



Melis RTOS 系统调试 使用说明

版本号: 1.1
发布日期: 2020.08.22

版本历史

版本号	日期	制/修订人	内容描述
1.0	2020.05.19	AWA1379	初始化添加 Melis RTOS 的系统常用调试方法
1.1	2020.08.22	AWA1379	添加 F133 芯片的软件调试介绍



目 录

1 概述	1
1.1 编写目的	1
1.2 使用范围	1
1.3 相关人员	1
2 LOG 输出	2
2.1 用途	2
2.2 LOG 输出方法	2
2.2.1 printf	2
2.2.2 printk	2
2.2.3 log/err/wrn/inf/msg	2
2.3 LOG 等级	3
2.3.1 静态确定输出等级	3
2.3.2 运行时指定输出等级	3
2.3.2.1 获取当前等级	4
2.3.2.2 设定等级	4
3 栈回溯	5
3.1 用途	5
3.2 配置	5
3.3 接口介绍	5
3.4 终端命令	5
3.5 回溯信息解析	6
3.6 注意事项	6
4 addr2line 分析	7
4.1 用途	7
4.2 用法	7
4.3 分析	8
5 slab debug	9
5.1 用途	9
5.2 配置	9
5.3 slab debug 错误信息分析	9
6 KASAN	11
6.1 用途	11
6.2 配置	11
6.3 KASAN 错误结果分析	11
6.4 注意事项	12
7 系统崩溃异常分析	13
7.1 ARM 架构芯片异常分析方法	13

7.1.1 崩溃 log 分析	13
7.2 RISC-V 架构芯片异常分析方法	15
7.2.1 RISC-V 异常分析	16
7.2.2 崩溃 log 分析	16
8 内存泄露分析	20
8.1 用途	20
8.2 接口介绍	20
8.3 终端命令	20
8.4 内存泄露 log 分析	20
9 系统状态分析	22
9.1 用途	22
9.2 终端命令	22
9.3 系统状态 log 分析	22
10 kgdb 调试	24
10.1 用途	24
10.2 配置	24
10.3 使用方法	24
11 断点调试	25
11.1 用途	25
11.2 配置	25
11.3 终端命令	25
11.4 断点异常分析	25
12 观察点调试	26
12.1 用途	26
12.2 配置	26
12.3 终端命令	26
12.4 观察点异常分析	27

1 概述

1.1 编写目的

此文档介绍从 Melis 4.0 开始，Melis RTOS 系统支持的常用软件调试方法，方便相关的开发人员快速高效地进行软件调试，提高解决软件问题的效率。

1.2 使用范围

Allwinner 软件平台 Melis 4.0 以上版本。

1.3 相关人员

使用 Melis 4.0 以上系统的广大客户以及软件开发人员。

2 LOG 输出

2.1 用途

输出程序运行日志、信息。

2.2 LOG 输出方法

2.2.1 printf

输出程序运行日志，可在任务中使用。

```
int printf(const char *fmt, ...);  
# 举例：printf("%s\n", __func__);
```

2.2.2 printk

输出程序运行日志，可在任务、中断上下文中使用。printk 可指定输出等级，如不指定，则默认为 1。

```
#include <log.h>  
  
int printk(const char *fmt, ...);  
  
# 举例：printk("%s\n", __func__); 默认等级为1。如运行时等级设为0，则会关闭printk接口输出。  
# 举例：printk(KERN_INFO"%s\n", __func__); 设定等级为4。如运行时等级数值高于4，则会关闭该句输出。
```

2.2.3 log/err/wrn/inf/msg

输出程序运行日志，可在任务、中断上下文中使用。

```
#include <log.h>  
  
__log("%s\n", __func__);  
__err("%s\n", __func__);  
__wrn("%s\n", __func__);
```

```
__inf("%S\n", __func__);  
__msg("%S\n", __func__);
```

2.3 LOG 等级

LOG 等级是指通过静态编译确定或者运行时指定 LOG 是否输出。在 Melis RTOS 系统中，printf 接口输出无等级设置，printk 和 __log 等接口可以通过指定 LOG 等级来设置是否输出。

表 2-1: LOG 等级表

接口	等级
__log	1
__err	2
__wrn	3
__inf	4
__msg	5

2.3.1 静态确定输出等级

配置项如下：

```
make menuconfig  
Environment Setup --->  
  (4) Default log level # 静态LOG等级。数值高于该等级的log输出不会被编译，用于减小固件大小。如  
    若设成0，则__log等接口无任何输出。
```

2.3.2 运行时指定输出等级

配置项如下：

```
make menuconfig  
Environment Setup --->  
  [*] dynamic log level support # 使能运行时指定输出等级。  
    Log Level Save (NONE) --->  
      (X) NONE # 使用全局变量保存运行时LOG输出等级，重启后失效。  
      ( ) RTC # 使用RTC数据寄存器保存运行时LOG输出等级，完全断电后失效。  
  (5) dynamic log level # 默认运行时LOG输出等级。如果设置成0，则printk，__log等接口无LOG输出。
```

2.3.2.1 获取当前等级

函数调用：

```
int get_log_level(void);
```

返回值：获取当前输出等级

同时还提供终端命令，

作用：获取当前LOG输出等级

用法：get_log_level

2.3.2.2 设定等级

函数调用：

```
void set_log_level(int level);
```

level：待设定的输出等级

同时还提供终端命令，

作用：设置LOG输出等级

用法：set_log_level level

3 栈回溯

栈回溯是指获取程序的调用链信息，通过栈回溯信息，能帮助开发者快速理清程序执行流程，提高分析问题的效率。

3.1 用途

获取程序调用关系，理清程序执行流程。

3.2 配置

配置项如下

```
make menuconfig
  Kernel Setup ---->
    Subsystem support ---->
      [*] Enable Backtrace Support
```

3.3 接口介绍

```
int backtrace(char *taskname, void *output[], int size, int offset, print_function
              print_func);
```

参数

taskname : 任务名字；可为NULL，表示回溯当前任务
output : 栈回溯结果保存数组，可以为NULL
size : output数组大小，可为0
offset : 栈回溯保存结果的偏移，可为0
print_func : 打印函数，在任务上下文可用printf，中断上下文可用printk

返回值

level : 回溯层次

3.4 终端命令

终端支持使用 backtrace 命令对指定的任务进行回溯。

作用：查看指定任务堆栈回溯信息
用法：backtrace [taskname]

3.5 回溯信息解析

1. 将回溯信息从终端中拷贝出来，在 Melis SDK 的 source 目录下创建 backtrace.txt 文件，并保存到 backtrace.txt 文件中。
2. 执行 `callstack backtrace.txt` 获取以下回溯信息。

```
mhd_start_scan at /xxx/mhd_apps_scan.c:334 #mhd_start_scan表示函数名，/xxx/mhd_apps_scan.c表示函数所在的文件路径，334表示函数调用处的行号。

mhd_softap_start at /xxx/mhd_apps_softap.c:263

wifi_recv_cb at /xxx/mhd_api_test.c:624

mhd_get_host_sleep at /xxx/mhd_apps_wifi.c:81

bswap_16 at /xxx/aw-alsa-lib/bswap.h:39

(inlined by) convert_from_s16 at ???:?
linear_init at /xxx/pcm_rate_linear.c:378

resampler_basic_interpolate_single at /xxx/resample_speexdsp.c:395
  or xxxx/xxxx/xxxx/xx.c: 405
  or xxxx/xxxx/xxxx/xx.c: 406

__fill_vb2_buffer at /xxx/videobuf2-v4l2.c:392

cci_read at /xxx/cc_helper.c:728

ecdsa_signature_to_asn1 at /xxx/ecdsa.c:294

cmd_wifi_fwlog at /xxx/mhd_api_test.c:449

# 函数调用顺序为从下到上，即cmd_wifi_fwlog -> ecdsa_signature_to_asn1 -> cci_read ... ->
  mhd_start_scan
# 注意，如果出现下列情况，则说明SDK中存在多个同地址的elf文件，需要用户从下列选择中选出一个合适的地址。
resampler_basic_interpolate_single at /xxx/resample_speexdsp.c:395
  or xxxx/xxxx/xxxx/xx.c: 405
  or xxxx/xxxx/xxxx/xx.c: 406
```

3.6 注意事项

请确保执行解析命令时所指定的 `melis30.elf` 为系统固件所对应的 `melis30.elf` 文件，否则解析后的栈回溯信息无法确保是正确的结果。

4 addr2line 分析

发生异常时，如果栈回溯失败，可以通过 addr2line 工具，对打印出来的栈上数据进行分析，从而确定栈回溯信息。

4.1 用途

在栈回溯失败时，使用 addr2line 从栈上数据中分析栈回溯信息。

4.2 用法

发生异常时当前栈内容打印如下：

```
sp stack memory:
0xc29c2f18: 0xc2639028 0xc299ba68 0x00000000 0x00000000
0xc29c2f28: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f38: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f48: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f58: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f68: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f78: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f88: 0x00000000 0x00000000 0x00000000 0x00000000
0xc29c2f98: 0x00000000 0xc24f3680 0x00000001 0xc299ba68
0xc29c2fa8: 0xc299ba68 0x00000001 0xc299b628 0x00000542
0xc29c2fb8: 0xc299bb68 0xc2141388 0xc299ba68 0xc24f3680
0xc29c2fc8: 0xc299a628 0xc299ba68 0xc299bb6a 0xc2142214
0xc29c2fd8: 0xc2141e2c 0x00000000 0xc2141e2c 0xdeadbeef
0xc29c2fe8: 0xdeadbeef 0xdeadbeef 0xdeadbeef 0xdeadbeef
0xc29c2ff8: 0xdeadbeef 0xc20d88b4 0x00000000 0x0001b63d
```

对所有的内存数据使用下列命令进行分析。

```
$(SDK_ROOT)/toolchain/bin/arm-melis-eabi-addr2line -a address -e source/ekernel/melis30.elf
-f

# 如果是F133芯片，则需要使用 $(SDK_ROOT)/toolchain/riscv64-elf-gcc-thead_20200528/bin/riscv64-
unknown-elf-addr2line 工具
# SDK_ROOT 表示SDK根目录
# -f: 显示函数名
# -a: address为打印出来的地址
# -e: 程序文件
```

4.3 分析

对于无法解析的内存数据予以丢弃后，可得到以下有效的分析信息。

```
0xc2141388
msh_exec
/home1/zhijinzeng/workPlace/temp/melis-v3.0/source/ekernel/subsys/finsh_cli/msh.c:415

0xc2142214
finsh_thread_entry
/home1/zhijinzeng/workPlace/temp/melis-v3.0/source/ekernel/subsys/finsh_cli/shell_entry.c
:746

0xc20d88b4
rt_thread_exit
/home1/zhijinzeng/workPlace/temp/melis-v3.0/source/ekernel/core/rt-thread/thread.c:93

# 函数调用关系 rt_thread_exit -> finsh_thread_entry -> msh_exec
```



5 slab debug

slab debug 是轻量级的堆内存越界检测工具，使能 slab debug 时，会在每块堆内存前后添加 8 字节的 redzone，在释放该内存块时，检查对应的 redzone 是否被篡改。如果被篡改，则进行报错。

5.1 用途

可用于分析 Melis RTOS 系统常见的堆内存越界以及重复释放等行为。

5.2 配置

slab debug 配置如下

```
make menuconfig
  Kernel Setup ---->
    RTOS Kernel Setup ---->
      RT-Thread Kernel Setup ---->
        [*] slab debug # 使能slab debug
        (16) slab debug backtrace level (NEW) # 设置分配内存时的栈回溯层级
```

5.3 slab debug 错误信息分析

slab debug 检测到内存越界行为之后，打印的错误信息如下：

```
page chunk check corrupted!, thread(tshell), chunk = 0xc3272ff8, size = 8192
backtrace : 0xc20dd98c
backtrace : 0xc20ddd20
backtrace : 0xc21a3b94
backtrace : 0xc20ecc84
backtrace : 0xc2197178
backtrace : 0xc21999a4
backtrace : 0xc2199a68
backtrace : 0xc219b810
backtrace : 0xc20d9534

total corrupted zone chunks = 0, total corrupted page chunk = 1

# page chunk check corrupted! 表示该内存块从page pool中分配
# zone chunk check corrupted! 表示该内存块从zone中分配
```

```
# thread(tshell) 表示分配该越界内存块时的任务名  
# chunk =0xc3272ff8 表示该越界内存块的地址  
# size = 8192 表示该越界内存块的大小  
# backtrace 表示分配该越界内存块时的栈回溯信息，可根据[栈回溯]章节内容进行解析
```



6 KASAN

KASAN 是一个动态检测内存错误的工具，可以检测全局变量、栈、堆的越界访问等问题，功能比 slab debug 齐全并且支持实时检测。当前该功能仅在 V833 和 V831 芯片实现。

6.1 用途

可用于分析 Melis RTOS 系统堆内存越界、堆内存释放后使用、堆内存重复释放、局部变量越界、全局变量越界等内存问题。

6.2 配置

```
KASAN依赖slab debug, 使能KASAN之前, 必须打开slab debug
make menuconfig
  Kernel Setup --->
    RTOS Kernel Setup --->
      RT-Thread Kernel Setup --->
        [*] slab debug # 使能slab debug
        (16) slab debug backtrace level (NEW) # 设置分配内存时的栈回溯层级

# 使能KASAN
make menuconfig
  Kernel Setup --->
    Subsystem support --->
      [*] Enable Kasan Support # 使能KASAN
      (0x54000000) kasan shadow offset (NEW) # 设置KASAN SHADOW偏移地址, 不可更改
      [ ] kasan inline (NEW)
      [*] kasan repeat report error (NEW) # 设置KASAN支持重复检测
```

6.3 KASAN 错误结果分析

```
out-of-bounds on rt_page_alloc
=====
BUG: KASAN: slab-out-of-bounds in c21dbfec
Read of size 1 at addr c7924001 by task tshell/0

page chunk check corrupted!, thread(tshell), chunk = 0xc7920ff8, size = 8192
Allocator Backtrace:
backtrace : 0xc20e074c
backtrace : 0xc20e0b90
```

```
backtrace : 0xc21f1bf0
backtrace : 0xc20f57a0
backtrace : 0xc21dbfb4
backtrace : 0xc21dc8fc
backtrace : 0xc21e33dc
backtrace : 0xc21e3528
backtrace : 0xc21e5a58
backtrace : 0xc20daf68
```

Caller Backtrace:

```
backtrace : 0XC20E074C
backtrace : 0XC20E0B90
backtrace : 0XC21F1BF0
backtrace : 0XC2671EC4
backtrace : 0XC2672014
backtrace : 0XC266EE64
backtrace : 0XC21DBFE8
backtrace : 0XC21DC8FC
backtrace : 0XC21E33DC
backtrace : 0XC21E3528
backtrace : 0XC21E5A58
backtrace : 0XC20DAF68
backtrace : 0XC20EFA64
```

```
=====
# BUG: KASAN: slab-out-of-bounds : 表示为内存越界错误
# Read of size 1 at addr c7924001 : 表示0xc7924001地址处越界
# Allocator Backtrace : 分配内存的栈回溯信息,可根据[栈回溯]章节内容进行解析
# Caller Backtrace : 错误发生时的栈回溯信息,可根据[栈回溯]章节内容进行解析
```

6.4 注意事项

使能 KASAN 之后,任务的栈空间、系统固件将会增大,程序运行卡顿,并且 KASAN 模块会占用全系统约 1/7 内存,使用时需要做好 DRAM 和 FLASH 空间的评估。使用 KASAN 时,至少预留约 1/7 的 DRAM 空间,同时需要适当增大系统固件分区的空间。如需使用,请联系全志工程师予以协助。

7 系统崩溃异常分析

系统崩溃异常主要是指 CPU 因非法地址访问、指令译码错误等原因，进入了异常模式，表现形式为系统打印异常栈信息和寄存器信息。

7.1 ARM 架构芯片异常分析方法

在 ARM 架构中，该类问题的分析方法如下：

1. 确认异常类型。
2. 栈回溯分析。栈回溯是指在系统崩溃之后，会打印发生异常时的栈回溯信息，供开发者进行分析，可参考[栈回溯](#)章节进行分析
3. 查看 DFAR 或者 IFAR 寄存器。当系统发生 data abort 或者 prefetch abort 异常时，发生异常时待访问的地址会被保存到 DFAR 寄存器中；发生 undefined instruction abort 时，发生异常时待访问的地址会被保存到 IFAR 寄存器中。

7.1.1 崩溃 log 分析

Melis 系统根据 CPU 异常的原因，有以下几种打印。

- memory access abort(IF)。CPU 异常为 prefetch abort，本意为取指错误，常见于软件运行到非法地址后，从该非法地址取指的场景。
- memory access abort(MA)。CPU 异常为 data abort，本意为数据访问错误，常见于访问非法地址的场景。
- undefined instructions trap。CPU 异常为 undefined instruction abort，本意为未定义指令异常，常见于内存越界导致代码被修改的场景。

以下是对崩溃 log 的分析。与体系结构紧密相关的部分，仅针对 ARM 架构平台。

```
=====
                        memory access abort(MA)
=====
thread: tshell, entry: 0xc21e4fd8, stack_base: 0xd7c99000,stack_size: 0x00014000.
cpsr: 0x80030053.
```

```
gprs:
r00:0x10000000 r01:0xd7cace40 r02:0xc21e62e8 r03:0x00000000
r04:0x10000000 r05:0x1af959c4 r06:0xd7c7b159 r07:0xdeadbeef
r08:0xdeadbeef r09:0xdeadbeef r10:0xdeadbeef r11:0xd7cacddc
r12:0xffa92814 sp:0xd7cacdc8 lr:0xc21e62e8 pc:0xc21e62ec
```

```
cp15:
fst_fsid:0x00000000
abt_dfar:0x10000000
abt_ifsr:0x0000123f
abt_dfsr:0x00000805
abt_lr :0xc21e62ec
abt_sp :0xffff3fac
```

```
cause:
DFSR signs section translation fault.
pgdbase = 0xc2000000
[0x10000000] *pgd=0x400000000
```

```
backtrace : 0XC21E62EC
backtrace : 0XC21E33DC
backtrace : 0XC21E3528
backtrace : 0XC21E5A58
backtrace : 0XC20DAF68
backtrace : 0XC20EFA64
```

-----TSK Usage Report-----

	name	errno	entry	stat	prio	tcb	slice	stacksize	stkfree
lt	pc	si	so						
	tshell	0	0xc21e4fd8	running	20	0xc79090f8	10	81920	79348
06	0x00000000	0000	0000						
	tidle	0	0xc20e7cb4	running	31	0xc6d00320	32	4096	2820
22	0xc20e15dc	0000	0000						
	timer	0	0xc20f1ecc	suspend	8	0xc6d01f00	10	16384	15656
10	0xc20e15dc	0000	0000						

memory info:

```
Total 0x1a6db000
Used 0x1030fcc8
Max 0x1031adb8
```

-----memory information-----

```
0xd7cacde8: 0x00000001 0xd7c7b158 0xd7cacea0 0x00000001
0xd7cacdf8: 0xc21e62c4 0x00000001 0xd7cace0c 0x00000000
0xd7cace08: 0x00000000 0xc27915a0 0x00000001 0x6ef959d4
0xd7cace18: 0x00000004 0xd7cacea0 0x41b58ab3 0xc2790a40
0xd7cace28: 0xc21e3214 0xc6d01c43 0x00000004 0xc79091cb
0xd7cace38: 0xd7cacea0 0xd7caceb4 0xd7c7b158 0x00000000
0xd7cace48: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7cace58: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7cace68: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7cace78: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7cace88: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7cace98: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7cacea8: 0x00000000 0x00000000 0x00000000 0x00000000
0xd7caceb8: 0x00000000 0x00000000 0xd7cacee4 0xc20ef6f8
```

=====

```
# memory access abort(MA): 内存访问错误, CPU异常类型为data abort
# thread : 表示异常发生时所在任务的信息
# gprs : 通用寄存器的值
# cp15 : cp15协处理器的寄存器的值
# abt_dfar : DFAR寄存器的值
# abt_ifsr : IFSR寄存器的值
# abt_dfsr : DFSR寄存器的值
# cause : 异常直接原因分析
# backtrace : 异常发生时栈回溯信息
# dump stack memory : 异常发生时栈的数据内容
```

7.2 RISC-V 架构芯片异常分析方法

在 RISC-V 架构中，该类问题的分析方法如下：

1. 确认异常类型。异常类型可参考 [RISC-V 异常类型表]。
2. 栈回溯分析。栈回溯是指在系统崩溃之后，会打印发生异常时的栈回溯信息[®]，供开发者进行分析，可参考[栈回溯](#)章节进行分析
3. 查看 spec 寄存器。当系统发生异常时，会将异常指令的地址保存到 sepc 寄存器中。如果 sepc 明显是一个非法的指令地址，可查看 ra 寄存器来确定异常地址。
4. 反编译查看异常指令，确定异常的直接原因并进行分析。常用反编译方法 riscv64-unknown-elf-objdump -d xxx.elf。xxx.elf 需要根据 spec 寄存器的值，确认其所属模块，然后选定对应的 elf 文件。

7.2.1 RISC-V 异常分析

异常种类	异常原因	排查方法
EXC_INST_MISALIGNED	指令地址不对齐, 要求 2bytes 对齐	1. 通过 spec 或 ra 寄存器, 确定出错的地址。 2. 检查该地址所在的函数是否使用函数指针, 该函数指针是否正确。
EXC_INST_ACCESS	取指异常	1. 通过 spec 或 ra 寄存器, 确定出错的地址。 2. 检查该地址所在的函数是否使用函数指针, 该函数指针地址是否处于 PMP 访问保护中, 是否 S/U 模式下无访问权限。
EXC_INST_ILLEGAL	译码失败, 非法指令	1. 通过 spec 或 ra 寄存器, 确定出错的地址。 2. 查看该地址的内存值, 并与反编译得出的该地址的值比较是否一致, 如果不一致, 则说明该处指令被篡改, 进一步排查篡改原因。
EXC_BREAKPOINT	触发 breakpoint	1. 通过 spec 寄存器, 确定出错的地址。 2. 检查该处地址是否为 ebreak 指令, 以及结合 backtrace 分析程序进入 ebreak 的原因。
EXC_LOAD_MISALIGN	加载数据的地址不对齐	1. 通过 spec 寄存器, 确定出错的地址, 反编译分析该地址处的指令。 2. 一般情况下, 出现该类异常, 与浮点运算和 ld 指令相关, 上述指令需要数据地址为 8 bytes 对齐。比如 <code>value = *(unsigned long *)addr</code> , 当 <code>addr</code> 为非 8 bytes 对齐时会出现此类异常。
EXC_LOAD_ACCESS	加载数据访问异常	1. 通过 spec 寄存器, 确定出错的地址。 2. 分析该地址是否处于 PMP 访问保护中, S/U 模式下是否无访问权限。
EXC_STORE_MISALIGN	写回数据的地址不对齐	1. 通过 spec 寄存器, 确定出错的地址, 反编译分析该地址处的指令。 2. 一般情况下, 出现该类异常, 与浮点运算和 sd 指令相关, 上述指令需要数据地址为 8 bytes 对齐。比如 <code>*(unsigned long *)addr = 0</code> , 当 <code>addr</code> 为非 8 bytes 对齐时会经常出现此类异常。
EXC_STORE_ACCESS	回写数据访问异常	1. 通过 spec 寄存器, 确定出错的地址, 分析该地址是否处于 PMP 访问保护中, S/U 模式下是否无访问权限。
EXC_INST_PAGE_FAULT	取指访问页面异常	1. 通过 spec 或 ra 寄存器, 确定出错的地址。 2. 检查该地址所在的函数是否使用函数指针, 该函数指针地址是否尚未被 MMU 映射。
EXC_LOAD_PAGE_FAULT	加载访问页面异常	1. 通过 spec, 确定出错的地址, 反编译分析该地址处的指令, 得到加载数据的源地址。 2. 分析该地址是否尚未被 MMU 映射; 若未映射, 需要结合 backtrace 分析该源地址的来源。
EXC_STORE_PAGE_FAULT	回写数据访问页面异常	1. 通过 spec, 确定出错的地址, 反编译分析该地址处的指令, 得到回写数据的目的地址。 2. 分析该地址是否尚未被 MMU 映射; 若未映射, 需要结合 backtrace 分析该源地址的来源。
EXC_SYSCALL_FRM_U	从 U 模式发起 ecall 系统调用	1. 通过 spec, 确定出错的地址, 以及结合 backtrace 分析程序进入 ecall 的原因。
EXC_SYSCALL_FRM_M	从 M 模式发起 ecall 系统调用	1. 通过 spec, 确定出错的地址, 以及结合 backtrace 分析程序进入 ecall 的原因。

图 7-1: RISC-V 异常处理分析

7.2.2 崩溃 log 分析

以下是对崩溃 log 的分析。与体系结构紧密相关的部分, 仅针对 RISC-V 架构平台。

```
=====
EXC_STORE_PAGE_FAULT
=====
```

thread: tshell, entry: 0x00000000400e5ad0, stack_base: 0x0000000040515000, stack_size: 0x00004000.

gprs:

x0:0x0000000000000000	ra:0x00000000400e4830	sp:0x0000000040518e60	gp:0
x0000000040260c70			
tp:0x00000000402b5980	t0:0x0000000040012494	t1:0x000000000000000f	t2:0
x0000000000000100			
s0:0x0000000040518e80	s1:0x00000000400e5ad0	a0:0x0000000000000001	a1:0
x0000000040518ea8			
a2:0x0000000040518ea8	a3:0x00000000400e67b4	a4:0x0000000040518ea8	a5:0
x00000000f0000000			
a6:0x0000000000000000	a7:0x000000000000000a	s2:0x0000000000000000	s3:0
x000000004002ff78			
s4:0x00000000deadbeef	s5:0x00000000deadbeef	s6:0x00000000deadbeef	s7:0
x00000000deadbeef			
s8:0x00000000deadbeef	s9:0x00000000deadbeef	s10:0x00000000deadbeef	s11:0
x00000000deadbeef			
t3:0x0000000000000001	t4:0x0000000000000000	t5:0x0000000000000000	t6:0
x0000000000000000			

other:

sepc :0x00000000400e67c8
sstatus :0x0000000000004120
sscratch:0x0000000000000000

-----backtrace-----

backtrace : 0X400E67C8
backtrace : 0X400E482E
backtrace : 0X400E48A8
backtrace : 0X400E60B0
backtrace : 0X40026498
backtrace : 0X400263E8

-----TSK Usage Report

name	errno	entry	stat	prio	tcb	slice	stacksize	stkfree
lt	si	so	stack_range					
system_msg	0	0xe170b208	suspend	24	0x402b7738	15	2048	760
15	0000	0000	[0x404c6000-0x404c6800]					
headbar_fresh	0	0xe1706994	suspend	24	0x402b74b8	15	8192	5880
15	0000	0000	[0x407a0000-0x407a2000]					
KsrvMsg	0	0xe1f157f0	suspend	24	0x402b6838	15	4096	2408
15	0000	0000	[0x40627000-0x40628000]					
init_process	0	0xe17039d8	suspend	24	0x402b65b8	15	65536	54952
02	0000	0000	[0x4062d000-0x4063d000]					
MsgSrv	0	0xe1f16314	suspend	24	0x402b6338	15	4096	1836
13	0000	0000	[0x40626000-0x40627000]					
OrangTmr	0	0xe18299f0	suspend	9	0x402b60b8	20	4096	2632
20	0000	0000	[0x405b5000-0x405b6000]					
tshell	0	0x400e5aa8	running	20	0x402b5938	10	16384	10492
10	0000	0000	[0x40515000-0x40519000]					
tp_input	0	0x400f9d44	suspend	3	0x402b56b8	10	2048	1440
10	0000	0000	[0x404c4800-0x404c5000]					
mose_input	0	0x400f9504	suspend	3	0x402b5438	10	2048	1440
10	0000	0000	[0x404c4000-0x404c4800]					
kb_input	0	0x400f8b60	suspend	3	0x402b51b8	10	8192	7400
10	0000	0000	[0x40513000-0x40515000]					

disp2	0	0x40031bd8	suspend	15	0x402b4f38	10	8192	5192
02 0000 0000		[0x403fe000-0x40400000]						
kservice	0	0x40101f3c	suspend	6	0x402b4538	10	4096	2424
08 0000 0000		[0x40360000-0x40361000]						
fs	0	0x4015d794	suspend	7	0x402b42b8	10	16384	12456
06 0000 0000		[0x40310000-0x40314000]						
app_init0	0	0x40026060	suspend	0	0x402b4038	10	16384	11464
01 0000 0000		[0x402bc000-0x402c0000]						
tidle	0	0x4002bae0	running	31	0x4026a030	32	8192	6668
18 0000 0000		[0x4026a288-0x4026c288]						
timer	0	0x40031740	suspend	8	0x4026c958	10	16384	11868
04 0000 0000		[0x4026cbb0-0x40270bb0]						

memory info:

Total 0x03d70000
Used 0x00be87c8
Max 0x00fd1478

-----memory information

dump stack memory:

0x0000000040518e60: 0x40518ea8 0x00000000 0x404106a1 0x00000001
0x0000000040518e70: 0x40518f20 0x00000000 0x40518f20 0x00000000
0x0000000040518e80: 0x402b5938 0x00000000 0x40518f34 0x00000000
0x0000000040518e90: 0x00000001 0x00000000 0x404106a0 0x00000000
0x0000000040518ea0: 0xdeadbeef 0x00000000 0x404106a0 0x00000000
0x0000000040518eb0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000040518ec0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000040518ed0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000040518ee0: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000040518ef0: 0x00000000 0x00000000 0x40193e78 0x00000001
0x0000000040518f00: 0x400e67b4 0x00000000 0x00000001 0x00000000
0x0000000040518f10: 0x40518f50 0x00000000 0x400e48ac 0x00000000
0x0000000040518f20: 0x00000001 0x00000000 0x404106a0 0x00000000
0x0000000040518f30: 0x00000001 0x00000000 0x00000001 0x00000000
0x0000000040518f40: 0x40518f90 0x00000000 0x400e60b4 0x00000000
0x0000000040518f50: 0x40518f90 0x00000000 0x00000000 0x00000000
0x0000000040518f60: 0x0000000d 0x00000000 0x404104b8 0x00000000
0x0000000040518f70: 0x40518f90 0x00000000 0x40025bb8 0x00000000
0x0000000040518f80: 0x40519000 0x00000000 0x4002649a 0x00000000
0x0000000040518f90: 0x402b5980 0x00000000 0x402b4f80 0x00000000
0x0000000040518fa0: 0xffffffff 0xffffffff 0x00000000 0x00000000
0x0000000040518fb0: 0x4002ff78 0x00000000 0x400e5ad0 0x00000000
0x0000000040518fc0: 0x402b5938 0x00000000 0x402b4f38 0x00000000
0x0000000040518fd0: 0xdeadbeef 0x00000000 0x4002ff78 0x00000000
0x0000000040518fe0: 0x00000000 0x00000000 0x400e5ad0 0x00000000
0x0000000040518ff0: 0xffffffff 0xffffffff 0x400263e8 0x00000000
0x0000000040519000: 0x10141cef 0x70000000 0x101420ef 0x70000000
0x0000000040519010: 0x101424ef 0x70000000 0x101428ef 0x70000000
0x0000000040519020: 0x10146cef 0x70000000 0x101470ef 0x70000000
0x0000000040519030: 0x101474ef 0x70000000 0x101478ef 0x70000000
0x0000000040519040: 0x00000000 0x00000000 0x00000000 0x00000000
0x0000000040519050: 0x00000000 0x00000000 0x00000000 0x00000000

dump sepc memory:

0x00000000400e67a0: 0x0007b023 0x853e4781 0x61056462 0x02908082
0x00000000400e67b0: 0xec061101 0x1000e822 0xfea43423 0xfeb43023

```
0x00000000400e67c0: 0xfe843703 0xfe043783 0x00e7f763 0xfe043783
0x00000000400e67d0: 0x4781c399 0x3783a82d 0x853efe84 0xc70410ef
0x00000000400e67e0: 0x873e87aa 0x04634785 0x478100f7 0x3783a00d
0x00000000400e67f0: 0xcf89fe04 0xfe043783 0x10ef853e 0x87aac524
0x00000000400e6800: 0x4785873e 0x00f70463 0xa0114781 0x853e4785
0x00000000400e6810: 0x644260e2 0x80826105 0x02900000 0xfc067139
0x00000000400e6820: 0x0080f822 0x302387aa 0x2623fcb4 0x3823fcf4
0x00000000400e6830: 0x2783fc04 0x871bfcc4 0x47850007 0x00e7ca63
0x00000000400e6840: 0x00093517 0x61050513 0xe4c440ef 0xa8f14781
0x00000000400e6850: 0xfcc42783 0x0007871b 0xd5634789 0x378308e7
0x00000000400e6860: 0x07a1fc04 0x0713639c 0x4601fd04 0x853e85ba
0x00000000400e6870: 0x9b42d0ef 0xfea43423 0xfc043783 0x639c07c1
0x00000000400e6880: 0xfd040713 0x85ba4601 0xd0ef853e 0x87aa99a2
0x00000000400e6890: 0xfef42223 0xfe446783 0xfe843703 0x3c2397ba
0x00000000400e68a0: 0x3783fcf4 0x3703fe84 0x85bafd84 0xf0ef853e
0x00000000400e68b0: 0x87aaf03f 0x3783c38d 0xcf91fd84 0xfe843703
0x00000000400e68c0: 0xfe442783 0x0027d79b 0x27812781 0x853a85be
0x00000000400e68d0: 0x889440ef 0x3517a891 0x05130009 0x40ef5925
0x00000000400e68e0: 0xa099db64 0xfc043783 0x639c07a1 0xfd040713
0x00000000400e68f0: 0x85ba4601 0xd0ef853e 0x342392e2 0x3783fea4
0x00000000400e6900: 0x4581fe84 0xf0ef853e 0x87aaeabf 0x3783cb81
0x00000000400e6910: 0x4589fe84 0x40ef853e 0xa0398434 0x00093517
0x00000000400e6920: 0x54c50513 0xd70440ef 0x853e4781 0x744270e2
0x00000000400e6930: 0x80826121 0x02900000 0xfc067139 0x0080f822
0x00000000400e6940: 0x302387aa 0x2623fcb4 0x3c23fcf4 0x2783fc04
0x00000000400e6950: 0x871bfcc4 0x47890007 0x00e7ca63 0x00093517
0x00000000400e6960: 0x52450513 0xd30440ef 0xa08d57fd 0xfc043783
0x00000000400e6970: 0x639c07a1 0xfd840713 0x85ba4601 0xd0ef853e
0x00000000400e6980: 0x34238a62 0x3783fea4 0x07c1fc04 0x0713639c
0x00000000400e6990: 0x4601fd84 0x853e85ba 0x88c2d0ef 0xfea43023
```

EXC_STORE_PAGE_FAULT: 回写数据访问页面异常, 可参考[RISC-V异常分析]来分析

thread : 表示异常发生时所在任务的信息

gprs : 通用寄存器的值

sepc : 异常发生时pc寄存器的值

sstatus : 异常发生时sstatus寄存器的值

sscratch : 异常发生时sstatus寄存器的值

backtrace : 异常发生时栈回溯信息

dump stack memory : 异常发生时栈的数据内容

dump sepc memory : 异常发生时sepc地址指向的数据内容

8 内存泄露分析

Melis RTOS 系统提供轻量级的内存泄露分析功能，启动内存泄露分析后，每当申请内存时，将该内存块挂入链表中，释放时将其从链表中摘除。最终还存在于链表之中的，便是可疑的内存泄露点。

8.1 用途

可用于分析、定位 Melis RTOS 系统的内存泄露问题。

8.2 接口介绍

```
void esKRNLMemLeakChk(uint32_t en);
```

参数

en : 0, 关闭内存泄露检测; 1, 打开内存泄露检测

注意：关闭内存泄露检测时，会打印可疑的内存泄露点，并清空链表

8.3 终端命令

终端提供 mmlk 命令来进行内存泄露检测。

作用：查看可疑内存泄露信息

用法：第一次执行mmlk表示开启内存泄露检查，第二次执行mmlk表示关闭内存泄露信息检查

8.4 内存泄露 log 分析

关闭内存泄露检测时，会打印可疑的内存泄露点及其回溯信息，用户可根据回溯信息，参考[栈回溯](#)章节进行分析。

```
000: ptr = 0xd2c3f000, size = 0x00001000, entry = 0xc2199d0c, thread = tshell, tick = 1190.  
      backtrace : 0xc20efd44  
      backtrace : 0xc2195388  
      backtrace : 0xc2195c78
```



```
backtrace : 0xc2198640  
backtrace : 0xc2198704  
backtrace : 0xc219a4ac  
backtrace : 0xc20d95b8
```

```
# ptr : 存留在链表中的内存块地址  
# size : 存留在链表中的内存块大小  
# entry : 分配该内存块时任务的入口  
# thread : 分配该内存块时任务的名称  
# backtrace : 申请该内存块时的栈回溯信息
```



9 系统状态分析

Melis RTOS 系统在终端提供 top 命令来查看系统状态信息。

9.1 用途

可用于分析 Melis RTOS 系统的堆栈越界、系统卡死等问题，也可用于分析系统性能瓶颈。

9.2 终端命令

作用：查看系统性能信息
用法：top

9.3 系统状态 log 分析

```
-----TSK Usage Report(tick:61283:s:613:n:291799341)-----
  name  errno  entry  cputime  stat  prio  tcb  slice stacksize
stkusg lt   pc    si  so  born  fpu
perf    0    0xc20f1064 000.05% running  1  0xc28b41e8  30    8192
21.83% 24 0xc20f1104 0001 0001 60877  1
tshell  0    0xc2199d0c 000.00% suspend 20  0xc28b4108  10   40960
02.67% 09 0xc20de6c0 0000 0000 0069  1
tidle   0    0xc20e24e8 099.95% running 31  0xc2761010  32    2048
47.27% 07 0xc20de6c0 0001 0001 0000  1
timer   0    0xc20ealb0 000.00% suspend  8  0xc2761e54  10   16384
02.86% 10 0xc20de6c0 0000 0000 0000  1
-----
memory info:
  Total  0x1ec88000
  Used   0x100bf0c8
  Max    0x100c00c8
```

```
# name      : 任务名
# entry     : 任务入口地址
# cputime    : cpu占用率
# stat      : 任务状态
# prio      : 任务优先级
# stacksize  : 任务栈大小
# stkusg    : 任务栈使用率
# si        : 任务得到调度机会的次数
# so        : 任务被调度出去的次数
```

```
# memory info 内存信息
# Total : 系统堆内存大小
# Used  : 当前使用的系统堆内存大小
# Max   : 当前使用的系统堆内存峰值
```



10 kgdb 调试

Kgdb 是一个 gdb 调试桩，需要结合开发主机端的 gdb 一起使用，通过 kgdb，可以对 Melis 系统进行单步调试，设置断点，观察变量，修改寄存器的值，查看任务栈回溯等。kgdb 通过串口线与目标板进行连接调试。该功能仅在 V833 和 V831 芯片上实现。

10.1 用途

可用于单步调试，理清代码运行流程，提升软件调试的效率等。

10.2 配置

```
make menuconfig
  Kernel Setup ---->
    Subsystem support ---->guanbi
      Allwinner Components Support ---->
        [*] kgdb_sutb
```

10.3 使用方法

1. 在串口终端执行kgdb，并退出串口终端。切记，此处一定要退出串口终端，否则主机端gdb无法与设备端kgdb进行连接。
2. 执行命令，/home/xxx/workPlace/melis-v3.0/toolchain/bin/arm-melis-eabi-gdb ekernel/melis30.elf，使用绝对路径来运行gdb，并加载符号表。
3. 在gdb界面执行set serial baud 115200，设置串口波特率。
4. 在gdb界面执行target remote /dev/ttyUSB0，此处ttyUSB0需要根据串口号进行设置。
5. 如此便可连接成功，可使用gdb进行断点调试、修改内存、寄存器的值，查看堆栈等。

11 断点调试

断点调试是指利用 CPU 的硬件断点或者软件断点来进行调试，通过对指定的地址设置断点，当程序执行到该地址时，触发 prefetch abort 异常，再根据异常信息进行分析。通过使用该方法，可以迅速判断程序是否执行到指定的地址。该功能仅在 V833 和 V831 芯片上实现。

11.1 用途

可用于分析软件执行流程，以及快速分析函数调用参数、返回值等。

11.2 配置

与 kgdb 相同，依赖 kgdb。

11.3 终端命令

作用：设置程序断点，当前仅使用硬件断点
用法：breakpoint [set | remove] addr
set：设置断点
remove：取消断点
addr：在该地址设置断点

11.4 断点异常分析

断点异常分析，可参考[系统崩溃异常分析](#)章节进行分析。

12 观察点调试

观察点调试是指利用 CPU 的硬件观察点来进行调试，通过对指定的地址设置指定属性的观察点，当 CPU 对该地址进行指定属性的操作时，会触发 data abort 异常，然后再根据异常信息进行分析。通过使用该方法，可以迅速判断某块内存是否被修改、读取或者访问。

表 12-1: 观察点属性表

属性	作用
write	监视写操作
read	监视读操作
access	监视访问操作，包括读和写

12.1 用途

可用于分析某块内存处是否被篡改等问题。

12.2 配置

与 kgdb 相同，依赖 kgdb。

12.3 终端命令

作用：设置硬件观察点，当前仅使用硬件断点
用法：watchpoint [write | read | access | remove] addr
write：监视写操作
read：监视读操作
access：监视访问操作
remove：取消观察点
addr：在该地址设置/取消观察点

12.4 观察点异常分析

观察点异常分析，可参考[系统崩溃异常分析](#)章节进行分析。



著作权声明

版权所有 © 2021 珠海全志科技股份有限公司。保留一切权利。

本文档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本文档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本文档内容的部分或全部，且不得以任何形式传播。

商标声明

、 全志科技、（不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本文档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本文档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本文档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本文档作为使用指导仅供参考。由于产品版本升级或其他原因，本文档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本文档中提供准确的信息，但并不确保内容完全没有错误，因使用本文档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本文档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本文档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。