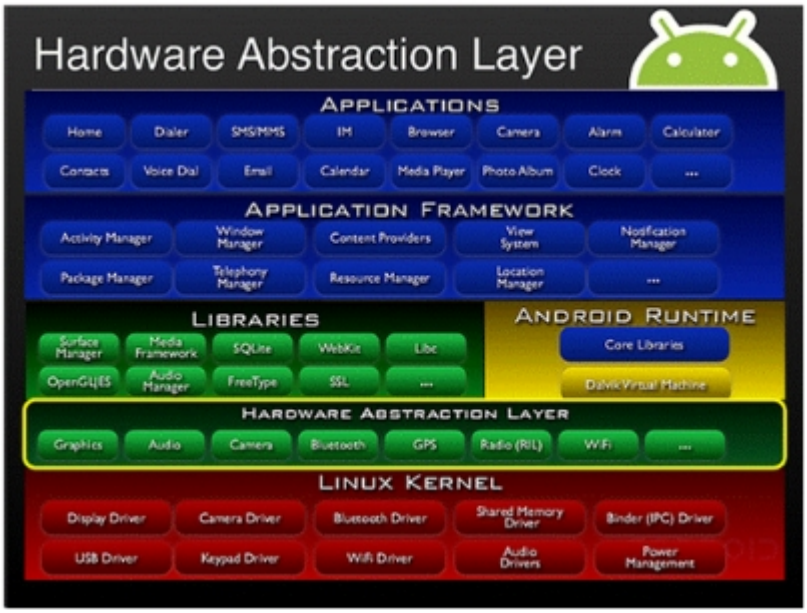
分析基础课程

一：前言

任何软件都可能存在 BUG，调试和修复 BUG 将伴随着整个软件开发流程，因此异常分析变得非常重要，决定软件质量及问题收敛。异常分析是一门需要大量基础知识堆积的学问。因为调试需要对底层运行机制了如指掌，才能找到问题点。目前普通的软件开发人员基本不了解底层运行机制，这也导致了这门课程学习门槛高。

再高门槛的学问都可以一一分解成小模块分步学习。在手机 Soc 系统里，软件是一层层叠加起来的，称为软件栈（software stack），如 android：



异常可能发生任何一层，如果是

- kernel 层发生异常，叫 KE（kernel exception）
- native 层发生异常，叫 NE（native exception）
- java 成异常，叫 JE（java exception）

不同层次的调试稍有不同，我们关注 kernel/native 发生的异常，里面涉及的知识点有共同之处。根据 NE/KE 调试所需知识，我们将异常分析分 3 个部分课程。

课程安排

以下每个部分学习时间为 1.5 个月左右，还有 0.5 个月用于巩固知识和编写学习报告。

- 分析基础课程：里面的知识点都是 KE/NE 都要用到的。我们是基于 ARM CPU 做的 Soc，因此 ARM 架构、指令集、内存架构、ABI 标准是必须的了。ELF 格式是软件编译结果的格式，调试需要用到，因此也需要了解。我们的代码是通过 GNU toolchain 编译的，调试过程中是需要用到的。在做指令层次分析时，对栈布局的掌握至关重要。最后 Mediatek 在 Android 系统集成的 AEE 模块，是异常信息收集系统，有助于我们更便捷的调试。
- KE 分析课程：掌握了基础知识后，只需要了解 panic/HWT 流程，及其他一些调试手段，就可以开始分析 KE 了。

- NE 分析课程：和 KE 差不多，了解 NE 流程，coredump 等，还有 native 基本模块，比如内存分配器（dlmalloc/jemalloc），就可以开始分析 NE 了。

学习成果

每个部分有学习报告，用于评估是否达到预期。

二：ARM 架构和指令集

学习要点

Mediatek 手机芯片是基于 ARM CPU 组成的 Soc，因此任何调试都是基于 ARM。不管是 NE 还是 KE 都需要在汇编层次上调试，因此有必要对 ARM 架构和指令集的了解。调试都需要借助工具，而 NE/KE 的调试工具 gdb 或 trace32 都需要 ARM 汇编基础才行。

ARM 是 RISC 的 CPU，存在大量的寄存器，要看懂汇编代码，就需要对 CPU 有大致的认知，了解寄存器的用途，熟悉常用指令，熟悉异常模型。当然刚开始可能比较生疏，需要经常翻阅 ARM datasheet，之后就可以达到基本不看 datasheet 的境界就完成这堂课的要求了。

目前 ARM 已发展到 64 位（版本是 ARMv8），而一颗 64 位的芯片是同时兼容 32 位和 64 位的，因此都要学习，这样才能从容调试 32 位或 64 位程序。

如果对 CPU 结构和基础知识不了解，建议先阅读课外读物中推荐的计算机结构，有助于学习和理解。

学习材料

ARM 文档中心：http://infocenter.arm.com/help/index.jsp

这是最权威的文档库了。因此要了解最新的最前沿的技术信息，找 ARM 官网就对了。我们需要在里面找 ARMv8 datasheet，位置在：ARM 体系结构 => Reference Manuals => ARMv8-A Reference Manual。需要自己注册一个帐号下载就行（免费），拿到 ARMv8 datasheet。

这是纯英文 datasheet，如果之前很少阅读这种纯英文专业资料，开始可能非常痛苦，大量晦涩专业术语和缺少专业背景知识，但请一定要坚持下去（新技术资料基本都只有英文版的），遇到不懂的请善用搜索和参考课外读物，毕竟 ARM 使用范围很广，参考资料也非常之多，可以借鉴了解。跨过这道坎就比较顺利了。

我们分 2 部分学习，第一部分是架构和指令集，学习所需内容有：

ARM 架构

- A1 章节，架构简介
  - 去除 A1.4.1~7、A1.5、A1.6 小节，这些部分涉及浮点可以忽略。
  - 可结合 64 位 Soc(如 MT6795) datasheet MCU 部分加深了解。
- B1、E1.1~2、E1.5 章节，应用层编程模型
  - 熟悉寄存器用途。
- D1、G1 章节，系统层编程模型
  - 去除 D1.18～19、G1.18 小节。

        - 异常模型，这是重中之重，可结合 linux kernel 异常向量表代码加深了解，代码位置：
            - alps/kernel/arch/arm64/kernel/entry.s
            - alps/kernel/arch/arm64/kernel/traps.c
            - alps/kernel/arch/arm64/mm/fault.c

- alps/kernel/arch/arm/kernel/entry-armv.s
- alps/kernel/arch/arm/kernel/traps.c
- alps/kernel/arch/arm/mm/fault.c

    - Virtualization 了解下就可以，目前没用到（未来可能会用到）。
    - Security 目前已有应用（ATF/TEE），因此也要学习。

指令集

- C1～C6、F1～F4、F6、F7 章节
    - 必须熟记常用的 Load/Store 指令、算术/逻辑指令和跳转指令。
    - 归纳整理所有指令，比如 Load/Store 指令寻址模式等等。
    - SIMD 和浮点相关指令可以忽略。

课外读物

有助于里面 datasheet 里面的背景知识。

- Porting to ARM 64-bit
- ARMv8_white_paper
- ARMv8_Architecture
- Introducing the 64-bit ARMv8 Architecture
- Linux on AArch64 ARM 64-bit Architecture
- ARMv8 与 linux 的新手笔记
- armv8 架構介紹
- 书籍：《大话处理器》
- 国立清华大学开放式课程：《计算机结构》之第 1～21 讲
- linux 异常向量表的设定

学习时间

- 7×8 学时

课后练习

- ARMv8 有几种 Execution state?
- 有几种 Exception levels 且分别做什么？（32 位/64 位都要讲到）
- ARMv8 32 位的两种指令集之间有何区别？
- ARMv8 有哪些通用寄存器分别做什么，32 位/64 位寄存器如何映射？（32 位/64 位都要讲到）
- 讲解 data abort 和 IRQ 发生后 32 位和 64 位模式下的 ARM 的行为。

- Load 指令有几种寻址模式？分别是什么？
- 写一个 Android 上的简单几个函数的可执行程序 test，然后用./prebuilts/gcc/linux-x86/aarch64/aarch64-android-linux-4.9/bin/aarch64-android-linux-objdump -S test，能看懂一个函数对应到的汇编代码(里面包含地址，机器码，汇编代码)。

三：ARM 内存管理架构

学习要点

学习完 ARM 架构和指令集后，对 ARM 已了解了一半，还有一半是内存管理架构，现代 CPU 都有或简单或复杂的内存体系，了解这个体系才能看懂操作系统内存管理模块，才能理解进程空间的本质，才能更好的调试内存相关的问题。

手机 AP 一般用的是 Cortex-A 系列，用的是 VMSA 架构。因此该课程重点是 MMU（内存管理单元），需要熟悉页表结构，Translation Table Walk 过程，cache 则比较次要。

单纯看 datasheet 比较枯燥，在 linux kernel 里有大量操作页表的代码、进程空间描述代码，可以结合着看。另外用户进程空间布局、kernel 空间布局是解决 NE/KE 的基础，这些都可以通过学习 linux kernel 了解。linux kernel 是开源的，网络上同样有大量的参考资料，可以借鉴了解。

学习材料

和'ARM 架构和指令集'课程同样的 ARMv8 datasheet。

- B2、E2 章节，应用层内存模型
  - 简单了解原子操作、内存类型。
- D3～4、G3～4 章节，系统层内存模型和 VMSA
  - 最好结合 kernel 内存管理一起看，看 kernel 如何使用 MMU，比如 vmalloc、ioremap 以及 kernel 如何管理进程空间。
  - MMU 也会发出 fault，请结合'ARM 架构和指令集'学到的异常模型，串联起来。

课外读物

- ARMv8 与 linux 的新手笔记-内存部分
- linux kernel 内存管理代码
- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 基础篇: 通过 log 分析 NE => 用户空间布局
- 国立清华大学开放式课程：《计算机结构》之第 22～26 讲
- cache 操作范围：PoU/PoC
- 硬件自动同步范围：Shareability Domain

学习时间

7×8 学时

课后练习

- 描述虚拟地址到物理地址的映射过程（TTW 的过程）？
- 假设在 kernel 里调用了一个函数，该函数里读取空指针时 ARM 发生了什么，kernel 是如何处理的？32 位和 64 位都讲述下。
- 简述 vmalloc 是如何实现的（进阶问题）？
- 描述下 kernel 内存布局。

四：AAPCS 标准和栈布局

学习要点

前面 2 个课程让我们深刻了解 ARM 的方方面面。大家觉得应该可以开始分析调试 ARM 汇编了。实际上还不够，ARM 汇编是一条条指令组成函数，一个个函数组成一个可执行程序。那么这些指令如何使用寄存器？函数的参数和返回的结果又如何规定？无规矩不成方圆，二进制也有二进制的规则，那就是 ABI。

ABI 全称是 Application Binary Interface，就是用来解决上面的问题的，对应的还有 API。ARM 有自己的 ABI 扩展，其中我们要学习的是 AAPCS（ARM Architecture Procedure Call Standard）和栈布局。

- 熟悉 AAPCS 后可以在调试时反推函数参数，反推寄存器代表的变量，将汇编层次的问题转化为源代码层次的问题，这也是作为系统异常调试工程师和普通软件工程师的差别。
- 每个函数的执行都有对应的栈帧，函数的调用对应的是栈帧的叠加，每个栈帧都可以通过 SP 或 FP 串起来，形成调用栈。知道栈布局，就可以手动推导调用栈，有时工具无法还原调用栈或栈被破坏时，手动还原找回现场将非常重要。

学习材料

- ARM 文档中心 => ARM 软件开发工具 => Application Binary Interface(ABI) for the ARM Architecture => ABI for the ARM 32-bit Architecture => Procedure Call Standard for the ARM Architecture
- ARM 文档中心 => ARM 软件开发工具 => Application Binary Interface(ABI) for the ARM Architecture => ABI for the ARM 64-bit Architecture => Release 1.1 => Procedure Call Standard for the ARM 64-bit Architecture
- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 基础篇: 通过 log 分析 NE => 流程-调用栈

课外读物

- ARM Calling Sequence Specification (Windows CE 5.0)
- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 进阶篇: coredump 分析 => AAPCS 标准

学习时间

3×8 学时

课后练习

- 一个函数有 9 个参数，请问在 32 位和 64 位下是如何传递的？请画出压栈图。
- 简单写一个 test1 函数调用 test2 函数，test2 函数调用 test3 函数，test3 函数调用 test4 函数的 Android native 程序。当运行到 test4 时，请画出栈布局。

五：ELF 格式

学习要点

ELF 是什么东西？它是一种文件格式，类似 window 上的 PE，规定了*.exe 等文件的格式，而 ELF 是 linux 上的可执行/链接文件格式。ELF 全称是 Executable and Linking Format。作为系统异常调试工程师，已熟悉了 ARM 汇编，还要了解这些指令是怎么存储在文件里的。一般情况下调试需要对应的 symbol 文件(包含调试信息的 elf 文件),symbol 文件里面包含了汇编到源文件的映射关系,则就是所谓的 debugging info，比如地址和文件名/行号的关系。任何异常调试都是从汇编层次转化为源代码级别调试，这个过程就需要借助 debugging info。

ELF 本身是一个容器，里面可以放二进制指令，也可以放字符串等资源。如何解读它呢？可以直接用 UltraEdit 等工具查看，当然我们有更好的工具，GNU tool chain 里的 readelf（该工具将在下一章节介绍）可以帮助你了解 elf 文件包含的内容。使用使用方法如下：

./prebuilts/gcc/linux-x86/aarch64/aarch64-linux-android-4.9/bin/aarch64-linux-android-readelf -a -W vmlinux > vmlinux.log。那么 vmlinux.log 就是 vmlinux 这个 elf 文件的解读了。

学习材料

- ELF Specification v1.2
- ARM 文档中心 => ARM 软件开发工具 => Application Binary Interface(ABI) for the ARM Architecture => ABI for the ARM 32-bit Architecture => ELF for the ARM Architecture
- ARM 文档中心 => ARM 软件开发工具 => Application Binary Interface(ABI) for the ARM Architecture => ABI for the ARM 64-bit Architecture => Release 1.1 => ELF for the ARM 64-bit Architecture (AArch64)

课外读物

- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 扩展篇：编译与加载 => ELF/coredump 结构、ELF 加载执行

学习时间

8 学时

课后练习

- 讲解下 section 和 segment 的区别？
- dwarf debugging info 放在哪里？都有哪些？

六：GNU Toolchain/Trace32

学习要点

工欲善其事，必先利其器。有工具借助，调试效率可以提升很多。这里介绍 2 套工具，GNU toolchain 和 Trace32。

GNU toolchain

Android native 和 kernel 就是用 GNU toolchain 里的 gcc 编译的，而 GNU toolchain 可不止有 gcc，还有调试用的 gdb，解读 elf 的 readelf 等等，这些都要掌握。特别是 gdb，要熟悉常用的命令，比如 bt、thread、info 等等，这样调试起来才能得心应手。

GNU toolchain 在网络上有非常多的资料，大家善用搜索工具即可。toolchain 放在 codebase 里的 prebuilts/gcc/linux-x86 目录下，分别有 arm（arm 32 位）和 aarch64（arm 64 位），比如 aarch64/aarch64-linux-android-4.9/bin 目录，里面就有一大堆工具。你也可以自己下载 NDK，里面有 windows 版本的 toolchain。

Trace32

Trace32 在调试行业里最强大的工具了，图形化界面，对于不喜欢 gdb 的命令行的工程师来说，Trace32 是很好的选择。不过 Trace32 只有 ARM 32bit 是免费的。

Trace32 支持强大的 PRACTICE 脚本和丰富的命令，可以搭建静态分析环境。

学习材料

- GDB 使用文档
- GNU Binary Utilities（包含 readelf、nm、objdump 等）
- DCC => Debugger_User Guide_v5.1.docx => 4.4 GDB
- MediaTek On-Line => Trace32 使用教程
- Trace32 官网 => Support => E-Learning => Introduction to TRACE32 GUI

课外读物

- Trace32 官网 => Support => Download Center => Trace32 Manuals Updates => HELP.ZIP
    - 里面有非常多的文档，让你了解 trace32 的方方面面。了解后可以根据自身的需求客制化工具。

学习时间

2 × 8 学时

课后练习

- 找 2 个 KE db（分别是 32 位和 64 位系统触发的 KE），分别用 gdb 和 trace32 调试，然后解决问题。

七：logging/AEE

学习要点

之前学习的知识让你有基本的调试能里，和 Android 没有多大关系，但从这里开始就需要学习 Android 一些基础知识了，毕竟我们是在 Android 上调试的。

调试有分 online 调试和 offline 调试，online 调试有：gdb、jtag。offline 调试有：log、gdb、trace32。了解 log 和知道 log 如何被打印出来，就可以知道代码跑到哪里，流程是否正常。当发生崩溃后，Mediatek 有 AEE 系统收集异常信息，并打包生成 db，而 db 是我们 offline 调试的材料，用 gdb 或 trace32 就可以调试 db 里的 coredump 文件。

GAT 是 Mediatek 扩展了 DDMS 而来，集成了 db 解压，log 浏览等常用功能，熟悉它非常必要。

学习材料

- DCC 上搜索 MediaTek Logging SOP.pptx
- DCC 上搜索 Android_log_debug_for_customer.pptx
    - 文档比较旧，但有些东西还是值得参考的。
- DCC 上搜索 GAT_User_Guide(Customer).docx
- MediaTek On-Line => Quick Start => Deep in MTK Turnkey Solution Logging Tools

课外读物

- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 基础篇：通过 log 分析 NE => 流程-产生 db

学习时间

8 学时

课后练习

- gdb 或 trace32 能调试 KE 和 NE db 里的哪些文件？
- kernel 是通过什么函数打印 log 的？如何能看到它？

八：学习报告

到这里第 1 阶段的学习就告一段落了。现在的你具备基础的异常分析能力。当然我们要检验下是否达到预期。学习报告是合适的选择，除了可以呈现所学知识，还可以巩固所学知识。

学习报告的内容来源之前所学知识，可以选择 1 个主题来完成你的报告：

- ARM 架构、指令集、AAPCS 标准和栈布局
- ARM 内存管理架构
- ELF、GNU Toolchain/Trace32、logging/AEE

之后就开始新的课程，学习不同软件层的调试技巧了。

KE 分析课程

一：KE 简介

掌握里基本分析方法，那不同的软件层异常调试就需要对那层软件的了解了。我们先从底层开始，最底层是 kernel，kernel 发生崩溃，我们叫它为 KE（Kernel Exception）。我们如何调试 KE 呢？首先你得了解 KE 的流程，之前学 ARM 时，知道访问非法地址将产生 MMU fault，会触发 abort，那么 kernel 如何处理 abort？这个就是我们学习的内容了。除了 abort 异常外，kernel 还有一类特殊的异常，就是看门狗复位了，系统如果卡住一段时间，我们认为是有问题的，需要及时发现并解决，看门狗可以发现这样的问题。我们要分析解决看门狗问题，就需要了解看门狗异常流程。

都知道 KE db 是分析 KE 的材料，但不清楚 db 里的文件是如何产生的，这时你就需要查看 AEE 驱动了，它能让你对 db 里的信息来源更加了解。甚至可以客制化添加自己的信息。

为了方便调试，kernel 里集成了一些调试模块，熟悉这些模块才能知根知底，用起来随心应手。

最后是 kernel 架构，kernel 内容太多了，但你对 kernel 的熟悉程度决定你调试问题的效率。比如一个 crash 出现在 hrtimer 里，如果熟悉 hrtimer 原理，那么调试起来就比较顺利了。

在 MOL 上有篇文章由浅入深讲解如何调试 KE，请按照该篇贯穿 KE 学习路线。

- MediaTek On-Line => Quick Start => 深入分析 Linux kernel exception 框架

二：panic 流程/调试

学习要点

调试的本质就是将汇编层次的问题转化为源代码级别的问题，然后根据你的背景知识将其解决。当没有 KE db 时，我们能分析就是 kernel log，而 kernel panic 流程就是将异常信息输出到 kernel log，所以我们要结合 kernel log 来看 kernel panic 流程。

学习材料

- kernel/arch/arm/kernel/entry-armv.s、kernel/arch/arm/kernel/entry.s、kernel/arch/arm/mm/fault.c、kernel/arch/arm64/mm/fault.c、kernel/arch/arm/kernel/traps.c、kernel/arch/arm64/kernel/traps.c、kernel/kernel/panic.c 等 kernel oops 代码
- MediaTek On-Line => Quick Start => 深入分析 Linux kernel exception 框架 => 基础篇：通过 log 分析 KE

课外读物

无

学习时间

2×8 学时

课后练习

- 画出 kernel oops 到重启的代码流程图。

三：HWT 流程/调试

学习要点

HWT 全称是 Hardware Watchdog Timeout。这个看门狗是防止 kernel 卡死的模块，如果发生了 HWT，表示 kernel 有被卡住，这会影响到性能，甚至可能变成卡死而重启。

学习材料

- MediaTek On-Line => Quick Start => 深入分析看门狗框架

课外读物

无

学习时间

2×8 学时

课后练习

- 独立解决 1 个 HWT db 的问题。

四：AEE 驱动模块

学习要点

之前讲过 KE/NE db 都是 aee 打包生成的，它是一套机制，其中在 kernel 里有 aee 驱动，该驱动插入了 kernel oops 和 panic 流程，因此在学习 panic 流程时，需要看下 aee 驱动到底做了什么。对调试有帮助。

当然了 AEE 驱动还不止对 oops/panic 流程的扩展，还有包含：

- SWT 卡死触发 HWT 的机制
- ramdump 功能
- HWT 机制

学习材料

- kernel/drivers/misc/mediatek/aee 代码
  - mrdump 目录：ramdump 功能。
  - aed/monitor_hang.c：SWT 卡死触发 HWT 的机制。
  - common/wdt-atf.c、common/wdt-handler.c：HWT 机制。
- bootable/bootloader/lk/app/mt_boot/aee 目录
  - ramdump 的 lk 部分，用于抓取 DRAM 上的资料并保存在 emmc 上。

课外读物

无

学习时间

2×8 学时

课后练习

- 画出 kernel oops aee 流程。

五：内建调试模块

学习要点

kernel 中有些模块是用于 debug 用的，有助于我们的分析，比如 ram console，是 last kmsg 的驱动，里面除了存储 kernel log 外，还有很重要的信息，比如 fiq step、low power 相关的标志。这些信息可以让我们知道异常时刻系统的状态。

学习材料

- kernel/drivers/misc/mediatek/ram_console/代码
- MediaTek On-Line => Quick Start => 深入分析 Linux kernel exception 框架 => 基础篇: 通过 log 分析 KE => ram console

课外读物

无

学习时间

2×8 学时

课后练习

- 讲解 ram_console 头部存储 reboot reason 的信息含义。

六：kernel 架构

学习要点

KE 可能发生在任何地方，而如果你不熟悉那个地方的架构，调试起来将非常困难，因此熟悉 kernel 的程度是提升调试能力的重要指标。

必须要熟悉的模块是：

- 内存管理
  - buddy system、slub、vmalloc、percpu
- timer、hrtimer
- irq
- workqueue

这些模块都是 kernel 比较常见的，熟悉这些模块，不仅有助于调试，还可以写出优秀的内核代码。

学习材料

- kernel 代码
- 书籍：《深入 linux 内核架构》

课外读物

- linux kernel 内存管理入门笔记

学习时间

长期

课后练习

无

七：学习报告

现在你已经有能力分析 KE 问题了，差的就是一些经验和对 kernel 的熟悉程度了。当然报告还是需要的，可以选择 1 个主题来完成你的报告：

- kernel panic 机制及一次 KE 实例讲解
- HWT 机制和分析

NE 分析报告

一：NE 简介

Native 属于 linux 再上一层，可以直接运行于 linux 的那一层，用于区别 java 层。native 层是由各种 lib/binary 组成。同样这一层也会出现这种异常，我们称为 NE。NE 的全称是 native exception。

对于应用程序的调试，我们可以 online 调试也可以 offline 调试。不过一般我们用的是 offline 调试，通过 coredump 借助 gdb 或 trace32 来调试。当然在讲 coredump 前，要先熟悉下 linux 信号和 ptrace 机制，coredump 是通过信号触发生成的。coredump 是进程空间保存到文件系统的镜像，因此能看到异常时刻的所有变量值，就可以知道问题出在哪里。

Android 是基于 linux 的，发生异常时，Android 扩展了调试机制，这个机制是 debuggerd 机制。在没有 coredump 下，debuggerd 以 log/tombstone 的方式输出异常信息，以便后面调试。

了解了这些信息，基本就可以开始调试了，但内存问题（踩坏、泄漏）还需要你熟悉内存分配器原理才行，否则就无从下手了。

最后是进阶课程：栈回溯。在'分析基础课程'里有学到栈布局，这个是它的扩展，在有 frame pointer 情况下是比较简单的，但 native 程序和库基本不支持 frame pointer，栈回溯就比较麻烦了，为此 GNU 和 ARM 都有自己的标准来完成栈回溯功能。

在 MOL 上有篇文章由浅入深讲解如何调试 NE，请按照该篇贯穿 NE 学习路线。

- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架

二：linux 信号/ptrace

学习要点

当进程发生崩溃时，kernel 会以信号的方式通知进程，每个信号伴随着动作，比如产生 coredump，或终止程序，具体定义就看是什么信号了。

ptrace 则用于跟踪调试进程的，通过 ptrace 可以获得目标进程的 CPU 寄存器，进程空间的任何内存内容。

了解了信号和 ptrace 将更容易理解后面的内容：coredump 和 debuggerd。

学习材料

- 书籍：《UNIX 环境高级编程 第 3 版》第 10 章：信号
- linux man pages-ptrace
- wiki-ptrace
- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 编程篇: linux c 编程

课外读物

无

学习时间

3×8 学时

课后练习

- 列出哪些信号会产生 coredump。
- 往 mediaserver 发送信号 5，查看表现。
- 编写一个程序，通过 ptrace 获取目标进程的 CPU 寄存器。
- 编写一个程序，主动触发 NE 的几个信号。

三：coredump 机制

学习要点

coredump 是分析 NE 的材料，也是最完整的材料，除了能从调用栈直接看出问题外，基本上 NE 调试都需要 coredump。

学习材料

- man pages - coredump
- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 基础篇: 通过 log 分析 NE => 流程-产生 db

课外读物

- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 扩展篇: 编译与加载 => ELF/coredump 结构

学习时间

8 学时

课后练习

- 通过设置，发送信号给 mediaserver，让 coredump 生成在/data/目录下。

四：debuggerd 机制

学习要点

Android 实现了进程崩溃异常信息抓取机制，通过一个叫 debuggerd 的守护进程完成。

学习材料

- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 基础篇: 通过 log 分析 NE => 流程-debuggerd
- bionic/linker/debugger.cpp
    - 注册会引起崩溃的信号，并通知 debuggerd。
- system/core/debuggerd/代码
    - 收到通知后，通过 ptrace 接上崩溃进程，然后导出异常信息，生成 tombstone。

课外读物

无

学习时间

2×8 学时

课后练习

- 画出 L 版本的生成 tombstone 的 debuggerd 流程图。

五：malloc 原理

学习要点

你现在已可以分析 NE 的问题了，但是内存问题（泄漏、踩坏）的问题，还比较困难，还需了解 malloc 分配器内部结构才行。

Android 有 2 个分配器可选择，L 版本之后默认是 jemalloc，之前一直用的是 dlmalloc。

Android 还开发了一些 malloc 调试的功能，Mediatek 也添加了调试功能，这些都要掌握，调试起来才不费力。

学习材料

- bionic/libc/upstream-dlmalloc/代码
    - dlmalloc 源代码
- bionic/libc/bionic/malloc_debug_*.cpp
    - android 集成的 malloc 调试模块
- external/jemalloc/代码

- DCC => Debugger_User Guide_v5.1.docx => 4.1 Bionic Debug Mode

课外读物

无

学习时间

6×8 学时

课后练习

- 画出 jemalloc/dlmalloc 的分配内存流程图。
- 找一个内存泄漏和踩坏的例子并解决掉。

六：Native 基础模块

学习要点

NE 可能发生在任何地方，而如果你不熟悉那个地方的架构，调试起来将非常困难，因此熟悉 native 的程度是提升调试能力的重要指标。

必须熟悉的模块是：pthread、systemproperty，这些都是 native 比较常见的模块。还可以了解下 linker。熟悉这些模块，不仅有助于调试，还可以写出正确的 native 代码。

学习材料

- bionic/libc/代码
  - 里面包含 pthread、property 代码。
- POSIX thread
- bionic/linker/代码
  - 了解一个程序如何被加载和链接。

课外读物

无

学习时间

10×8 学时

课后练习

- 编写程序，里面包含创建线程和读写 property 代码，并可正确运行。

七：栈回溯机制

学习要点

这个是提升 NE 分析能力的课程。正常情况下，我们借助 gdb 或 trace32 工具可以轻松还原出调用栈。但任何情况下都不能太依赖工具，有时工具无法还原调用栈，这时就需要手动还原了。在'分析基础课程'里有学过栈布局，知道如何推导有 FP 的调用栈，而这里教你如何推导没有 FP 的调用栈。

ARM 有 exidx，GNU 有 eh_frame，而通用的调试格式 dwarf 有 debug_frame。有了这些信息，就可以轻松推导调用栈了。而 gdb 或 trace32 正是借助这些调试信息来还原调用栈的。

学习材料

- MediaTek On-Line => Quick Start => 深入分析 Android native exception 框架 => 基础篇: 通过 log 分析 NE => 流程-调用栈
- ARM 文档中心 => ARM 软件开发工具 => Application Binary Interface(ABI) for the ARM Architecture => ABI for the ARM 32-bit Architecture => Exception Handling ABI for the ARM Architecture
    - ARM exidx 资料
- DWARF 4
- ARM 文档中心 => ARM 软件开发工具 => Application Binary Interface(ABI) for the ARM Architecture => ABI for the ARM 32-bit Architecture => DWARF for the ARM Architecture
- LSB - eh_frame

课外读物

- external/libunwind/代码

学习时间

长期

课后练习

无

八：学习报告

现在你已经有能力分析 NE 问题了，差的就是一些经验和对 native 的熟悉程度了。当然报告还是需要的，可以选择 1 个主题来完成你的报告：

- debuggerd 机制及一次 NE 实例讲解
- jemalloc 剖析
- dlmalloc 剖析