嵌入式 Linux 系统软件开发经验漫谈

——华恒科技 2006 新员工培训

范 美 辉
<mhfan@hhcn.com>

软件研发部门
华 恒 科 技

2006 年 7 月 20 日

Outline

嵌入式 Linux 开发的特点

- 交叉编译环境
- 嵌入式系统 资源受限！
  - CPU、Memory 资源受限——时间、空间优化
  - 调试方式、分析方法受限——本地开发、调试
  - 使用最少的资源发挥最大的价值
- Linux 是一个开放的平台
  - 开放的网络资源
  - 开放的文档
  - 开放的代码

嵌入式 Linux 开发的特点

- 交叉编译环境
- 嵌入式系统 资源受限！
  - CPU、Memory 资源受限——时间、空间优化
  - 调试方式、分析方法受限——本地开发、调试
  - 使用最少的资源发挥最大的价值
- Linux 是一个开放的平台
  - 开放的网络资源
  - 开放的文档
  - 开放的代码

嵌入式 Linux 开发的特点

- 交叉编译环境
- 嵌入式系统 资源受限！
  - CPU、Memory 资源受限——时间、空间优化
  - 调试方式、分析方法受限——本地开发、调试
  - 使用最少的资源发挥最大的价值
- Linux 是一个开放的平台
  - 开放的网络资源
  - 开放的文档
  - 开放的代码

嵌入式 Linux 开发的特点

- 交叉编译环境
- 嵌入式系统 资源受限！
  - CPU、Memory 资源受限——时间、空间优化
  - 调试方式、分析方法受限——本地开发、调试
  - 使用最少的资源发挥最大的价值
- Linux 是一个开放的平台
  - 开放的网络资源
  - 开放的文档
  - 开放的代码

交叉编译环境

- 交叉编译的概念：
  - 编译机器（build）
  - 宿主机器（host）
  - 目标机器（target）
- 交叉工具链的组成：
  - gcc：C/ C++ 语言的预处理器、编译器和链接器
  - libc：标准 C 库或者 uClibc/newlib 等替代品
  - binutils：处理二进制文件的工具包括汇编器
  - kernel header
  - gdb/gdbserver
- 交叉工具链的构建——crosstool/buildroot

交叉编译环境

- 交叉编译的概念：
  - 编译机器（build）
  - 宿主机器（host）
  - 目标机器（target）
- 交叉工具链的组成：
  - gcc：C/ C++ 语言的预处理器、编译器和链接器
  - libc：标准 C 库或者 uClibc/newlib 等替代品
  - binutils：处理二进制文件的工具包括汇编器
  - kernel header
  - gdb/gdbserver
- 交叉工具链的构建——crosstool/buildroot

交叉编译环境

- 交叉编译的概念：
  - 编译机器（build）
  - 宿主机器（host）
  - 目标机器（target）
- 交叉工具链的组成：
  - gcc：C/ C++ 语言的预处理器、编译器和链接器
  - libc：标准 C 库或者 uClibc/newlib 等替代品
  - binutils：处理二进制文件的工具包括汇编器
  - kernel header
  - gdb/gdbserver
- 交叉工具链的构建——crosstool/buildroot

交叉编译环境

- 交叉编译的概念：
  - 编译机器（build）
  - 宿主机器（host）
  - 目标机器（target）
- 交叉工具链的组成：
  - gcc：C/ C++ 语言的预处理器、编译器和链接器
  - libc：标准 C 库或者 uClibc/newlib 等替代品
  - binutils：处理二进制文件的工具包括汇编器
  - kernel header
  - gdb/gdbserver
- 交叉工具链的构建——crosstool/buildroot

底层开发的特点

- 与硬件结合紧密
- 驱动结构分层：
  - 物理层——体系结构相关、依赖硬件
  - 中间层——相应数据的维护和管理
  - 接口层——内核态与应用层的通信
- 驱动提供给上层机制而非策略
- 修改 Vs. 重写
- 设备驱动 Vs. ProcFS/SysFS

底层开发的特点

- 与硬件结合紧密
- 驱动结构分层：
  - 物理层——体系结构相关、依赖硬件
  - 中间层——相应数据的维护和管理
  - 接口层——内核态与应用层的通信
- 驱动提供给上层机制而非策略
- 修改 Vs. 重写
- 设备驱动 Vs. ProcFS/SysFS

底层开发的特点

- 与硬件结合紧密
- 驱动结构分层：
  - 物理层——体系结构相关、依赖硬件
  - 中间层——相应数据的维护和管理
  - 接口层——内核态与应用层的通信
- 驱动提供给上层机制而非策略
- 修改 Vs. 重写
- 设备驱动 Vs. ProcFS/SysFS

底层开发的特点

- 与硬件结合紧密
- 驱动结构分层：
  - 物理层——体系结构相关、依赖硬件
  - 中间层——相应数据的维护和管理
  - 接口层——内核态与应用层的通信
- 驱动提供给上层机制而非策略
- 修改 Vs. 重写
- 设备驱动 Vs. ProcFS/SysFS

底层开发的特点

- 与硬件结合紧密
- 驱动结构分层：
  - 物理层——体系结构相关、依赖硬件
  - 中间层——相应数据的维护和管理
  - 接口层——内核态与应用层的通信
- 驱动提供给上层机制而非策略
- 修改 Vs. 重写
- 设备驱动 Vs. ProcFS/SysFS

底层开发的特点

- 与硬件结合紧密
- 驱动结构分层：
  - 物理层——体系结构相关、依赖硬件
  - 中间层——相应数据的维护和管理
  - 接口层——内核态与应用层的通信
- 驱动提供给上层机制而非策略
- 修改 Vs. 重写
- 设备驱动 Vs. ProcFS/SysFS

```
#define LCD_REGS_NUM                                (20)
static union {
    u8 __[LCD_REGS_NUM];
    struct lcd_regs {
        // ...  ...
        union {       u8 _;
            struct { u8 hdir:1, vdir:1, mode:2, sel:2, x
            // NOTE: set sel to 0x01 for RGB666
        }   r04;          // 0x0b
        // ...  ...
    }   regs;
}   hhbf_lcdr;

static int lcd_send_cmd(u8 reg, u8 val) {
    u16 cmd = (((u16)reg << 8) | val);
    // ...  ...

```

应用开发的特点

- Linux 系统是 C 语言的虚拟机
- 可移植性 Vs. 代码优化
  - 字节序问题
  - 内存访问地址的对齐问题
  - 编译链接时的符号解析和重定位问题

应用开发的特点

- Linux 系统是 C 语言的虚拟机
- 可移植性 Vs. 代码优化
  - 字节序问题
  - 内存访问地址的对齐问题
  - 编译链接时的符号解析和重定位问题

应用开发的特点

- Linux 系统是 C 语言的虚拟机
- 可移植性 Vs. 代码优化
  - 字节序问题
  - 内存访问地址的对齐问题
  - 编译链接时的符号解析和重定位问题

应用开发的特点

- Linux 系统是 C 语言的虚拟机
- 可移植性 Vs. 代码优化
  - 字节序问题
  - 内存访问地址的对齐问题
  - 编译链接时的符号解析和重定位问题

应用开发的特点

- Linux 系统是 C 语言的虚拟机
- 可移植性 Vs. 代码优化
  - 字节序问题
  - 内存访问地址的对齐问题
  - 编译链接时的符号解析和重定位问题

移植开发的推荐目录布局

- project/
  - project-src/
  - build-x86/
  - build-arm/
  - build-. . . /
- 利用 1ndir 统一代码

移植开发的推荐目录布局

- project/
  - project-src/
  - build-x86/
  - build-arm/
  - build-. . . /
- 利用 1ndir 统一代码

移植开发的推荐目录布局

- project/
  - project-src/
  - build-x86/
  - build-arm/
  - build-. . . /
- 利用 1ndir 统一代码

移植开发的推荐目录布局

- project/
  - project-src/
  - build-x86/
  - build-arm/
  - build-. . . /
- 利用 1ndir 统一代码

use right tool to do right thing!

- 代码维护/版本管理——SVN/CVS/GIT/Mercurial...
- 代码时间性能分析——Oprofile/gprof...
- 代码空间性能分析——valgrind/dmalloc...
- readelf/objdump/nm/size/line...

use right tool to do right thing!

- 代码维护/版本管理——SVN/CVS/GIT/Mercurial...
- 代码时间性能分析——Oprofile/gprof...
- 代码空间性能分析——valgrind/dmalloc...
- readelf/objdump/nm/size/line...

use right tool to do right thing!

- 代码维护/版本管理——SVN/CVS/GIT/Mercurial...
- 代码时间性能分析——Oprofile/gprof...
- 代码空间性能分析——valgrind/dmalloc...
- readelf/objdump/nm/size/line...

use right tool to do right thing!

- 代码维护/版本管理——SVN/CVS/GIT/Mercurial...
- 代码时间性能分析——Oprofile/gprof...
- 代码空间性能分析——valgrind/dmalloc...
- readelf/objdump/nm/size/line...

```
typedef struct __attribute__((packed)) cmv_format_t {
    // ... ...
    union {
        float    fDisplayRatio; // logical format
        uint32_t nDisplayRatio; // storage format
    };
    // ... ...
    union {
        uint64_t full;
        struct { uint32_t low, high; };
    };
    // ... ...
}   cmv_format_t;

```

开发经验总结

- 应用开发——尽量采用本地开发、调试
- 驱动开发——结构规范、接口标准、功能正交
- 注意体系结构的特点：字节序、内存地址对齐
- 链接库的顺序，编译指示，C 语言中联合、位段的运用
- 用正确的工具去做正确的事情
- 参考、阅读开放的文档、代码等资源

开发经验总结

- 应用开发——尽量采用本地开发、调试
- 驱动开发——结构规范、接口标准、功能正交
- 注意体系结构的特点：字节序、内存地址对齐
- 链接库的顺序，编译指示，C 语言中联合、位段的运用
- 用正确的工具去做正确的事情
- 参考、阅读开放的文档、代码等资源

开发经验总结

- 应用开发——尽量采用本地开发、调试
- 驱动开发——结构规范、接口标准、功能正交
- 注意体系结构的特点：字节序、内存地址对齐
- 链接库的顺序，编译指示，C 语言中联合、位段的运用
- 用正确的工具去做正确的事情
- 参考、阅读开放的文档、代码等资源

开发经验总结

- 应用开发——尽量采用本地开发、调试
- 驱动开发——结构规范、接口标准、功能正交
- 注意体系结构的特点：字节序、内存地址对齐
- 链接库的顺序，编译指示，C 语言中联合、位段的运用
- 用正确的工具去做正确的事情
- 参考、阅读开放的文档、代码等资源

开发经验总结

- 应用开发——尽量采用本地开发、调试
- 驱动开发——结构规范、接口标准、功能正交
- 注意体系结构的特点：字节序、内存地址对齐
- 链接库的顺序，编译指示，C 语言中联合、位段的运用
- 用正确的工具去做正确的事情
- 参考、阅读开放的文档、代码等资源

开发经验总结

- 应用开发——尽量采用本地开发、调试
- 驱动开发——结构规范、接口标准、功能正交
- 注意体系结构的特点：字节序、内存地址对齐
- 链接库的顺序，编译指示，C 语言中联合、位段的运用
- 用正确的工具去做正确的事情
- 参考、阅读开放的文档、代码等资源

推荐阅读/参考

- 《深入理解计算机系统》
- LDD2/LDD3
- APUE
- “Linker & Loader”


Thank you!