# 软件 Demo
# 开发指南文档

**版本号: 0.1**
**发布日期: 2020-09-30**

# 版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
|---|---|---|---|
| 1.2 | 2020.9.28 | allwinnner | 修改文档名称 |
| 1.1 | 2020.9.28 | KPA0527 | 1. 增加接口所属文件<br>2. 修改编译指示 |
| 1.0 | 2020.9.24 | KPA0527 | 1. 增加编译说明 2. 增加接口所属文件 |
| 0.1 | 2020.9.15 | KPA0527 | 建立文档 |

# 目　录

# 插　图

# 1 前言

## 1.1 文档简介

本文档为 P166 项目 Demo 软件的用户开发指南。

## 1.2 目标读者

在 P166 项目软件 Demo 上进行二次开发的开发者和使用者。

## 1.3 适用范围

适用范围为全志 T507 平台。

# 2 模块介绍

## 2.1 模块功能介绍

### 2.1.1 编解码模块

编解码模块的功能包括：数据编码和数据解码。

- 数据编码
  对发送的字符串编码
- 数据解码
  对编码字符串进行解码还原

### 2.1.2 hash 模块

hash 模块的功能包括：字符串 hash 值计算、文件 hash 值计算和 hash 值校验。

- 字符串 hash 值计算
  计算字符串的 hash 值
- 文件 hash 值计算
  计算文件的 hash 值
- hash 值校验
  将原始文件的 hash 值字符串和收到的 hash 值字符串进行比较以确认文件的一致性

### 2.1.3 封装模块

封装模块的功能包括：数据封装和数据解封。

- 数据封装
  将待发送的数据、发送者、接受者和消息类型等信息一起封装起来。
- 数据解封
  对封装数据进行解封、获取发送者、接受者、消息类型和数据内容等信息。

### 2.1.4 netlink 模块

netlink 模块的功能包括：netlink 初始化、netlink 套接字创建、netlink 地址绑定、消息发送和消息接收。

- netlink 初始化
  完成客户端和服务端的 netlink 套接字创建地址初始化与绑定
- netlink 套接字创建
  创建用户态 netlink 地址套接字
- netlink 地址绑定
  将 netlink 套接字与相关的地址进行绑定
- 消息发送
  完成用户态 netlink 消息发送
- 消息接收
  完成用户态 netlink 消息接收

### 2.1.5 连接控制模块

连接控制模块的功能包括：连接确认和 ID 获取。

- 连接确认
  根据客户端的连接请求确定是否进行连接
- ID 获取
  获取客户端的 prot ID 号

### 2.1.6 服务端模块 A

后台服务应用 A，包括三个子功能，分别是连接控制、发送数据、接收数据、HASH 校验模块和文件传输。

- 连接控制
  服务端根据客户端的连接请求消息控制连接是否建立
- 发送数据
  服务端可主动向客户端发送消息
- 接收数据：
  接收来自内核模块 K 转发的数据

- HASH 校验
  根据 HASH 值完成 HASH 校验，如果成功则完成通信，失败则抛出异常码 ERN120 或 ERN110
- 文件传输
  服务端作为被动的文件中转站，客户端可下载服务端的文件，也可以向服务端上传文件

### 2.1.7 内核模块 K

内核模块 K 的功能包括：接收数据、数据转发和状态监测

- 接收数据
  接收来自应用程序的数据，识别出数据的来源和去向

- 数据转发
  如果通信合法，将收到到的数据转发给接收者
  如果通信非法，将数据丢弃并返回非法信息
- 状态监测
  定期对应用程序进程的运行状态进行检测，通过 dmesg 打印出来
- 调试节点内核模块创建了 debugfs 调试节点，可动态观测通信次数

### 2.1.8 客户端模块 BC

客户端的功能包括连接请求、发送数据、接收数据、hash 校验、文件下载和文件上传

- 连接请求
  客户端在与服务端通信之前要发送连接请求进行连接，连接成功方可通信
- 发送数据
  客户端可主动给服务端发送数据
- 接收数据
  接收来自内核转发的数据
- hash 校验
  对文件或消息的 hash 值进行校验确保发送接收无误
- 文件上传
  向服务端上传文件
- 文件下载
  从服务端目录下载文件

# 2.2 相关术语介绍

## 2.2.1 netlink

netlink 是用以实现用户进程和内核进程的一种特殊通信机制，也是网络应用程序与内核通信的最常用的接口。
Netlink 是一种特殊的 socket，它是 Linux 所特有的，类似于 BSD 中的 AF_ROUTE 但又远比它的功能强大，目前在 Linux 中常用在内核空间与用户空间的通信。

## 2.2.2 HASH 校验

散列函数（或散列算法，又称哈希函数，英语：Hash Function）是一种从任何一种数据中创建小的数字“指纹”的方法。散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来。该函数将数据打乱混合，重新创建一个叫做散列值（hash values，hash codes，hash sums，或 hashes）的指纹。散列值通常用来代表一个短的随机字母和数字组成的字符串。
hash 校验即通过比较两个文件或字符串的散列值去判断两者是否相等。

## 2.2.3 客户端和服务端

客户端一般指应用程序 B 和应用程序 C，服务端一般指服务程序 A。

## 2.2.4 内核模块和应用程序

内核模块一般指内核模块 k，应用程序指用户态程序 A、B 和 C。

# 3 模块配置介绍

## 3.1 开发环境配置

本章主要介绍了如何在本地搭建编译环境来编译 Longan SDK 源代码。目前本 Demo 只支持在 linux 环境下编译 Ubuntu 14.04(64 bit)。一个典型嵌入式开发环境通常包括 linux 服务器、Windows PC 和目标硬件板。linux 服务器上建立交叉编译开发环境，为软件开发提供代码更新下载，代码交叉编译服务。

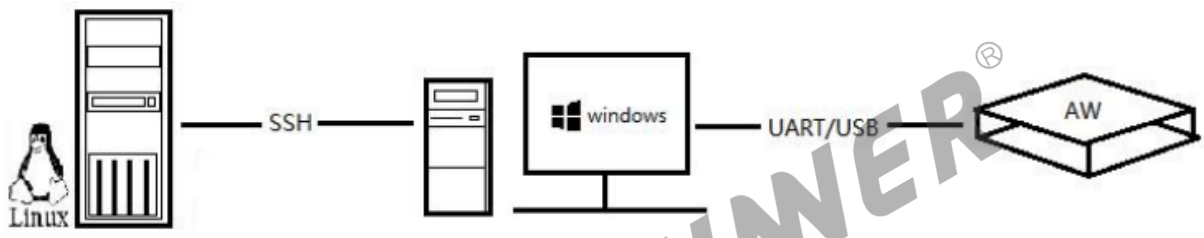


图 3-1: 开发环境搭建

Windows PC 和 Linux 服务器共享程序，Windows PC 并安装 SecureCRT 或 puTTY，通过网络远程登陆到 Linux 服务器，在 linux 服务器上进行交叉编译和代码的开发调试。Windows PC 通过串口和 USB 与目标开发板连接，可将编译后的镜像文件烧写到目标开发板，并调试系统或应用程序。

### 3.1.1 Linux 服务器环境搭建

- 硬件配置
  推荐 64 位系统，硬盘空间大于 30G。如果您需要进行多个构建，请预留更大的硬盘空间。
- 系统版本

```
Linux SzExdroid20 3.19.0-80-generic #88~14.04.1-Ubuntu SMP Fri Jan 13 14:54:07 UTC 2017
    x86_64 GNU/Linux
```

- 网络环境
  安装 nfs、samba、ssh 等网络组件，并自行配置网络。

- 软件包
  配置好网络环境之后，则可以通过如下命令安装编译需要的软件包。

```
sudo apt-get install git
sudo apt-get install gcc
sudo apt-get install wget
sudo apt-get install bison
sudo apt-get install ncurse
sudo apt-get install build-essential
sudo apt-get install zip
```

## 3.1.2 windows PC 开发环境搭建

- 开发工具
  代码编辑：SourceInsight、Notepad++ 等
  串口工具：puTTY、Xshell、mobaxterm 等

- 开发板驱动安装
  一般在 Windows7 的环境下，当目标板设备上电并插上 USB 线之后，会自动安装 USB 设备驱动程序。如果安装成功，则会在 Windows 管理器中出现下图中红色椭圆形标识的设备 Android Phone。
  在 windows10 环境下需要自行安装驱动，驱动目录位于

```
tools/windows 7_10_ _32_64 USB-to-Serial Comm Port/windows 7_ 10
```


图 3-2: 串口驱动

:::note 有些电脑在设备上电并插上 USB 线之后，自动安装 USB 设备驱动程序会失败。推荐使用驱动人生等软件，自动检索安装驱动程序。:::

## 3.1.3 开发板介绍

本 Demo 目前暂时在 T5 平台上运行测试，AW T5 的公版如下

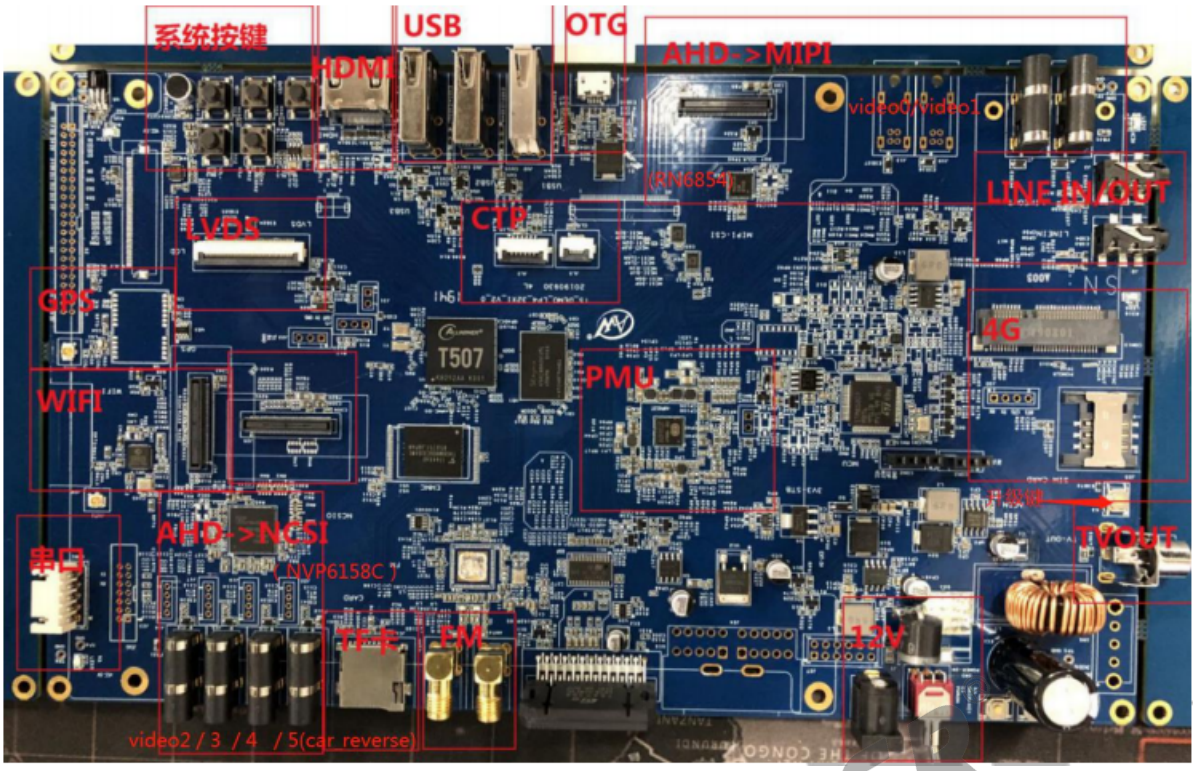


图 3-3: 开发板详情

::: note 本软件 demo 的开发主要关注 DCIN12V(连接 12V 直流电源)，CPU 调试串口，USB-OTG micro 端口 (用于烧录和 ADB 调试)。:::

- 使用准备
  请检查串口硬件工具以及串口连接线、12V 直流电源、以及 miniUSB 线等是否就绪。
- 开发板供电
  请使用 12V 直流电源为开发板供电，供电电流推荐 2A 左右。
- 串口连接
  默认的调试串口用的是 uart0，电压为 3.3v，连接如下图。

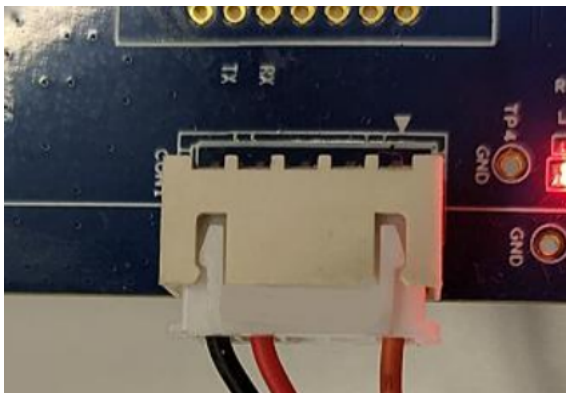


图 3-4: 串口连接

- usb 调试连接
  请使用 USB Micro 数据线，连接开发板和 windows PC 和 usb 端口。

## 3.2 kernel menuconfig 配置说明

关于内核模块随内核编译的配置过程如下，准备 longan 内核代码、将 kernelspace 中的模块代码、Kconfig 和 Makefile 拷贝到内核驱动目录

```
./build.sh config
#按照相关编译环境进行配置
./build.sh menuconfig
#将Device Drivers下的P166_KERNEL_CONNECTION_K选项打开即可随内核编译
```

## 3.3 源码结构介绍

```
├── docs                        项目相关文档目录
├── hdr                         头文件目录
│   ├── codec.h                 编解码头文件
│   ├── connect.h               连接控制模块头文件
│   ├── encapsulation.h         封装模块头文件
│   ├── hash.h                  hash模块头文件
│   ├── netlink.h               netlink模块头文件
│   └── protocol.h              协议相关头文件
├── kernelspace                 内核源码目录
│   ├── kernel_k.c              内核模块k源码
│   ├── Kconfig                 编译相关配置文件
│   └── Makefile                编译相关配置文件
├── lib                         依赖的外部库
│   ├── include
│   └── lib
├── out                         编译产物目录
│   ├── client_b
│   ├── client_c
│   ├── kernel_k.ko
│   └── server_a
├── README.md
├── script                      脚本文件目录
│   └── build.sh
├── src                         源文件目录
│   ├── client_b.c              客户端b源码
│   ├── client_c.c              客户端c源码
│   ├── codec.c                 编解码模块源码
│   ├── connect.c               连接控制模块源码
│   ├── encapsulation.c         封装模块源码
│   ├── hash.c                  hash模块源码
```

```
│   ├── Makefile                编译Makefile
│   ├── netlink.c               netlink模块源码
│   └── server_a.c              服务端a源码
├── tools                       相关编译工具
└── test                        测试文件目录
    ├── codec_test
    ├── file_test
    ├── hash_test
    ├── netlink_file
    ├── pack_test
    └── socket_file
```

# 3.4 编译方式

## 3.4.1 整体编译

下载代码后进入 demo 顶级目录，按照实际编译环境配置好编译工具的环境变量，然后执行编译脚本即可。

```
#内核源码目录
export KDIR=/home/tangjianxiong/work/T507/kunos/kernel/linux-4.9/
export ARCH=arm64
#交叉编译工具名
export CROSS_COMPILE=aarch64-linux-gnu-
#交叉编译工具目录
export PATH=$PATH:~/work/T507/kunos/out/gcc-linaro-7.4.1-2019.02-x86_64_aarch64-linux-gnu/
    bin/
```

编译后的四个产物会出现在 out 目录下，则编译成功，通过 adb 工具推到开发板上即可进行运行调试。

```
.
├── client_b        客户端b
├── client_c        客户端c
├── kernel_k.ko     内核模块k
└── server_a        服务端a
```

## 3.4.2 单独编译应用程序

进入 src/目录, 执行 make 命令。

### 3.4.3 单独编译内核模块

进入 kernelspace 目录，执行 make 命令。

## 3.5 功能使用说明

### 3.5.1 消息功能

加载内核模块，运行服务端 A，运行客户端的消息模式。即可通过命令行发送消息。

```
insmod kernel_k.ko
./server_a
./client_b -m
./client_c -m
```

### 3.5.2 文件功能

加载内核模块，将客户端服务端分别放入各自的工作目录，运行服务端 A，运行客户端的文件模式。即可通过命令行传输文件，默认的顶层目录就是各自的工作目录。

```
insmod kernel_k.ko
./server_a
./client_b -f
./client_c -f
```

# 4 模块接口说明

## 4.1 编解码模块

### 4.1.1 [msg_encode]

```
int msg_encode(const unsigned char *in, unsigned int inlen, char *out);
```

- 所属文件：src/codec.c
- 作用：数据编码
- 参数：
  - 参数 1: 待编码数据指针
  - 参数 2: 待编码数据长度
  - 参数 3: 接收编码后数据的变量
- 返回：
  - 非负整型：编码后数据长度
  - 负数：失败错误码

### 4.1.2 [msg_decode]

```
int msg_decode(const unsigned char *in, unsigned int inlen, char *out);
```

- 所属文件：src/codec.c
- 作用：数据解码
- 参数：
  - 参数 1: 待解码数据指针
  - 参数 2: 待解码数据长度

- 参数 3: 接收解码后数据的变量

- 返回:
  - 非负整型：解码后数据长度
  - 负数：失败错误码

# 4.2 连接控制模块

## 4.2.1 [passwd_vertify]

```
int passwd_vertify(char *passwd, char name);
```

- 所属文件：src/connection.c
- 作用：确认连接口令是否有效
- 参数:
  - 参数 1: 连接口令
  - 参数 2: 连接客户端名称
- 返回:
  - 1：连接成功
  - -1：连接失败

# 4.3 封装模块

## 4.3.1 [pack]

```
int pack(const unsigned char *in, unsigned int inlen, char recv, char send, char msgtype,
    char *out);
```

- 所属文件：src/pack.c
- 作用：数据封装，加入消息头
- 参数:

- 参数 1: 待封装消息
- 参数 2: 待封装消息长度
- 参数 3: 接收者名称
- 参数 4: 发送者名称
- 参数 5: 消息类型
- 参数 6: 接收封装后数据的变量

- 返回：
  - 非负整型：封装后数据长度
  - 负数：失败错误码

### 4.3.2 [unpack]

```
int unpack(const unsigned char *in, unsigned int inlen, char *send, char *msgtype, char *
    out);
```

- 所属文件：src/pack.c
- 作用：数据解封，解析消息字符串、长度、发送者、消息类型
- 参数：
  - 参数 1: 待解封消息
  - 参数 2: 待解封消息长度
  - 参数 3: 接收发送者变量
  - 参数 4: 接收消息类型变量
- 参数 5: 接收解封后数据的变量
- 返回：
  - 非负整型：解封后数据长度
- 负数：失败错误码

# 4.4 hash 模块

## 4.4.1 [hash_str]

```
int hash_str(const char *str, int len, char *output);
```

- 所属文件：src/hash.c
- 作用：计算字符串的 hash 值
- 参数：
  - 参数 1: 待计算的字符串
  - 参数 2: 字符串长度
  - 参数 3: 接收计算的 hash 值
- 返回：
  - 1：计算成功
- 负数：失败错误码

## 4.4.2 [hash_file]

```
int hash_file(const char *str, int len, char *output);
```

- 所属文件：src/hash.c
- 作用：计算文件的 hash 值
- 参数：
  - 参数 1: 待计算文件的文件名
  - 参数 2: 文件名长度
  - 参数 3: 接收计算的 hash 值
- 返回：
  - 1：计算成功
- 负数：失败错误码

# 4.5 netlink 模块

## 4.5.1 [netlink_init]

```
int netlink_init(int id);
```

- 所属文件：src/netlink.c
- 作用：netlink 初始化，完成套接字创建和地址绑定
- 参数：- 参数 1: 端口号
- 返回：- 正整数：成功，返回 socket 描述符 - 负数：失败错误码

## 4.5.2 [netlink_send_message]

```
int netlink_send_message(int sock_fd, const unsigned char *message, int len, unsigned int
    send_pid, unsigned int recv_pid, unsigned int group);
```

- 所属文件：src/netlink.c
- 作用：通过 netlink 向内核发送消息
- 参数：- 参数 1:socket 描述符 - 参数 2: 待发送消息 - 参数 3: 消息长度 - 参数 4: 发送者端口号 - 参数 5: 接收者端口号 - 参数 6: 接收者所在组
- 返回：- 0：成功 - 负数：失败错误码

:::note 参数 5 和参数 6 在向内核发送消息的时候一般是 0，内核默认的端口号和组 id 为 0。:::

## 4.5.3 [netlink_send_message]

```
int netlink_recv_message(int sock_fd, unsigned char *message, int *len);
```

- 所属文件：src/netlink.c
- 作用：通过 netlink 接收来自内核的消息
- 参数：
  - 参数 1:socket 描述符
  - 参数 2: 接收消息变量
  - 参数 3: 消息长度
- 返回：
  - 0：成功
- 负数：失败错误码

## 4.6 二次定制

本 demo 的消息通信、编解码、hash 值计算和消息长度均以解耦方式进行设计，可通过修改接口函数和协议头文件进行二次定制即可。

### 4.6.1 消息通信方式

消息通信方式的二次定制，需要修改 netlink.c 下的相关函数实现，只需将新的通信方法重新修改即可，主要包括初始化、消息发送和消息接收。

### 4.6.2 消息长度

消息的最大长度可在 protocol.h 中修改 MAX_DATA_SIZE 字段即可。

### 4.6.3 编解码方式

修改编解码模块中的 encode 函数和 decode 函数的实现方式即可。

### 4.6.4 hash 计算

修改 hash 模块中的 hash_str 函数和 hash_file 函数实现方式即可

# 5 FAQ

本章介绍软件 demo 在开发运行的过程中调试方法、调试工具和常见问题。

## 5.1 调试方法

### 5.1.1 各模块 netlink 初始化

当 netlink 初始化失败时，会有失败错误打印，根据打印可定位原因。主要分为两类：netlink 套接字创建失败、地址绑定失败。

debug 流程如下图：

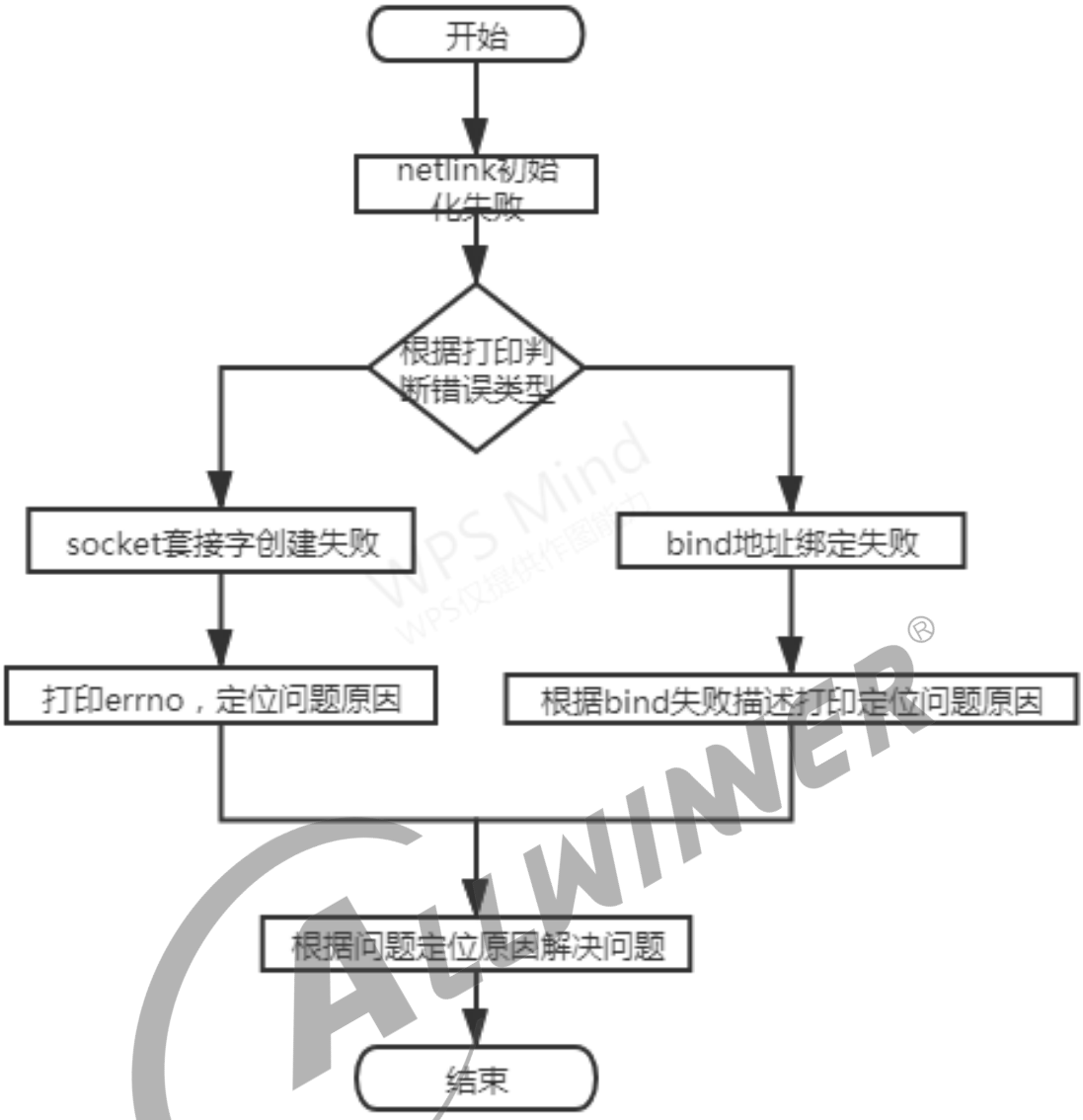


图 5-1: netlink 初始化 debug 流程

- 套接字创建失败

```
create socket error!
[errno]93
```

当套接字创建失败时，会返回相应的错误码，根据错误码可定位创建套接字失败的原因。如上所示，可以通过 errno 值 93 知道，此错误码的原因为 Protocol not supported，根据原因即可定位问题。* bind 地址绑定失败

```
sh-4.4# ./client_b -f
bind: Address already in use
```

当地址绑定失败时，会返回相应的错误码，根据错误码可定位地址绑定失败的原因。如上所示，根据错误打印可知当前地址已经与一个 socket 描述符绑定，不可重复绑定。

### 5.1.2 消息通信 debug

消息通信出问题时，首先需要根据问题场景定位问题所在环节。一般可分为：公共模块问题和客户端与内核模块通信问题。一般 debug 流程如下图。

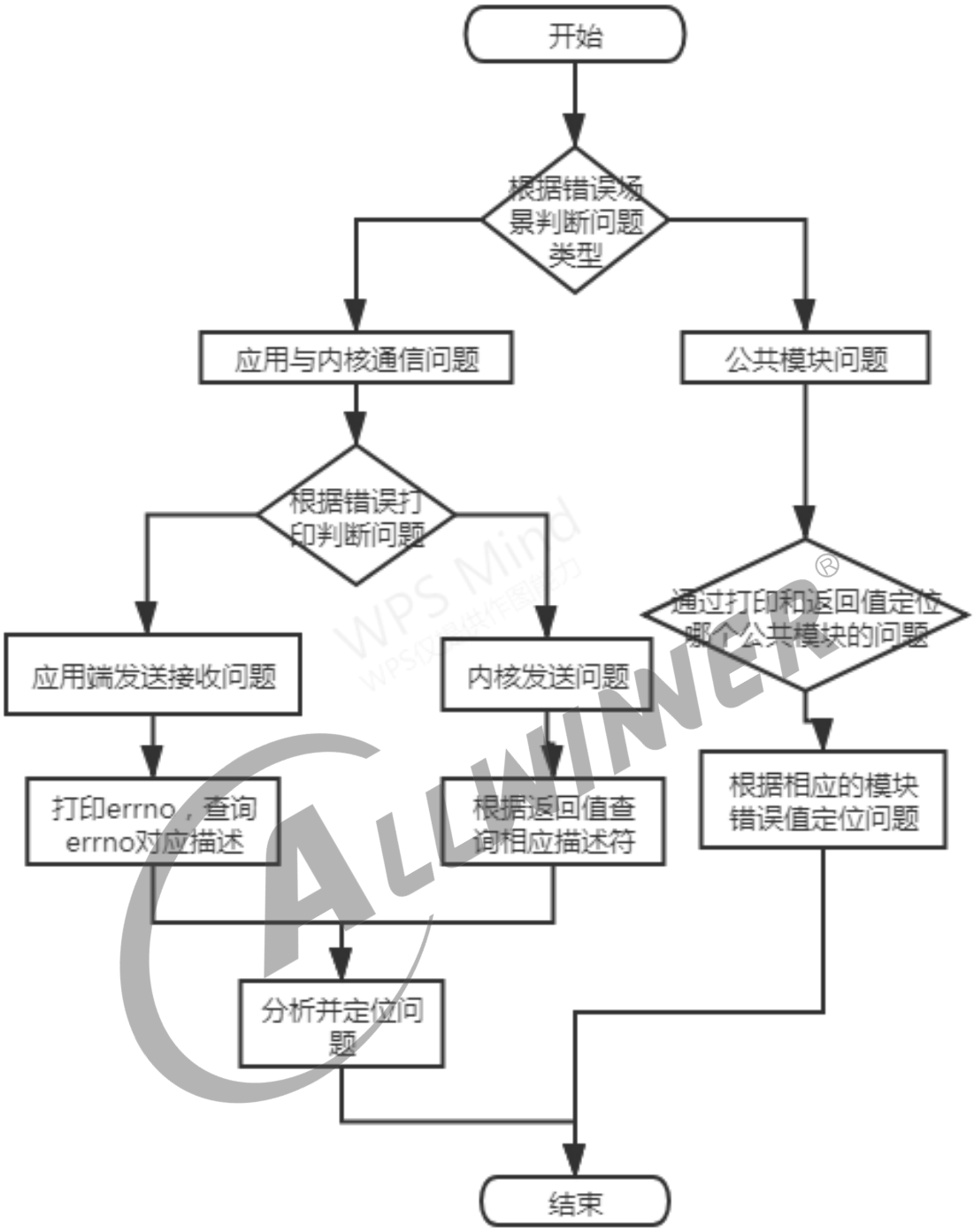


图 5-2: 消息通信 debug 流程

- 公共模块问题公共模块的问题可通过将每一次调用公共模块时的结果和返回值打印出来判断调用情况定位问题。
- 编码和解码功能
  编解码失败时，可以根据编解码函数的返回负值定位失败原因。

- 数据封装与解封
  编解码失败时，可以根据编解码函数的返回负值定位失败原因。

- hash 计算
  hash 计算错误时，可根据 hash 计算函数的返回负值定位失败原因。
- 应用层与内核模块通信问题应用层与内核模块的通信问题主要包括 3 种情况：应用层发送失败、应用层接收失败、内核模块发送失败。根据出错时的打印可判断是内核发送问题还是应用层发送和接收的问题。
- 应用层发送失败应用层发送失败之后，可打印 errno，可根据错误码描述快速定位问题。
- 应用层接收失败应用层接收失败之后，可打印 errno，可根据错误码描述快速定位问题。如下所示，当接收消息失败时，打印对应的 errno，查询得知对应的 errno 描述为 Bad file descriptor，查询资料可定位问题原因 socet 描述符是无效的，根据问题定位可解决问题。

```
[recvmsg error!]
[errno]9
```

- 内核发送失败内核的 netlink 发送函数会直接返回错误码的值，可根据错误码负值，查询源码中对应的原因定位问题。常见的发送失败错误码和原因如下：
  errno -11 内核发送 socket 队列已满，前面的消息应用层还未来得及接收，出现-11 的错误码就是内核发送的太快，应用端接收的太慢，想办法加快接收速度或者减慢发送速度即可解决。
  errno -111 #define ECONNREFUSED 111 /* Connection refused */
  errno -512 ERESTARTSYS 一般是内核发送函数为阻塞式的，当队列满了之后，接收端接收停止，内核模块会一直阻塞在发送消息的命令处，直至应用端被结束，此时就会出现-111 和-512 的错误。解决办法可以给发送消息进行超时处理。

# 5.2 调试工具

## 5.2.1 adb 调试

- adb 简介
  adb 全称为 Android Debug Bridge，是 Android SDK 里的一个工具，用于操作管理 Android 模拟器或真实的 Android 设备。其主要功能有：运行设备的 shell（命令行）；管理模拟器或设备的端口映射；在计算机和设备之间上传/下载文件。
- 运行 adb Windows PC 端的 adb 使用方法和 adb 应用程序，请自行从网络搜索。
- adb 常用命令
- pc 端查看当前连接的设备

```
adb devices
```

- PC 端进入设备 shell 命令行模式

```
adb shell
```

- 将电脑上的文件上传至设备

```
adb push <local path> <remote path>
```

- 下载设备里的文件到电脑

```
adb pull <remote path> <local path>
```

### 5.2.2 GDB 调试工具

开发板中有标准 gdb 工具可以在开发板上进行本地调试。在终端可以直接运行 gdb 来调试程序

```
root@kunos:/# gdb
GNU gdb (GDB) 8.1.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "aarch64-buildroot-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
```

```
Type "apropos word" to search for commands related to "word".
(gdb)
(gdb)
```

## 5.3 调试节点

本 demo 的内核模块创建了 debugfs 调试节点，可通过调试节点查看通信次数、应用运行状态等等信息。挂载 debugfs，进入创建的节点即可进行调试。

```
root@kunos:/root# mkdir debugfs
root@kunos:/root# mount -t debugfs none ./debugfs
root@kunos:/root# cd debugfs/communiation_debugfs/
```

## 5.4 常见问题

软件 Demo 的常见问题及对应的解决方案，具体问题和解决方案如下：1. 启动应用程序时出现 socket 创建失败并报错 errno93。
解决办法：内核模块未加载，检查内核模块是否加载成功，重新加载内核模块。2. 启动应用程序出现初始化失败地址已经被绑定的情况。
解决办法：该程序已经被加载，程序对应的 portid 已经被绑定，检查相关程序的运行情况。

**著作权声明**



**商标声明**



**免责声明**