|  |  |
| --- | --- |
| **项目名称：** | **InWise 8.0** |
|  |  |
| **文档名称：** | **代码分析报告** |
|  |  |
| **文档编号：** | **CX08-05** |

|  |  |  |  |
| --- | --- | --- | --- |
| **拟制：** | 刘华先 | **日期：** | 2020.9.23 |
|  |  |  |  |
| **审核：** | 刘偊赜 | **日期：** | 2020.9.23 |
|  |  |  |  |
| **批准：** | 刘偊赜 | **日期：** | 2020.9.23 |

修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **版本** | **修订说明** | **修订者** | **修订日期** |
| 1 | v1.0 | 编写InWise 8.0代码分析报告 | 董金彩 | 2020.9.23 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目 录**

[1 引言 4](#_Toc27890)

[1.1 编写目的 4](#_Toc15268)

[1.2 背景 4](#_Toc27549)

[1.3 定义 4](#_Toc5616)

[1.4 参考资料 4](#_Toc31982)

[2 结构分析 4](#_Toc6141)

[1.5 运行环境 4](#_Toc4326)

[1.6 软件结构 5](#_Toc22875)

[1.7 系统数据结构 5](#_Toc29215)

[2 模块1（application migration）分析 6](#_Toc28251)

[2.1 模块描述 6](#_Toc3901)

[2.2 系统结构 6](#_Toc28742)

[2.3 接口 8](#_Toc13524)

[3 模块2（operation development）分析 8](#_Toc22557)

[3.1 模块描述 8](#_Toc25492)

[3.2 数据结构 9](#_Toc18600)

[3.3 流程逻辑 12](#_Toc3256)

[3.4 接口 13](#_Toc2452)

# 引言

## 编写目的

使之成为整个项目开发中系统开发设计的依据，也是项目后期系统测试依据。本文档定义隐患排查操作系统的运行环境、功能需求、性能需求以及各个子系统的功能分析,指导将来系统升级或重写设计及编码。

## 背景

2015年6月2日下午，“国产化操作系统及其产业在国防科技领域的应用”论 坛在北京中华世纪坛成功召开。旨在宣传推广国产化操作系统及其产业在国防 科技领域中的应用，推动中国智能终端操作系统产业联盟成员单位国产化技术、 产品的市场推广。为响应号召，中科红旗投入国产系统的研发行列中，并成功 研发出红旗inWise操作系统。

## 定义

InWise 8.0：红旗公司针对ARM架构，定制研发的操作系统名称。

LFS管理器： 由红旗公司自主研发符合金融服务接口规范的程序。

## 参考资料

I/O APIC：<http://wiki.osdev.org/IOAPIC>

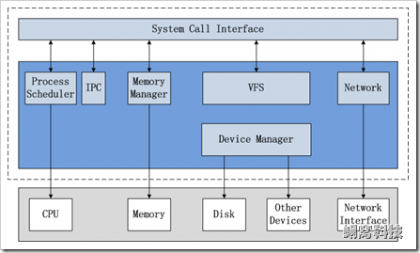
# 结构分析

因为源代码主要是用C/C++，python，shell脚本构成的，使用代码分析工具从代码检查、静态结构分析、代码质量度量等对代码进行分析。代码检查代码检查包括代码走查、桌面检查、代码审查等，主要检查代码和设计的一致性，代码对标准的遵循、可读性，代码的逻辑表达的正确性，代码结构的合理性等方面；可以发现违背程序编写标准的问题，程序中不安全、不明确和模糊的部分，找出程序中不可移植部分、违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

## 运行环境

硬件环境是ARM架构的PC上，在centos7的源码的基础上，交叉编译移植出操作系统InWise 8.0，然后再在此基础上，移植用户要求的软件包及其组件，包括ATM模式切换、LFS管理器和NTFS。代码分析工具均在InWise 8.0上运行。

## 软件结构



用户程序

操作系统

ARM架构的PC

## 系统数据结构

内核通过提供系统调用函数使用户和内核进行交互,在syscall.h中定义了所有的系统调用API。

模块间的内部接口在linux/include/ 或 linux/arch/\*/include/ 下都有定义。

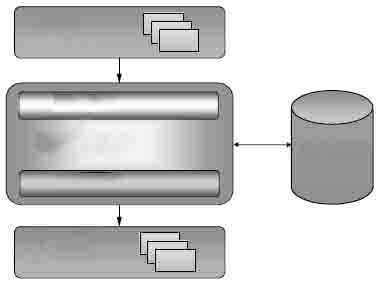
# 模块1（application migration）分析

## 模块描述

模块1主要是代码的编写、阅读和移植，针对客户的需求和实际移植情况，运用代码分析工具对各个应用程序的源代码，根据其功能模块进行结构化的分析。其中涉及到LFS管理器代码的编写，其代码分析如下。

## 系统结构

下图是金融服务扩展（XFS）系统的结构示意图。



**以Windows为基础**

**的应用程序**

#### XFS API

**XFS管理器 配置信息**

**服务提供程序**

应用程序使用API集通过金融服务扩展（XFS）管理器与服务提供程序通讯。大部分API都可以“同步”（管理器使应用程序等待，直到API函数执行完成）或“异步”调用（应用程序立即重新获得控制，与其函数并行执行）。

本金融服务扩展（XFS）规范的所有共同执行工具就是金融服务扩展管理器，这个管理器可以为指定的API找到相应的SPI，然后发送请求到适当的SP。管理器使用配置信息使用API调用（对到“逻辑服务”或“逻辑设备”的）适当的SP（Service Provider）入口点（即使作为最终目标的设备或服务在远程设备上，但入口点总是在本地设备上）。注意虽然API的调用可能是同步的也可能是异步的，但是SPI的调用总是异步的。

金融服务的开发使用经过XFS和可依赖的金融服务周边设备的制造商负责开发和发行的关于服务和设备的SP。每个设备或服务的安装程序也必须定义适当的配置信息。配置信息能允许应用程序在任何时间点上请求关于设备状态的信息以及可使用服务的能力。

SP的主要功能包括以下内容：

* 将通用（即以表单为基础的）服务请求转化为服务专用命令。
* 发送请求到本地服务或设备，或是一个远程系统，在SP中定义一个有效的对等接口。
* 多个应用程序访问一个服务或设备，当请求时提供唯一访问权。
* 管理服务或设备的硬件接口。
* 以适当的方式管理服务和设备的异步性，通过Windows消息XFS管理器和应用程序总是具备此能力。

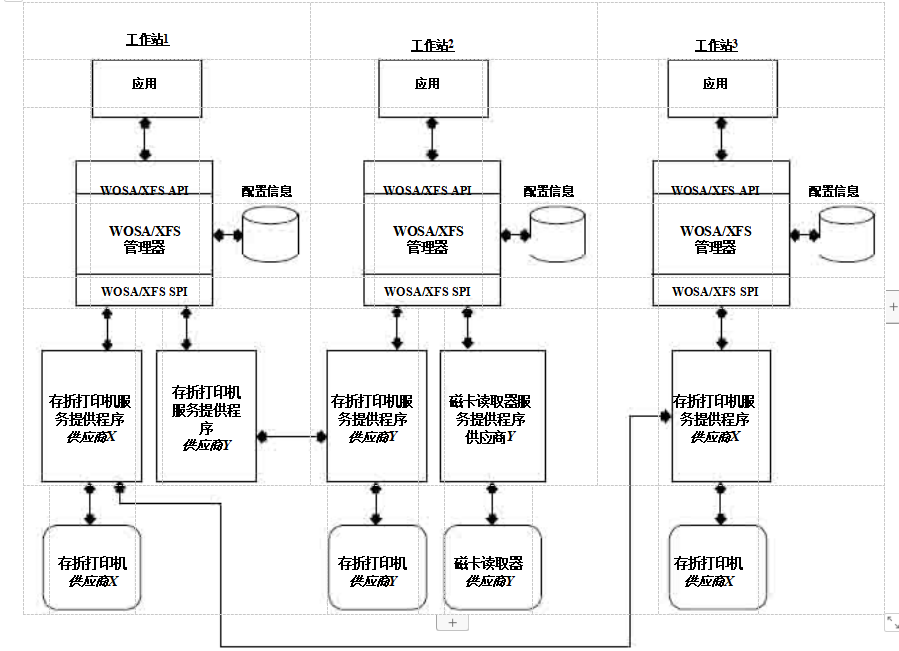
该系统的设计通过在其所有功能方面提供最大的灵活性，来支持对复杂问题的解决方案，通常当前系统不能解决这些问题：

* 多个服务提供程序，由多个厂商开发，能够在同一个单一的系统和网络中共存。
* 服务类别定义是根据服务的逻辑功能作出的，没有考虑实际物理结构。具备多个不同功能（在本规范中称为“复合设备”）的物理设备被视作几个逻辑服务；由服务提供程序解决任何冲突。注意一个逻辑服务可能包含多个物理设备（例如，既有纸币发放功能又有硬币发放功能的自动柜员机）。
* 同样地，一台物理设备可能由两个或三个用户共同使用（即，柜员机），而物理设备的同步是在服务提供程序层上管理的。
* API的定义和相关的服务提供超时功能，从而允许应用程序避免锁死该类型，如果两个应用程序同时都想要对多个服务进行唯一访问，就可能会发生这种锁死的情况。
* 该结构设计用于为将来开发网络和系统监控、措施和管理提供框架。

注意图2.1是该结构的示意图，尤其要指出的是，该图未显示服务提供程序和服务提供程序管理的服务间的区别。本规范的重点在与服务提供程序，而不是服务本身，因为服务提供程序和服务间的通信是开发商的内部设计问题，应用程序和XFS管理器不涉及这些问题。事实上，服务提供程序可以用很多种不同的方式让应用程序能够调用服务。因此，本规范主要关注服务提供程序，因为与XFS管理器进行通信的是服务提供程序。只有在适当的时候才会偶尔提到“服务”。

举例说明

下面的图展示了一个支持一组金融周边设备的XFS系统。注意，在这个框架中，XFS管理器接口直接协调一组直接与物理设备相连的服务提供程序。因此，服务提供程序显示为在执行服务提供程序函数、服务函数和设备驱动函数，尽管这些很可能是在两个或两个以上的单独层上执行的。还有很多可能的其他配置。



## 接口

API结构如下列命令集：

* **基本函数**，如：**StartUp/CleanUp**（启动/清除）、**Open/Close**（打开/关闭）、**Lock/Unlock**（锁定/解锁）和Execute（执行），这些命令是所有金融服务扩展（XFS）设备/服务类别的通用命令；
* **管理函数**，如设备初始化、重置、暂停或恢复，这些命令用于管理设备和服务，以及
* **专用命令**，用来请求服务/设备信息，以及初始化设备/服务专用函数；这些命令将作为**GetInfo**（获得信息）和**Execute**（执行）等基本函数的参数发送给设备和服务。这些服务专用命令由一组单独的规范规定，每个规范包括一个服务类别。

尽最大可能，使用于多个设备/服务类别的具体命令的语法在所有设备上保持一致。主要目的就是为尽可能广泛的设备类型提供标准化的函数代码和函数结构。

SPI尽可能与API保持一致。一些命令专门由XFS管理器处理，因此，不由SPI处理，这样在这两个接口层上传递的具体参数就只有微小的差异。

下面是典型的API使用方案。本例以打印表格所用的函数举例说明。

●**StartUp**（启动，将应用程序连接到XFS管理器，包括版本协商）

●**Open**（打开，在应用程序和服务提供程序间建立会话）

●**Register**（注册，指定应用程序应该从服务提供程序收到的消息）

●**Lock**（锁定，通过应用程序获得对服务的唯一访问）

●多个**Execute**（执行）函数，传递一个或多个具体命令：

●**Print\_Form**（打印表格）

●等等

●**UnLock**（解锁，通过应用程序获得对服务的唯一访问）

●**Deregister**（注销，指定应用程序不再从服务提供程序收到消息）

●**Close**（关闭，结束应用程序和服务提供程序间的会话）

●**CleanUp**（清除，断开应用程序和XFS管理器的连接）

# 模块2（operation development）分析

模块2是在centos7的基础上，交叉编译出openpower服务器上的操作系统，主要涉及五个系统组件。

## 模块描述

主要提供操作系统的基本功能，具体如下：

1. Process Scheduler，也称作进程管理、进程调度。负责管理CPU资源，以便让各个进程可以以尽量公平的方式访问CPU。
2. Memory Manager，内存管理。负责管理Memory（内存）资源，以便让各个进程可以安全地共享机器的内存资源。另外，内存管理会提供虚拟内存的机制，该机 制可以让进程使用多于系统可用Memory的内存，不用的内存会通过文件系统保存在外部非易失存储器中，需要使用的时候，再取回到内存中。
3. VFS（Virtual File System），虚拟文件系统。Linux内核将不同功能的外部设备，例如Disk设备（硬盘、磁盘、NAND Flash、Nor Flash等）、输入输出设备、显示设备等等，抽象为可以通过统一的文件操作接口（open、close、read、write等）来访问。这就是 Linux系统“一切皆是文件”的体现。
4. Network，网络子系统。负责管理系统的网络设备，并实现多种多样的网络标准。
5. IPC（Inter-Process Communication），进程间通信。IPC不管理任何的硬件，它主要负责Linux系统中进程之间的通信。

因为centos7显卡的部分并不支持VGA显示图像功能，所以代码主要涉及修复内核中关于显卡的代码，使得VGA能提供图形显示。

## 数据结构

1. 缓冲部首:

　　linux操作系统为每个缓冲提供了一个缓冲控制块,称缓冲部首。其中保留对应缓冲的地址,对应的设备名,使用情况等定义为charbuffers[NBUF][514];(NBF=15)

缓冲部首说明如下:

　struct buf { int b\_flags ; //标志字位串 struct buf\*b\_forw; //设备(b)链向前指针 struct buf\*b\_back; //设备(b)链向后指针 struct buf\*av\_forw; //空闲(av)链向前指针 struct buf\*av\_back; //空闲(av)链向后指针 int b\_dev; //设备名 int b-wcount; //传送字数(二进制补码) char \* b\_addr; //内存地址低16位 char \* b\_xmem; //内存地址高16位 char \* b\_blkno; //设备上的物理块号 char \* b\_error; //返回I/0出错信息 char \* b\_resid; //因出错而未传送的剩余字数 } buf[NBUF];

　　为了有效地管理这些缓冲,对应的缓冲部首连人到两个双向链中,一个是设备链,或简称b链,它把一类设备所用的所有缓冲连接起来。另一个是空闲链,或简称av链,它把操作系统中所有未用的缓冲连接起来,包括已属于某设备的b链中的缓冲。

1. 设备名:

　　struct{ char d\_minor (次设备号) char d\_major (主设备号) };

　　设备名的高字节存设备类型号d\_major,也叫设备号,表示某一类设备,同类设备使用相同的驱动程序,低字节存设备号d\_minor,也叫次设备号,表示某类设备的第几台设备。

块设备表:含有该类设备的状态信息、IO队列头尾指针和上面提到的该类设备的b链头尾指针。

1. 设备表说明如下:

　　struct devtab { char d\_active; //忙闲标志 char d\_errent; //出错重试次数 struct \*buf b\_forw; //b链链头指针向前指针 struct \*buf b\_back; //b链链头指针向后指针 struct \*buf d\_actf; //IO队列头指针 struct \*buf d\_act1; //IO队列尾指针 };

1. 块设备开关:每类设备自己的管理程序

　　struct bdevsw { int (\*d\_open) ( ); //打开子程序入口地址 int (\*d\_close) ( ); //关闭子程序入口地址 int (\*d\_strategy) ( ); //启动子程序入口地址 int \*d\_tab //对应设备表地址(devtab) } bdevsw;

　　5bfreelist:是av链的链头,

　　用了buf中的四个指针。但b\_forw和b\_back指向的b链不与任何设备相连系。

　　swbuf:专门用于程序对换。

rrkbuf:专门用于原始磁盘操作

1. 关于VGA驱动的改动：在kernel/drivers/video/samsung/x4412\_lcds.c中添加结构体vga\_1440\_900：

/\* VGA-1440X900 \*/

static struct s3cfb\_lcd vga\_1440\_900 = {

.width         = 1440,

  .height        = 900,

         .bpp  = 32,

         .freq  = 60,

         .timing = {

                   .h\_fp = 48,

                   .h\_bp          = 80,

                   .h\_sw         = 32,

                   .v\_fp = 3,

                   .v\_fpe         = 1,

                   .v\_bp          = 17,

                   .v\_bpe        = 1,

                   .v\_sw         = 6,

         },

         .polarity = {

                   .rise\_vclk      = 1,

                   .inv\_hsync      = 1,

                   .inv\_vsync      = 0,

                   .inv\_vden       = 0,

         },

         .init\_ldi = NULL,

};

在结构体数组x4412\_lcd\_config中添加1440\*900的信息：

static struct {

         char \* name;

         struct s3cfb\_lcd \* lcd;

} x4412\_lcd\_config[] = {

         { "ek070tn93",             &ek070tn93},

         { "vs070cxn",               &vs070cxn},

         { "vga-1024x768",       &vga\_1024\_768},

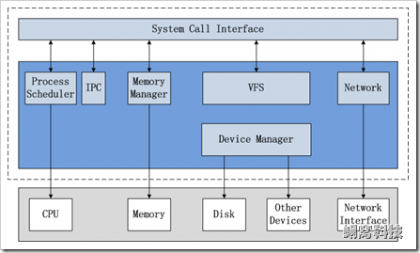
         { "vga-1280x1024",     &vga\_1280\_1024},

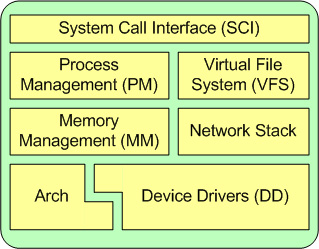
         { "vga-1440x900",      &vga\_1440\_900},

         { "vga-1920x1200",     &vga\_1920\_1200},

};

## 流程逻辑





## 接口

关于内核模块的接口：在./linux/kernel 中可以找到 SCI 的实现，并在 ./linux/arch 中找到依赖于体系结构的部分。内存管理的源代码可以在 ./linux/mm 中找到。文件系统的源代码可以在 ./linux/fs 中找到。内核中网络源代码可以在 ./linux/net 中找到。设备驱动程序的代码可以在 ./linux/drivers 中找到。依赖体系结构的代码可以在 ./linux/arch 中找到。

VGA改进后的接口如下：

在kernel/drivers/video/samsung/s3cfb\_ops.c的s3cfb\_map\_default\_video\_memory函数中，默认存在如下程序：

memset(fb->screen\_base, 0, fix->smem\_len);

       修改为：

memset(fb->screen\_base, 0, fix->smem\_len/3);

       这是内核关于不同分辨率的视频缓存存在的BUG。

       编译内核后，更新内核映像，同时进入uboot后，修改uboot环境变量，执行指令如下：

1. setenv bootargs "lcd=vga-1440x900 tp=ft5x06-1024x600 cam=ov2655 mac=00:09:c0:ff:ee:58"

       这时，在x4412\_lcds.c的lcd\_setup函数会通过uboot传递给内核的环境变量自动匹配LCD参数，真正实现一个内核管理多个LCD屏及多个分辨率的VGA的功能。