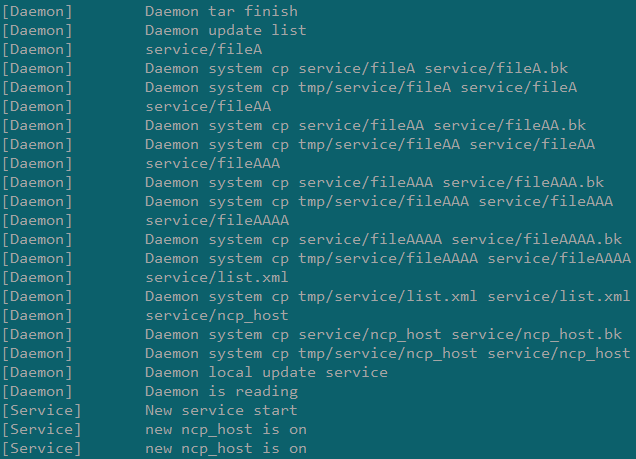
确保mydaemon的工作目录的tmp文件夹内有要更新的文件包service.tar.gz



编辑ncp\_host.c，flag参数置为0x03，编译后导入开发板，并在开发板上运行./daemon

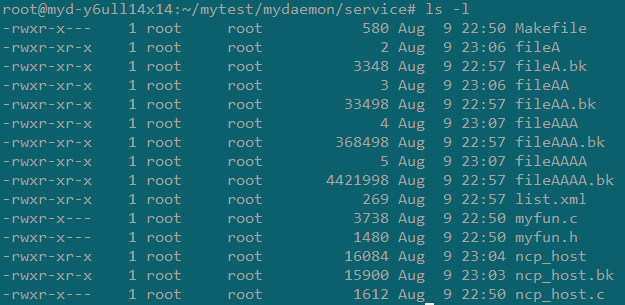


当程序在本地找到压缩包后，网关守护进程进行解压，替换，最后拉起程序

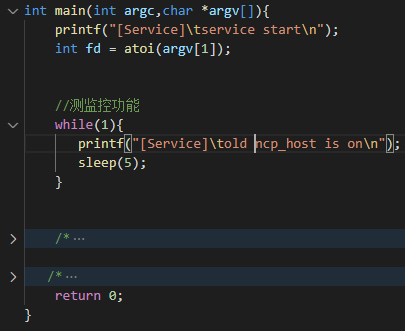


# 3.6 版本回退功能

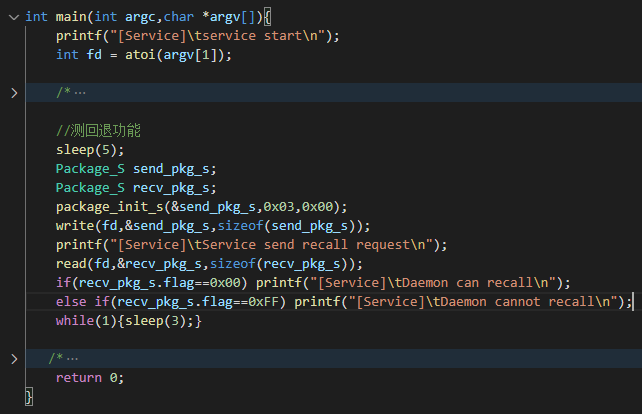
在版本回退前查看service文件夹中的文件详情，确保是在线或离线更新后的文件目录（要有list.xml以及其中指示对应的备份文件）



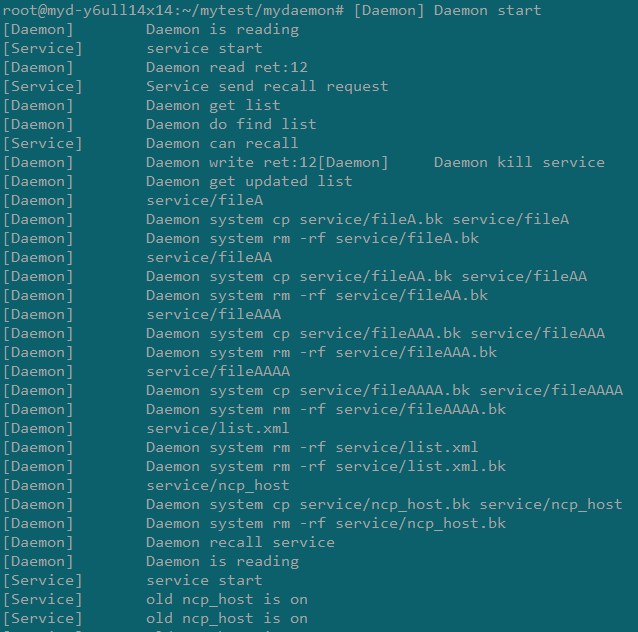
编辑ncp\_host.c，编译后导入开发板作为ncp\_host.bk，用于循环输出，代表回退成功



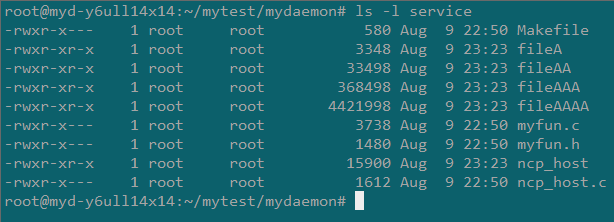
编辑ncp\_host.c，编译后导入开发板作为ncp\_host，用于开启回退流程



网关守护进程成功完成回退



查看回退效果



4 可拓展方面

# 4.1 异常返回值

在整个网关守护进程中，异常返回值的出现，往往会直接伴随着网关守护进程的退出，这一点在调试阶段可以帮助我们更快地定位错误处，但决不适用于直接应用的场景。

因为其存在的风险为：①没有将原网关主程序同步杀死，这样可能会在守护进程重启拉新网关主程序后，同时存在两个网关主程序，导致设备使用流程形成死锁。②网关守护进程异常退出后，网关主程序的更新功能完全失效，此时用户无法执行更新或回退操作，甚至用户可能都无法察觉到守护进程的退出。

解决方案如下：

针对异常状况设计个异常反馈包，在网关守护进程遇到异常后，先发送异常反馈包给网关主程序，再退至外循环，即退至守护进程初始化开始读指令的地方，这样一方面保证守护进程不会随便退出了，也确保用户知晓守护进程操作受阻。

# 4.2 日志功能

在现阶段的网关守护进程中，日志仅实现了基本的将日期以及输出信息写my.log文件，这对于系统性的日志要求是不足的。

而日志的基本要求需要有：①对于日志进行分级，不同级别的日志单独记录。②定时对日志执行清空。

解决方案如下：

使用开源的log4c库，其中设计了日志分级和定向输出相关内容，可以解决不同级别的日志单独记录的需求。对于定时清空，可以在日志程序中设计一个12小时时间戳，每调用一次日志功能，检查当前时间和时间戳的差是否大于12小时，如果大于，则清空内容并更新时间戳，反之则继续写入。

5 实习总结

通过本次实习，我第一次切实投入软件的开发中，经过了接近一个月的开发，我有以下几点实习总结：

1. 前期充分的设计能让工作事半功倍。在本次开发过程中，我前一个版本由于为考虑到实际网络可能存在的丢包问题，直接使用socket进行读写，在流程审批会议上被否定了，导致我不得不返工重写。尽管,1.0版本的开发确实给2.0的代码结构有所铺垫，但在1.0所花费的时间以及精力确实付之东流。所以说如果实现我对于网关守护进程的流程做好充分的设计，大概率可以一次通过审批。
2. 工业标准远高出我的预期。在学校的开发过程中，我们不会去考虑校验，也很少处理异常，但在实际产品的开发中，这些都是必须要做的事情，哪怕相关的工作量与开发原产品的工作量相当。此外，我也认识到了，无论多严谨的系统都多多少少存在一定的漏洞，而以工业标准进行编程，能帮助我们的就是避免漏洞被利用以及减少漏洞带来的损失。
3. 良好的日志习惯将有助于开发。在开发过程中，我经常去打一些输出到控制台来调试，但打着打着，整个屏幕就满是输出，这调试起来就非常麻烦，需要专门去找对应输出。但后来，采用了日志进行输出管理，将一部分输出转入日志，并在日志中加入时间条，此举帮助我可以定位出某些输出发生的前后关系，从而确定程序运行的关系，更好地定位问题所在，所以说，良好的日志习惯可以有助于开发的进展。