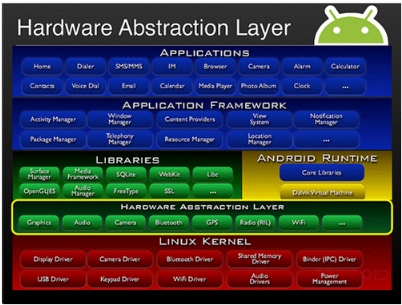
#### 分析基础课程

#### 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](http://community.arm.com/docs/DOC-8453)
* [ARMv8\_white\_paper](http://www.arm.com/files/downloads/ARMv8_white_paper_v5.pdf)
* [ARMv8\_Architecture](http://www.arm.com/files/downloads/ARMv8_Architecture.pdf)
* [Introducing the 64-bit ARMv8 Architecture](http://andrew.wafaa.eu/files/EuroBSDConARMv8.pdf)
* [Linux on AArch64 ARM 64-bit Architecture](https://events.linuxfoundation.org/images/stories/pdf/lcna_co2012_marinas.pdf)
* [ARMv8与linux的新手笔记](http://loda.hala01.com/2014/12/armv8-%E8%88%87-linux%E7%9A%84%E6%96%B0%E6%89%8B%E7%AD%86%E8%A8%98/)
* [armv8架構介紹](http://wiki.csie.ncku.edu.tw/embedded/ARMv8#armv8%e6%9e%b6%e6%a7%8b%e4%bb%8b%e7%b4%b9)
* 书籍：《大话处理器》
* 国立清华大学开放式课程：《[计算机结构](http://ocw.nthu.edu.tw/ocw/index.php?page=course&cid=76&)》之第1～21讲
* [linux异常向量表的设定](http://www.wowotech.net/238.html)

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](http://loda.hala01.com/2014/12/armv8-%E8%88%87-linux%E7%9A%84%E6%96%B0%E6%89%8B%E7%AD%86%E8%A8%98/)
* linux kernel内存管理代码
* [MediaTek On-Line](https://online.mediatek.com/) => 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](http://ocw.nthu.edu.tw/ocw/index.php?page=course&cid=76&)》之第22～26讲
* cache操作范围：[PoU/PoC](http://community.arm.com/thread/4887)
* 硬件自动同步范围：[Shareability Domain](http://community.arm.com/thread/5400)

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](http://infocenter.arm.com/help/index.jsp) => 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](http://infocenter.arm.com/help/index.jsp) => 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](https://online.mediatek.com/) => 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)](https://msdn.microsoft.com/en-us/library/ms933779.aspx)
* [MediaTek On-Line](https://online.mediatek.com/) => 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](https://refspecs.linuxbase.org/elf/elf.pdf)
* [ARM文档中心](http://infocenter.arm.com/help/index.jsp) => 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](http://infocenter.arm.com/help/index.jsp) => 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](https://online.mediatek.com/) => 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](https://sourceware.org/gdb/current/onlinedocs/gdb/)
* [GNU Binary Utilities](https://sourceware.org/binutils/docs-2.25/binutils/index.html)（包含readelf、nm、objdump等）
* [DCC](https://dcc.mediatek.com/) => Debugger\_User Guide\_v5.1.docx => 4.4 GDB
* [MediaTek On-Line](https://online.mediatek.com/) => Trace32使用教程
* [Trace32官网](http://www.lauterbach.com/) => Support => E-Learning => Introduction to TRACE32 GUI

课外读物

* [Trace32官网](http://www.lauterbach.com/) => 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](https://online.mediatek.com/) => [Quick Start](https://online.mediatek.com/_layouts/15/mol/topic/ext/TopicHome.aspx) => Deep in MTK Turnkey Solution Logging Tools

课外读物

* [MediaTek On-Line](https://online.mediatek.com/) => 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](https://online.mediatek.com/) => 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](https://online.mediatek.com/) => 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](https://online.mediatek.com/) => 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](https://online.mediatek.com/) => 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](http://loda.hala01.com/2012/10/androidlinux-kernel-%E8%A8%98%E6%86%B6%E9%AB%94%E7%AE%A1%E7%90%86-%E5%85%A5%E9%96%80%E7%AD%86%E8%A8%98/)

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](https://online.mediatek.com/) => 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](http://man7.org/linux/man-pages/man2/ptrace.2.html)
* [wiki-ptrace](https://en.wikipedia.org/wiki/Ptrace)
* [MediaTek On-Line](https://online.mediatek.com/) => 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](http://man7.org/linux/man-pages/man5/core.5.html)
* [MediaTek On-Line](https://online.mediatek.com/) => 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](https://online.mediatek.com/) => 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](https://online.mediatek.com/) => 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](https://dcc.mediatek.com/) => 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、system property，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](https://computing.llnl.gov/tutorials/pthreads/)
* 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](https://online.mediatek.com/) => 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](http://infocenter.arm.com/help/index.jsp) => 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](http://www.dwarfstd.org/doc/DWARF4.pdf)
* [ARM文档中心](http://infocenter.arm.com/help/index.jsp) => 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](http://refspecs.linuxbase.org/LSB_3.1.0/LSB-Core-generic/LSB-Core-generic/ehframechpt.html)

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