

广东中兴新支点

Kmemchk 用户手册

嵌入式操作平台

目 录

1. 功能概述.....	1
1.1. 实现原理.....	1
1.2. 检测范围.....	2
2. 快速入门.....	3
2.1. 内存泄露检测.....	3
2.2. 内存实时越界检测.....	7
3. 获取工具.....	11
3.1. 编译工具链的获取.....	11
3.2. kmemchk 客户端工具和配置文件的获取.....	11
4. 编译程序.....	11
4.1. 内存泄露检测.....	12
4.2. 内存实时越界检测.....	12
4.3. 构建注意事项.....	14
5. 配置说明.....	15
5.1. 内存泄露检测配置项.....	15
5.1.1. 常用配置项.....	15
5.1.2. 其它配置项.....	16
5.1.3. 配置方式.....	18
5.1.4. 配置文件实例.....	19
5.2. 内存实时越界检测配置项.....	24

5.2.1.	配置日志文件的生成路径	24
5.2.2.	配置写文件的信号	24
5.2.3.	功能配置	24
6.	运行使用.....	26
6.1.	内存泄露检测.....	26
6.1.1.	运行程序	26
6.1.2.	获取日志	27
6.2.	内存越界检测.....	28
6.2.1.	运行程序	28
6.2.2.	统计信息实时查看	29
6.2.3.	获取日志文件	29
7.	KMCHKTOOL 工具	30
7.1.	使用方法.....	30
7.1.1.	运行工具程序	30
7.1.2.	指定应用程序符号表	30
7.1.3.	指定库文件路径	31
7.1.4.	指定分析目标日志文件	31
7.2.	命令介绍.....	31
7.2.1.	kmchk_leak 命令.....	31
7.2.2.	kmchk_slop 命令.....	54
8.	检测信息详解	58
8.1.	kmchk_leak 信息详解.....	58
8.1.1.	打印输出的信息	58
8.1.2.	内存泄露检测统计/跟踪日志信息	61
8.2.	kmchk_slop 信息详解	64
8.2.1.	打印输出的信息	65
8.2.2.	工具解析信息	65
9.	CGSL 环境下使用 KMEMCHK	68

1. 功能概述

内存检测工具（kmemchk）是一款集合用户态内存泄露检测和内存越界实时检测两大功能的检测工具。它可以在不修改源代码的前提下，检测应用程序的内存泄露情况和内存越界操作。检测范围包括 C/C++ 语言，具有对应用程序影响较小，检测准确、可快速定位故障的特点。

1.1. 实现原理

■ 内存泄露检测

Kmemchk 内存泄漏检测工具通过在标准内存操作库函数中挂接钩子函数来接管应用程序的内存申请和释放操作，从而实现跟踪和记录程序的内存使用情况。gcc 编译器提供了构造函数（constructors）和析构函数（destructors）两大属性，使用构造属性声明的函数可在程序 main 函数之前执行，使用析构属性声明的函数可在程序 main 函数退出之后执行，内存泄漏检测工具分别定义了这两类属性的函数来在程序初始化阶段在标准内存操作库函数中挂接钩子函数和在退出阶段还原钩子函数。

可使用如下方式指定构造函数和析构函数：

```
static void __attribute__((constructor)) kmchk_init(void)
static void __attribute__((destructor)) kmchk_fini(void)
```

■ 内存越界实时检测

在应用程序编译和链接阶段，内存实时越界检测工具自动在应用程序进行内存操作的时候插入监控函数，在程序运行起来后，记录应用程序的内存分配释放情况，并在程序进行内存访问之前，将本次访问的内存地址和范围，与已记录内容比对，判断本次操作是否越界，如果越界则将相关信息实时输出到日志文件中。

越界检测输出的信息包括越界操作的对象名称、内存地址、范围、函数调用链、源文件行定位等，对于溢出类型的越界，还会同时输出该段内存分配时的内存信息（对象名称、内存地址、范围、函数调用链、源文件行定位等），十分有利于故障定位，解决了越界发生点和故障现象关联不上的内存错误难题。

1.2. 检测范围

■ 内存泄露检测

内存泄露检测功能已经提供了广泛的 CPU 架构支持，目前已支持的架构包括：

- X86-32
- X86-64
- MIPS-32
- MIPS-64
- PPC
- ARM
- ARM-NOMMU

内存泄露检测功能能够检测的应用程序调用函数包括：

- malloc
- realloc
- memalign
- mmap
- free
- munmap

■ 内存越界实时检测

kmemchk 的内存实时越界检测功能能够探测到各种超出整个变量地址空间的越界操作，主要包括以下几个方面：

- 1) 全局、局部、静态、动态数组和多维数组的越界读写操作；
- 2) 字符串、内存拷贝溢出；
- 3) 未初始化的内存读操作；
- 4) 读写空指针（NULL）操作；
- 5) 动态内存 free 后再进行读写操作；
- 6) 小结构体指针强制转化为大结构体指针（包括结构体动态分配和静态分配），越界访问操作；

- 7) 通过操作形参触发（包括指针、变量）的在子函数中发生的内存越界操作；
- 8) 通过操作形参触发（包括指针、变量）的在父函数中发生的内存越界操作；
- 9) 线程栈越界检测。

目前，暂时不能提供越界检测的操作，主要有：

- 1) 结构体成员变量越界，但越界范围没有超过结构体的内存范围；
- 2) 数组某一单元越界，但越界范围没有超过数组的内存范围；
- 3) 大结构变量转化成小结构变量进行操作，小结构变量越界但未超出大结构范围。例如 `int a; char *p = &a; memset (p, 0, 2);`
- 4) 一段内存的部分转化为结构体，结构体越界，但未超出内存范围。例如将一段 IP 报文头指针转化为 IP 头结构体指针，操作 IP 头结构体指针偏移出结构体范围进行读写。

2. 快速入门

内存检测工具的使用总体上分为三大步骤：程序配置编译、运行程序获取日志和检测信息解析。

下面分别介绍内存泄露检测和内存实时越界检测的主要操作顺序和基本步骤。

2.1. 内存泄露检测

- 1) 编译程序，添加链接参数“-kmchk_leak”即可；

以 test_leak.c 代码为例，源码如下：

```
#include <stdio.h>
#include <malloc.h>
#include <pthread.h>

void *thread1();
int main(int argc, char *argv[])
{
    pthread_t t_1;
    pthread_create(&t_1, NULL, thread1, NULL);
    pthread_join(t_1, NULL);
}
```

```
while(1)
sleep(1);
return 0;
}
void *thread1()
{
char *p;
int i = 0;
for(i=0;i < 10;i++)
{
p=malloc(10*(i+1));
}
return 0;
}
```

使用如下命令，对其进行静态编译：

```
/opt/kmemchk/x86_gcc4.1.2_glibc2.5.0/bin/i686-pc-linux-gnu-gcc -g test_leak.c
-o test_leak -kmchk_leak -static
```

- 2) 运行程序。将可执行程序 and 配置文件（若使用默认配置，则不需要配置文件）放置到运行环境下，且将配置文件放到与应用程序同一个目录下，然后运行程序；
- 3) 获取日志。如果在程序运行过程中想获取日志，则需要发送生成日志的信号。生成日志的信号默认为 SIGURG，但是当 SIGURG 被应用程序占用时，也可以在配置文件中（详见 5.1 节）指定其它信号来代替，发送信号的命令如下：

```
kill -信号 pid
```

例如

```
kill -SIGURG 768
```

另外，在程序正常退出时 kmemchk 也会生成日志，默认情况下日志与程序在同一目录，并按如下格式命名：

```
程序名_进程号_kmchk_leak.log
```

例如对于 test_leak，生成的日志文件为“test_leak_768_kmchk_leak.log”。该日志文件中保存了发送信号时，进程所占用的全部动态内存信息。

- 4) 解析日志。将 kmemchk 生成的日志文件拷贝到主机端 kmchktool 工具所在目录下，可按照如下步骤使用 kmchktool 解析日志：

- a) 运行 kmchktool;
- b) 执行 **set program** 命令指定应用程序符号表, 如 “**set program test_leak**”;
- c) 执行 **set libpath** 命令指定库文件路径 (如果是静态链接的程序则不需要指定);
- d) 执行 **set log** 命令指定待解析的由 kmemchk 导出的日志文件 (该命令必须执行), 如 “**set log test_leak_768_kmchk_leak.log**”;
- e) 使用各类 **show** 命令查看相应信息, 如下所示:

show prof 查看统计信息

```
(kmem) show prof
##### Process prof log #####

```

	calls count	failed count	total size(byte)
malloc	214	0	57682193
realloc	200	0	15150
free	21	0	115343365
memalign	110	0	57676730
mmap	5	0	33566848
munmap	2	0	1048576
unfree mem size	32548980		
unfree mem peak	43014540		

```
NOTES
The failed count item include the following conditions:
allocate memory failed,allocate 0 byte memory and free a NULL pointer.
```

图 2-1 show prof 命令查看统计信息

show prof 命令显示的信息中各字段的意义如下:

- 第 1 列, 显示函数: malloc、realloc、free、memalign、mmap、munmap
- 第 2 列, calls count, 每个函数的调用次数;
- 第 3 列, failed count, 内存操作返回失败 (包括申请或释放内存失败、申请 0 字节以及释放空指针的情况) 的次数;
- 第 4 列, 每个函数操作的动态内存总大小, total size, 单位 byte
- unfree mem size, 应用程序当前使用动态内存的值, 单位 byte
- unfree mem peak, 应用程序运行过程中使用的动态内存最大值, 单位 byte

show unfree 查看未释放内存信息

```
RecordNo : 2
type: malloc
count: 10
size: 550
call trace :
0 : 0x806e19f <__libc_malloc>
1 : 0x8048310 <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a <start_thread>
3 : 0x807276e <clone>
```

图 2-2 show unfree 命令按函数调用链归类显示的未释放内存信息

show unfree 命令默认情况下按函数调用链归类显示日志文件中 kmemchk 记录的未释放内存信息，并将信息以可读的文本形式打印输出到终端，如果带-l 参数则将信息写到-l 参数指定的文件中。显示内存信息中各字段的意义如下：

- RecordNo: 信息编号；
- type: 内存块是通过哪个标准内存申请库函数申请的，上例中为 malloc；
- count: 在同一个函数调用链下有多少未释放的内存块，上例中为 10 块；
- size: 在同一个函数调用链下未释放的内存块的总大小，上例中为 550 个字节；
- call trace: 内存块是在哪个函数调用链下申请的；

若要查看每块未释放内存的详细信息，可在 show 命令中增加-p 选项，即

```
show unfree -p
```

信息格式如下：

```
RecordNo : 2
tid : 11140
seq_num : 3
memory operation : malloc
size : 10
start_addr : 0x8f6b040
call trace :
0 : 0x806e19f <__libc_malloc>
1 : 0x8048310 <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a <start_thread>
3 : 0x807276e <clone>
```

图 2-2 show unfree -p 显示的一块未释放内存的详细信息

其中各字段的意义如下：

- RecordNo: 信息编号，该编号是解析工具为每条信息编的号，不是 kmemchk 按时间顺序对内存操作编的号，此处为 2，表示第 2 条信息；
- seq_num: 序列号，该序列号是 kmemchk 按时间顺序对内存操作编的号；

- tid: 线程号, 申请该内存块的线程 id;
- memory operation: 内存块是通过哪个标准内存申请库函数申请的, 上例中为 malloc;
- size: 该内存块的大小, 上例中为 10 个字节。注意该字段与不加-p 参数时的 size 字段意义不同;
- start_addr: 该内存块的起始地址;
- call trace: 内存块是在哪个函数调用链下申请的;

还有一个常用的查看命令 **show diff**, 该命令用于比较一个程序在运行过程中不同时刻分别发送信号获取的两份日志, 得到获取两份日志之间的时间段内申请的且未释放的内存信息, 例如,

```
show diff log1 log2 -l diff12.txt
```

其中 log1、log2 分别为一个程序在运行过程中不同时刻获取的日志, -l 指定保存比较结果的文件。比较得到的信息格式与 show unfree 一样, 区别在于其中只记录了两份日志获取的时间段内申请的且未释放的内存信息。该命令常用于分析那些具有累加性的内存泄露, 比如程序每执行一次某个功能就会泄露一些内存的情况。关于 kmchktool 工具支持命令的详细说明, 请参见 7.2 节。

2.2. 内存实时越界检测

- 1) 编译程序, 添加相应的编译参数-kmchk_slop, 开启检测功能;

以 test.c 代码为例, 源码如下:

```
#include <stdio.h>
int a[2];
void tt()
{
    int c = a[2];
}
int main(int argc, char *argv[])
{
    int b = a[3];
    tt();
    return 0;
}
```

使用如下命令, 对其进行静态编译:

```
/opt/kmemchk/x86_gcc4.1.2_glibc2.5.0/bin/i686-pc-linux-gnu-gcc test.c -o test
-kmchk_slop -static
```

- 2) 运行程序。将可执行程序放置到运行环境下（相应单板）运行。运行之前，可以通过环境变量对内存实时越界检测功能进行配置（详见 4.2），如无特殊需求，采取默认配置即可。

```
./test
```

- 3) 获取日志。程序启动后会在程序运行的当前目录下生成名为“程序名_进程号_kmchk_slop.log”的二进制格式的日志文件，此处生成了一个名为 test_10992_kmchk_slop.log 的日志文件；（注：由于 kmemchk slop 是实时输出检测结果，这点与 kmemchk leak 不同，所以一般可以直接获取日志文件，不需要发信号。详见 5.2.2）
- 4) 解析日志。将 kmemchk 生成的日志文件拷贝到主机端 kmchktool 工具所在目录下，可按照如下步骤使用 kmchktool 解析日志：

- a) 运行 kmchktool

```
./kmchktool
```

- b) 加载应用程序符号表（即带符号表信息的二进制文件）：

```
set program test
```

- c) 加载日志文件：

```
set log test_10992_ kmchk_slop.log
```

- d) 保存完整的违法信息

```
save
```

成功完成以上命令后，会在当前目录下会生成一个文本文件 test_10992_ kmchk_slop_all_info.txt。打开该文件可以看到内存实时越界检测结果，显示违法信息如下：

```
log type: kmchk_slop
endian: little endian      #标识是小端 cpu
pointer size: 32          #标识 32 位系统
slop count: 2             #标识存在两条越界信息
overflow read              : 1
overflow write             : 1
*****
kmchk_slop num: 1          #越界信息序号
slop type: overflow write  #标识是溢出类型的写越界
slop range: 0x80e8f80 -- 0x80e8fab size: 44 #产生越界操作的内存范围
source info: main.c:39 (vt_2) #产生越界操作的 PC
tid: 9325                  #线程 ID
```

```
backtrace info:          #堆栈回溯信息
    0x8054700 (__mfu_check   mf-runtime.c:2095)
    0x8053ebe (__mf_check   mf-runtime.c:1774)
    0x804848b (vt_2        main.c:32)
    0x805816a (start_thread  pthread_create.c:297)
    0x8089eee (clone        clone.S:130)
-----definition info-----
memory type: global or static variable
memory range: 0x80e8f80 -- 0x80e8fa7 size: 40
source info: main.c:11 array
tid: 9323
*****

kmchk_slop num: 2
slop type: overflow read
slop range: 0x80e8f80 -- 0x80e8fab size: 44
source info: main.c:24 (vt_1)
tid: 9324
backtrace info:
    0x8054700 (__mfu_check   mf-runtime.c:2095)
    0x8053ebe (__mf_check   mf-runtime.c:1774)
    0x8048338 (vt_1        main.c:17)
    0x805816a (start_thread  pthread_create.c:297)
    0x8089eee (clone        clone.S:130)
-----definition info-----
memory type: global or static variable
memory range: 0x80e8f80 -- 0x80e8fa7 size: 40
source info: main.c:11 array
tid: 9323
Log type: kmchk_slop
endian: little endian
pointer size: 32
slop count: 2
overflow read          : 2
*****
*****

kmchk_slop num: 1
slop type: overflow read
```

```
slop range: 0x80d055c -- 0x80d056b size: 16
variable name: a
source info: tem.c:9 (main)
tid: 10767
backtrace info:
    0x8048327 (main                tem.c:8)
    0x8056a59 (__libc_start_main   libc-start.c:231)
    0x8048131 (_start              start.S:119)
-----definition info-----
memory type: global or static variable
memory range: 0x80d055c -- 0x80d0563 size: 8
variable name: a
source info: tem.c:2
tid: 10767
*****
*****

kmchk_slop num: 2
slop type: overflow read
slop range: 0x80d055c -- 0x80d0567 size: 12
variable name: a
source info: tem.c:5 (tt)
tid: 10767
backtrace info:
    0x804829d (tt                  tem.c:4)
    0x8048334 (main                tem.c:10)
    0x8056a59 (__libc_start_main   libc-start.c:231)
    0x8048131 (_start              start.S:119)
-----definition info-----
memory type: global or static variable
memory range: 0x80d055c -- 0x80d0563 size: 8
variable name: a
source info: tem.c:2
tid: 10767
*****
```

3. 获取工具

内存检测工具的正常使用的，需要有三方面支持：

- 支持 kmemchk 的交叉工具链

由于 kmemchk 与 gcc 和 c 库紧密相关，所以必须要有工具链的特定支持才能实现检测功能。

- kmchktool 客户端日志分析工具

客户端分析工具对程序运行后产生的日志文件进行分析，并根据日志文件、符号表等相关信息，向用户提供关键信息的检索、过滤，程序故障点定位等功能。

- kmemchk 功能配置文件

可以根据需要灵活配置 kmemchk 服务端的功能。

3.1. 编译工具链的获取

如果安装了 V2.08.10_P1 及更高版本的 KIDE，在路径：<\$(KIDE_install_path)\host\ide\tools_chain>下的工具链就提供 kmemchk 功能的支持。

3.2. kmemchk 客户端工具和配置文件的获取

若安装了 V2.08.10_P1 及更高版本的 KIDE，内存检测工具所在路径为：

<\$(KIDE_install_path)\target\tools\kmemchk>，其中\$(KIDE_install_path)为 KIDE 的安装根路径。其中包含的文件有：

- kmchktool: Linux 下的客户端工具，用于解析 kmemchk 生成的日志文件；
- kmchktool.exe: Windows 下的客户端工具；
- kmchkcfig.ini: 配置文件，配置 kmemchk 的功能（在第 4 节有详细介绍）。

kmchktool 是在 Linux 编译主机 (X86-32) 上使用的客户端工具，需要在命令行下运行。kmchktool.exe 是在 Windows 下运行的客户端工具，与 kmchktool 功能一致。

4. 编译程序

内存检测工具 kmemchk 主要分为内存泄露检测和内存实时越界检测两大功能，其在编译方式上有所

区别。

4.1. 内存泄露检测

内存泄露检测所需要使用到的库文件是集成在工具链中发布的，在编译应用程序时，需要显示指定链接参数“-kmchk_leak”才能链接到应用程序中，下面将以 2.1 中的 test_leak.c 为例，介绍如何编译带内存泄露检测功能的应用程序，编译命令如下：

```
/opt/kmemchk/x86_gcc4.1.2_glibc2.5.0/bin/i686-pc-linux-gnu-gcc -g test_leak.c  
-o test_leak -kmchk_leak -static
```

或者

```
/opt/kmemchk/x86_gcc4.1.2_glibc2.5.0/bin/i686-pc-linux-gnu-gcc -g test_leak.c  
-o test_leak -kmchk_leak
```

其中带-static 参数为静态链接，否则为动态链接。动态链接时，运行程序前需要将工具链中的库文件上传到运行环境中。

另外，若是希望通过使用 KIDE 构建程序，则可通过建立 User 项目，添加源码，并在项目属性中为其添加“-kmchk_leak”链接参数即可，如图 4-1：

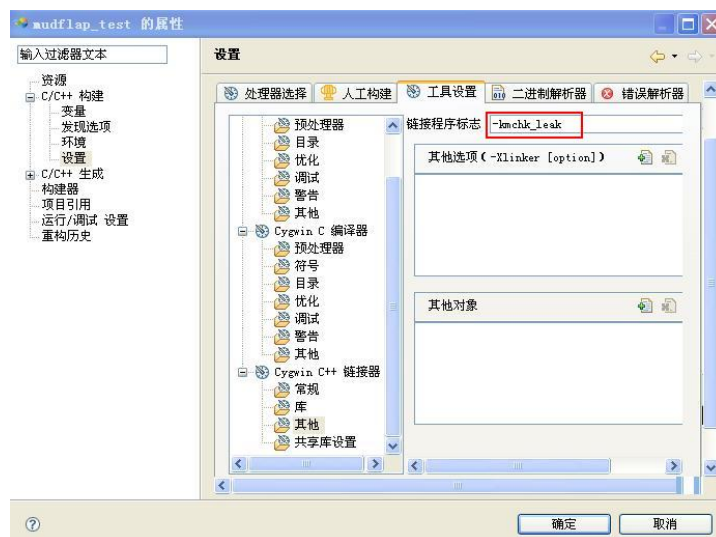


图 4-1 KIDE 中添加内存泄露检测链接参数

4.2. 内存实时越界检测

kmemchk 的内存实时越界检测功能由编译器自带，不依赖于其他任何工具，只需要在构建应用程序

时添加编译参数“-kmchk_slop”，即可开启检测功能。如下：

静态编译：

```
xxx-gcc mf_single_thread.c -o test -kmchk_slop -g -static
```

动态编译：

```
xxx-gcc mf_single_thread.c -o test -kmchk_slop -g
```

若是在 KIDE 中构建程序，则通过在项目属性中同时为其添加“-kmchk_slop”的编译参数，如图 4-2 和“-kmchk_slop”的链接参数，如图 4-3 即可。

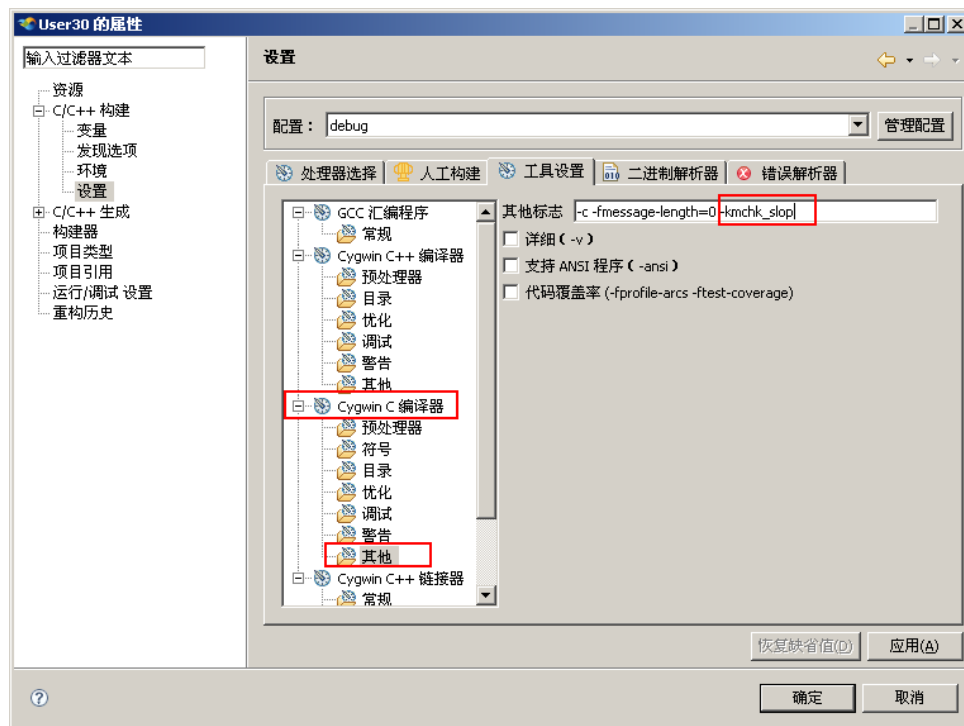


图 4-2 KIDE 中添加内存越界检测编译参数

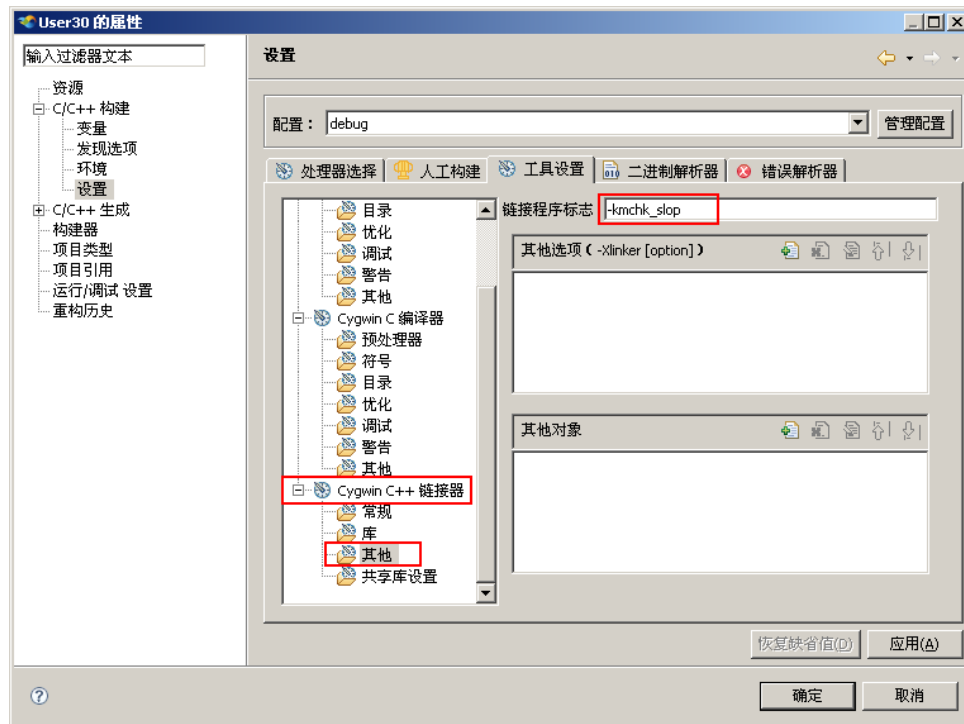


图 4-3 KIDE 中添加内存越界检测链接参数

4.3. 构建注意事项

- 1) kmchk_slop、kmchk_leak 是为简化构建参数而封装的伪参数，需要 gcc 解析。所以在链接时不能直接使用 ld、必须用 gcc 链接；
- 2) 通过静态方式构建带内存实时越界检测功能的应用程序后，可能会出现类似下面的告警信息：

warning: Using 'dlopen' in statically linked applications requires at runtime the shared libraries from the glibc version used for linking.

告警提示用户 dlopen（包含其的库）等被静态链接到应用程序中，它们（dlopen）操作的对象（比如 dlopen 加载的库）也应该使用相同版本的工具链构建。

即便应用程序未使用这些函数，该告警依然可能出现。原因是 kmchk_slop 中封装了 dlopen 等库函数，而这些封装函数调用了它们。封装这些函数，可以提高实时越界检测的精确性。如果应用程序中未使用这些函数，那么可以无视告警，它们不会对程序的执行造成影响。

- 3) -mregparm=num: 此编译参数的作用是指定传参使用的寄存器个数，将改变默认的参数传递方式，强制使用寄存器传参。由于 kmemchk 的库默认采用堆栈传参，所以在构建应用程序时若使用了 -mregparm=num 参数，应用程序和工具链的库的参数传递方式会不一致，从而程序运行时可能

会产生不可预知的错误。

5. 配置说明

内存检测工具提供了灵活的功能组合方式，用户可以通过配置项来开启或关闭某些功能。为了方便用户灵活使用，可由用户根据自己的实际需要，通过静态或动态两种方式进行配置。如果用户不进行配置，所有功能将采用默认配置进行开放或关闭。

5.1. 内存泄露检测配置项

配置项的作用是在应用程序启动前，设置相应的参数（可用配置文件或环境变量的方式）。启动应用程序时，在内存检测工具初始化流程中通过检测参数来确定开启或禁用某些功能。例如，用户不需要使用红区检测功能，而红区检测功能在应用中是默认打开的，如果直接启动应用，则该功能就会生效，因此可以在配置文件中通过设置一个关键字的值来开关该项功能。

5.1.1. 常用配置项

表 5-1 内存泄露检测常用配置项

配置项	功能描述	可能取值	约束关系	可否动态配置
KMCHK_HEAD_TACKDEPTH	配置记录的函数调用链层数。值为0则不记录，为1~64之间的整数，则最大记录对应的层数，默认值为8，表示最多只记录8层函数调用链，如果函数调用链不超过8层则记录实际层数，如果超过8层则只记录前面8层	0~64， 默认值为8	该配置项只有在打开配置项 KMCHK_MEM_HEAD（默认打开）时才生效	是
KMCHK_SIG	配置应用程序运行过程中，发送什么信号通知 kmemchk 更新配置和导出当前未释放内存信息，默认为 SIGURG，如果应用程序本身会用到 SIGURG，则必须修改该配置项，配置其它信号	信号名， 默认为 SIGURG	KMCHK_ALL 开启时生效	否

5.1.2. 其它配置项

表 5-2 内存泄露检测其它配置项

配置项	功能描述	可能取值	约束关系	可否动态配置
KMCHK_ALL	开启/禁用内存检测工具，默认开启，如果配置为0则禁用kmemchk	0/1 默认为1	无	是
KMCHK_OUTPUT_LOG	内存检测工具是否记录所有内存操作信息，如果打开该功能，kmemchk将详细记录程序每一次内存申请或释放操作，这将有助于查看内存申请或释放失败的信息，但开启该功能对性能有较大影响。默认关闭该功能，只记录未释放内存信息	0/1 默认为0	KMCHK_ALL开启时生效	是
KMCHK_DEMO_THREAD	配置开启/禁用管理线程，默认禁用，如果开启，kmemchk初始化时会创建一个管理线程，用于接收和执行kmchktool的修改配置的命令	0/1 默认为0	同上	否
KMCHK_PROF	开启/禁用统计功能，默认开启，对整个进程的内存操作进行统计，比如malloc调用次数、申请内存总大小等	0/1 默认为1	同上	是
KMCHK_THREAD	开启/禁用按线程进行统计功能，默认禁用，如果开启，则单独统计每个线程的内存操作情况	0/1 默认为0	KMCHK_PROF开启时生效	是
KMCHK_RANDOM_FAIL	开启/禁用随机分配失败功能，默认禁用，如果启用，则在内存分配时kmemchk制造一定几率的分配失败，主要用于测试应用程对内存申请失败的异常情况有无处理	0/1 默认为0	同上	是
KMCHK_MEM_HEAD	是否开启在待分配的内存头部添加保留字段，用来记录本次内存操作信息，比如申请的内存块大小、函数调用链等信息，默认开启，如果关闭该功能，则kmemchk无法进行内存泄露检测	0/1 默认为1	KMCHK_ALL开启时生效	是

KMCHK_MMAP	是否对mmap映射的内存进行跟踪，默认开启，如果关闭则不能对mmap映射的内存进行泄露检测	0/1 默认为1	KMCHK_MEM_HEAD 开启时生效	是
KMCHK_HEAD_TID	内存头部中是否记录线程tid信息，默认要记录，如果关闭，则记录的未释放内存信息中将没有线程号	0/1 默认为1	同上	是
KMCHK_HEAD_TIME	内存块头部中是否记录内存操作的系统时间，默认为1，不记录时间而是记录内存块的全局序列号，如果为0，则既不记录时间也不记录序列号，如果为2则同时记录时间和序列号	0/1/2 默认为1	同上	是
KMCHK_TIMER_SIG	在KMCHK_HEAD_TIME配置为2的情况下，配置定时器超时信号，默认为SIGALRM，如果应用程序本身会用到SIGALRM，则必需修改为其它信号	信号名，默认为SIGALRM	KMCHK_HEAD_TIME 为2时生效	否
KMCHK_TIME_INTERVAL	在KMCHK_HEAD_TIME配置为2的情况下，配置定时器超时时间间隔，单位为秒	非负整数，默认为1	同上	否
KMCHK_SNAP_INTERVAL	在KMCHK_HEAD_TIME配置为2的情况下，配置自动获取内存快照的间隔，单位为秒，默认为0，如果配置为正整数，则kmemchk每隔该时间间隔自动获取一次当前未释放内存信息	非负整数 默认为0	同上	是
KMCHK_SNAP_DETAIL	配置自动获取的快照信息中是否包含跟踪信息，默认为0，只获取统计信息，不获取跟踪信息	0/1 默认为0	同上	是
KMCHK_MAPS_SIZE	日志文件中预留用于保存进程maps信息的空间大小，单位为k，该项一般不需要修改	正整数，默认为32	同上	否

KMCHK_SLOP OVER	内存头部信息中是否开启堆一致性检测功能，默认开启，如果关闭内存泄露检测将没有附带的动态内存非实时越界检测功能	0/1 默认为1	同上	是
KMCHK_SLOP OVER_MALLO C	是否在分配内存时检查所有内存块是否越界，默认关闭，如果开启对性能有较大影响	0/1 默认为0	KMCHK_S LOPOVER 为1时生效	是
KMCHK_SLOP OVER_FREE	是否在释放内存时检查所有内存块是否越界，默认关闭，如果开启对性能有较大影响	0/1 默认为0	同上	是
KMCHK_MEM_ CHECK	动态配置后，应用重新加载配置文件，该配置项指定应用是否需要检测所有内存块是否越界，默认开启	0/1， 默认为1	KMCHK_M EM_HEAD 开启时生效	是
KMCHK_PRINT _PROFINFO	在内存检测工具重新动态配置后，是否将统计信息打印到终端上，默认不打印	0/1， 默认为0	KMCHK_P ROF开启时 生效	是

5.1.3. 配置方式

目前提供了两种配置方式，

➤ 通过配置文件进行配置

可在程序运行前通过编辑配置文件 kmchkefg.ini 修改各配置，程序启动过程中会自动读取该文件并初始化 kmemchk 的各功能；也可以在程序运行过程中修改该配置文件，然后通过 kill 命令发送指定的信号给进程，通知 kmemchk 更新配置并重新初始化一些功能。



提示：只有部分配置项能够在程序运行过程中重新配置，其他配置项只能在程序运行前完成配置，

具体哪些配置项可以在程序运行过程中重新配置，请参见表 5-1 和表 5-2 中“是否可重新配置”一栏。

- 通过 kmchktool 工具进行远程在线配置，详见 7.2.1.5 节 config 命令。只有部分配置项可以进行远程在线配置，请参见表 5-1 和表 5-2 中“是否可重新配置”一栏。

5.1.4. 配置文件实例

```
#####
#          kmemchk 工具配置文件          #
#####

#####

# 总开关，应用程序是否开启内存检测工具
# 取值：
#      0---禁用内存检测工具
#      1---启用内存检测工具
#####
KMCHK_ALL=1

#####

# 记录未释放内存信息还是所有内存操作详细信息
# 取值：
#      0---记录未释放内存信息
#      1---记录所有内存操作详细信息
#####
KMCHK_OUTPUT_LOG=0

#####

# 是否创建管理线程
# 取值：
#      0---不创建管理线程
#      1---创建管理线程
#####
KMCHK_DEMO_THREAD=0

#####

# 指定触发导出日志的信号
# 取值：
#      信号编号或者信号名，默认信号为 SIGURG
```

```
#####
KMCHK_SIG= SIGURG

#####

# 内存统计信息功能开关
# 取值:
#      0---关闭统计功能
#      1---启用统计功能
#####
KMCHK_PROF=1
#####
# 是否按线程对程序内存使用情况进行统计
# 取值:
#      0---只以进程为单位进行统计
#      1---除了以进程为单位进行统计, 还要分线程进行统计
# 说明: 该配置项只在 KMCHK_PROF 配置为 1 的情况下生效
#####
KMCHK_THREAD=0

#####
# 是否跟踪 mmap 信息开关
# 取值:
#      0---不跟踪
#      1---跟踪 mmap 分配的内存信息
#####
KMCHK_MMAP=1
#####
# 随机分配失败功能
# 取值:
#      0---关闭该功能
#      1---开启该功能
#####
KMCHK_RANDOM_FAIL=0

#####
##### 内存泄露、跟踪、溢出检测功能 #####
#####
```



```

#内存泄露、跟踪、溢出检测功能功能开关
#####
# 是否在内存块头部申请额外空间保存信息
# 取值:
#      0---不申请额外空间
#      1---申请额外空间来保存一次内存操作信息
# 说明: 如果该项配置为 0, 则 kmemchk 就关闭了跟踪功能,
#      只能对内存操作进行统计。
#####
KMCHK_MEM_HEAD=1
#####
# 更是否记录内存操作的线程号
# 取值:
#      0---不记录线程号
#      1---记录线程号
#####
KMCHK_HEAD_TID=1

#以下三个选项配置获取内存操作时间相关功能#
#####
# 是否开启时间字段
# 取值:
#      0---关闭时间字段
#      1---只获取序列号
#      2---获取序列号, 同时还要获取时间
#####
KMCHK_HEAD_TIME=1
#####
# 在配置了获取系统时间时, 指定定时器到期时产生何种信号
# 取值:
#      信号编号或者信号名, 默认信号为 SIGALRM
#####
KMCHK_TIMER_SIG=SIGALRM
#####
# 在开启时间字段的时候, 配置定时器超时间隔
# 取值:
#      0---不创建定时器

```

```

#      正整数---定时器超时时间间隔，单位为秒。
#####
KMCHK_TIME_INTERVAL=1
#####
# 自动获取内存快照的时间间隔
# 取值：
#      0---不定时获取内存快照
#      正整数---定时获取内存快照的时间间隔，单位为秒。
#####
KMCHK_SNAP_INTERVAL=0
#####
# 自动获取内存快照时，是否获取详细信息,默认为 0
# 取值：
#      0---只获取统计信息
#      1---获取统计信息和跟踪信息
#####
KMCHK_SNAP_DETAIL=0

#0 为关闭 1 为默认堆栈层数 8 最大为 64 层
#####
# 配置堆栈回溯支持的最大层数，不超过 64 层
# 取值：
#      0---不获取堆栈回溯信息
#      1---默认最大记录 8 层堆栈回溯信息
#      2-64---堆栈回溯支持的最大层数
#####
KMCHK_HEAD_STACKDEPTH=1
#####
# 日志文件中预留用于保存 maps 信息的空间大小
# 取值：
#      大于 0 的整数,默认值为 32
#####
KMCHK_MAPS_SIZE=32

#####
# 是否支持越界检测
# 取值：

```

```
#      0---不支持越界检测
#      1---支持越界检测

#####
KMCHK_SLOPOVER=1

#####

# 在分配内存前是否检测所有未释放的内存是否发生越界
# 取值:
#      0---不检测
#      1---检测

#####
KMCHK_SLOPOVER_MALLOC=0

#####

# 在释放内存前是否检测所有未释放的内存是否发生越界
# 取值:
#      0---不检测
#      1---检测

#####
KMCHK_SLOPOVER_FREE=0

#####

# 在更新配置后是否检测所有未释放的内存是否发生越界
# 取值:
#      0---不检测
#      1---检测

#####
KMCHK_MEM_CHECK=1

#####

# 在发送信号导出日志后是否在终端打印输出统计信息
# 取值:
#      0---不打印
#      1---打印

#####
KMCHK_PRINT_PROFINFO=0
```

5.2. 内存实时越界检测配置项

内存实时越界检测功能提供运行环境配置方式。通过设置运行环境的环境变量来修改配置选项，可以控制 kmchk_slop 的检测范围、信息输出方式和检测行为。

5.2.1. 配置日志文件的生成路径

配置方式：

```
export KMCHK_SLOP_FILE_PATH='xxx-path'
```

其中 xxx-path 为新指定的日志路径，例如：

```
export KMCHK_SLOP_FILE_PATH='/home/test'
```

备注：若不指定日志路径，则日志文件默认保存在程序当前所在路径。

5.2.2. 配置写文件的信号

配置方式：

```
export KMCHK_SLOP_WRITE_FILE_SIG='signal-name'
```

其中 signal-name 为信号名，例如：

```
export KMCHK_SLOP_WRITE_FILE_SIG='SIGUSR1'
```

5.2.3. 功能配置

5.2.3.1. 配置方式

功能配置是通过设置目标端（被检测程序的运行端）环境变量 KMCHK_SLOP_OPTIONS 实现配置的。如设置 -viol-abort，则 kmchk_slop 检测到越界操作后调用 abort 终止程序功能。需要在程序运行前设置其环境变量，如下：

```
export KMCHK_SLOP_OPTIONS='-viol-abort'
```

然后运行程序，若检测到违法操作，那么 kmchk_slop 就会调用 abort 终止进程。

如果需要关闭某项配置，在该配置选项前加 -no 即可，如下：

```
export KMCHK_SLOP_OPTIONS='-no-record-alloc-backtrace'
```

如果需要同时配置多项功能，使用空格分隔各个配置项即可，如下：

```
export KMCHK_SLOP_OPTIONS='-no-record-alloc-backtrace -report-unfind
-collect-stats'
```

如上所示，该配置关闭了分配内存时纪录堆栈回溯信息功能，并同时打开了输出 unfind 类型越界检测信息和输出统计信息的功能。

5.2.3.2. 默认配置项

表 5-3 内存实时越界检测默认配置项

配置项	功能描述	默认配置
-mode-check	配置具有实时内存越界检测功能	是
-heur-stdlib	配置监控(argv, errno, stdin, ...)内存功能，减少误报，提高检测精确度	是
-output-as-file	将检测信息保存为日志文件。用户可以通过后端工具解析此日志文件，获得更精确的故障定位	是
-record-alloc-backtrace	纪录用户分配内存处的堆栈回溯信息，便于故障定位。但是会影响软件运行效率，如应用程序对效率要求较高，可选择关闭该配置	是

5.2.3.3. 可选配置项

表 5-4 内存实时越界检测可选配置项

配置项	功能描述	默认配置
-mode-violate	将所有检测范围内的内存操作视为越界，输出检测信息	否
-viol-abort	检测到越界操作后调用abort终止程序	否
-viol-segv	检测到越界操作后触发SIGSEGV信号	否
-viol-gdb	检测到越界操作后，启动gdb调试程序	否
-collect-stats	程序退出前，通过标准错误输出打印检测的统计信息	否
-check-initialization	探测未初始化的动态内存读操作，并输出检测信息	否
-ignore-reads	忽略读操作的检查	否
-heur-proc-map	只要是maps范围内的内存操作都是合法的	否

-heur-stack-bound	栈空间内的内存操作都是合法的	否
-heur-start-end	_start到_end范围内（即应用程序的入口地址到结束地址之前的范围）的内存操作都是合法的	否
-print-violation	检测信息会通过标准错误输出打印到屏幕上	否
-report-unfind	输出unfind类型的越界检测信息。unfind类型的越界是由于程序操作了未分配的内存区域，或操作的内存区域没有被kmemcheck注册	否
-out-of-memory	配置后，工具可以在内存耗尽环境下继续运行并输出检测信息	否
-keep-memory-size	配置-out-of-memory后生效，设置预留给工具的内存大小(单位Mbytes)。工具会在运行环境内存耗尽后切换使用预留的内存保证工具正常工作。默认预留内存为10M	否

提示：使用-report-unfind 配置项时，在某些情况下会引起误检，如：为全局变量 A 分配内存空间的代码放在 a 文件中，而对 A 进行读写操作的代码放在 b 文件中。如果编译代码时只对 b 文件加了内存实时越界编译参数“-kmchk_slop”，而未对 a 文件加该编译参数，则会误报对 A 进行读写操作是违法的。这种情况下，需将 a、b 文件都加上参数“-kmchk_slop”进行编译。

6. 运行使用

6.1. 内存泄露检测

6.1.1. 运行程序

如果程序为动态链接，运行程序前需要将编译工具链 lib 目录下的动态库拷贝到运行环境中；如果需要修改默认配置，则可根据需要设置配置文件 kmchkcfig.ini 中需要修改的配置项，然后将编辑好的配置文件 kmchkcfig.ini 拷贝到运行环境中，与应用程序放到同一个目录下，然后就可以启动程序，获取日志。

一般需要关注以下两项主要配置：

➤ KMCHK_SIG：用于指定发送哪个信号给 kmemchk，通知其导出日志，默认为 SIGURG，如果

应用程序本身会用到 SIGURG，那么必需修改该项配置，指定使用其他信号来通知 kmemchk 导出日志；

➤ **KMCHK_HEAD_STACKDEPTH**：用于指定堆栈回溯层数，最大支持 64 层，默认 8 层，表示最多只记录 8 层函数调用链，如果函数调用链不超过 8 层则记录实际层数，如果超过 8 层则只记录前面 8 层。如果 8 层不能满足要求，则可以修改该值。

其它配置项一般使用默认值即可，关于配置文件中各配置项的详细说明请参见第 4 章。

6.1.1.1. 静态程序的运行

静态程序对运行环境没有太强的依赖，所以一般可以直接运行。

6.1.1.2. 动态程序的运行

由于 kmemchk 依赖于工具链支持，所以需要确保运行环境的库与构建应用程序的工具链具有兼容性，特别是 c 库、pthread 库和 libkmemchk.so 这 3 个库一定要与工具链的库一致。如果运行库与工具链库不一致，在嵌入式环境下，一般直接 copy 工具链的库文件覆盖运行环境相应库文件即可（比如 x86 工具链，其库文件在 x86_gcc4.1.2_glibc2.5.0\i686-pc-linux-gnu\lib 目录下，运行环境的库文件一般在 /lib 或 /lib64 下）。如果在 cgs1 服务器环境下运行可以直接利用脚本，详见第 9 节。

6.1.2. 获取日志

在程序运行过程中如果要查看日志，则只需发送 **KMCHK_SIG** 配置项指定的信号给进程即可。kmemchk 收到信号后就将进程当前占用的每一块动态申请的内存信息写到日志文件中，日志文件的名称为“程序名_进程号_kmchk_leak.log”，如果配置项 **KMCHK_PROF** 与 **KMCHK_PRINT_PROFINFO** 同时启用的话，此时 kmemchk 还会打印输出此刻进程的动态内存操作统计信息，例如图 6-1：

```
##### Program memory prof info #####
calls count      failed count      total size(byte)
malloc           214              0          57682193
realloc          200              0           15150
free             21              0       115343365
memalign         110              0       57676730
mmap              5              0       33566848
munmap           2              0        1048576
unfree mem size 32548980
unfree mem peak 43014540
```

图 6-1 发送信号时 kmemchk 输出的内存统计信息

统计信息各字段的详细解释请参见 8.1.1。

发送信号后将生成的日志文件上传到主机，就可以使用 kmchktool 工具查看日志了。内存泄露检测的基本原理就是 kmemchk 端将进程未释放的动态内存信息记录到日志文件，然后通过 kmchktool 解析日志

文件，对日志文件中的信息进行过滤、排序等，以便辅助开发人员分析一块未释放的内存块是否发生泄露。最常见的内存泄露为累加性的泄露，即程序每执行一次某个流程就会产生一些未释放的内存，对于这种情况最有效的分析办法就是每隔一段时间获取一份日志，然后通过这些日志分析是否存在某个流程申请的且未释放的内存呈递增的趋势，如果存在，则这样的流程就很可能产生内存泄露。因此在程序运行过程中最好多获取几份日志。关于 kmchktool 的使用请参见第 7 章。

此外，内存泄露检测功能还提供了自动定时获取日志信息功能。如果启用该功能，则 kmemchk 会每隔一段时间自动获取一份日志，并将这些日志按获取时间顺序保存在一个文件中，文件名为“程序名_进程号_kmchk_leakautosnap.log”。要启用该功能，Kmemchk 的配置文件需按下述方式进行设置：

KMCHK_HEAD_TIME=2	//打开记录时间功能
KMCHK_TIMER_SIG=SIGALRM	//设置定时器超时信号，默认为 SIGALRM，如果应用程序会用到该信号，则必需设置为其它信号
KMCHK_TIME_INTERVAL=interval1	//定时器超时时间间隔，单位为秒，默认为 1 秒
KMCHK_SNAP_INTERVAL= interval2	//自动获取日志信息的时间间隔，单位为秒，默认为 0，即不自动获取，设置为其它正整数，则每隔该时间就自动获取一份日志。该间隔不能设置太小且 interval2 应大于 interval1，否则对性能有较大影响。
KMCHK_SNAP_DETAIL=1	//获取每块内存的详细信息，如果设置为 0，只获取统计信息

开启上述配置后，kmemchk 会每隔一段时间自动获取一份日志，无需手动发送信号获取日志。该功能与手动发送信号获取日志功能不冲突，仍然可以使用前面介绍的手动发送信号获取。

6.2. 内存越界检测

6.2.1. 运行程序

如果程序为动态链接，运行程序前需要将编译工具链 lib 目录下的动态库拷贝到运行环境中；如果需要特殊配置，则在运行前，设置环境变量 KMCHK_SLOP_OPTIONS（详见 5.2）。所有准备工作设置好后，运行程序。

6.2.1.1. 静态程序的运行

静态程序对运行环境没有太强的依赖，所以一般可以直接运行。

6.2.1.2. 动态程序的运行


由于 kmemchk 依赖于工具链支持，所以需要确保运行环境的库与构建应用程序的工具链具有兼容性，

特别是 c 库、pthread 库和 libslop.so 这 3 个库一定要与工具链的库一致。如果运行库与工具链库不一致，在嵌入式环境下，一般直接 copy 工具链的库文件覆盖运行环境相应库文件即可（比如 x86 工具链，其库文件在 x86_gcc4.1.2_glibc2.5.0\i686-pc-linux-gnu\lib 目录下，运行环境的库文件一般在/lib 或/lib64 下）。如果在 cgs1 服务器环境下运行可以直接利用脚本，详见第 9 节。

6.2.2. 统计信息实时查看

如果配置了 **-print-violation** 功能，当程序发生违法操作，就会立刻打印违法信息（信息含义参见 8.2.1），如下所示：

```
kmchk_slop violation 1
violation type:  read(check)
violation rang: 0x81035fc -0x810360b size: 16
pc=0x8048325  location='test.c:9 (main)'
backtrace:
    0x804a3e2
    0x804b707
    0x8048325
    0x804a9e6
    0x80528b9
    0x8048131
```

 **提示：** -print-violation 功能输出的检测信息没有符号信息，只能提供一次的越界情况，不便于定位。

需要准确定位故障可以选择生成日志文件，交由后端工具解析。

6.2.3. 获取日志文件

默认情况下，在程序退出后，会在可执行程序目录下生成一个名为“程序名_进程号_kmchk_slop.log”的日志文件。该日志文件记录了违法检测信息，需要交由后端工具 kmchktool 进行解析。

同时，用户可以通过对应用程序发送指定信号的方式更新日志文件，默认信号为 SIGURG，命令如下：

```
kill - SIGURG pid
```

7. kmchktool 工具

kmchktool 工具是一款配合内存检测工具使用的日志分析、参数配置软件。**kmchktool** 可通过分析内存检测工具生成的日志文件，从而实现对程序运行过程中的内存操作进行分析统计，并最终判断出是否发生了内存泄漏、内存越界等信息，并给出内存分配释放的详细调用关系。

7.1. 使用方法

kmchktool 工具提供在 X86 平台上对各体系架构目标机运行的程序进行分析的功能，用户可直接对程序运行中生成的跟踪日志文件进行分析操作。可按照如下步骤使用内存检测工具：

- 1) 运行 **kmchktool**;
- 2) 执行 **set program** 命令指定应用程序符号表;
- 3) 执行 **set libpath** 指定库文件路径（如果是静态链接的程序则不需要指定）;
- 4) 执行 **set log** 命令指定分析目标日志文件，该命令必须执行;
- 5) 使用各命令查看相应信息，对于内存泄露检测和内存实时越界检测的查看命令不一样

以上 **set** 命令对于内存泄露检测和内存实时越界检测是一致的。以下是各步骤的详细说明。

7.1.1. 运行工具程序

kmchktool 目前支持在 X86 32 位平台下的 Linux 系统或 cygwin 下运行。在命令行操作界面下添加可执行权限 **chmod +x kmchktool** 后，执行如下命令即可运行 **kmchktool** 工具：

```
[root@localhost kmemcheck]# ./target/kmchktool
(kmem)
(kmem) █
```

图 7-1 启动 **kmchktool** 工具

7.1.2. 指定应用程序符号表

指定应用程序的符号表，可以保证 **kmchktool** 工具正常地查找符号，以便回溯堆栈。如果不指定，则堆栈回溯信息中只有地址，没有函数符号。指定符号表文件是使用 **set program** 命令进行的，命令格式如下：

```
set program program_file
```

7.1.3. 指定库文件路径

如果是动态链接的应用程序，客户端工具需要得到程序使用的库信息及库路径，如果不指定，则堆栈回溯信息中对于库函数中的地址，将没有函数符号。如果是静态链接的程序则不需该操作。命令格式如下：

```
set libpath path
```

7.1.4. 指定分析目标日志文件

在指定了应用程序的符号表文件和库路径后，还需要使用 **set log** 命令指定待分析的日志文件（应用程序运行过程中生成的后缀为.log 的文件）。命令格式如下：

```
set log filename
```

例如，目标文件 test_leak_768_kmchk.log 与 kmchktool 存放在同一目录下，那么输入命令：

```
set log test_leak_768_kmchk.log
```

在完成上述设置后，就可以执行具体的查看命令查看各类信息，具体的查看命令请参见 [7.2.1](#) 和 [7.2.2](#)。如果需要分析多个目标文件，可以在 kmchktool 不退出的情况下，重新指定目标日志文件即可。

7.2. 命令介绍

7.2.1. kmchk_leak 命令

7.2.1.1. set 命令

7.2.1.1.1. set program 设置符号表文件

功能：

用于设置目标应用程序可执行文件的本地路径。由于在进行堆栈信息解析的时候，需要指定程序的符号表进行符号分析，如果没有指定对应的目标程序路径，那么堆栈回溯信息中就只有地址没有符号。另外，如果应用程序被 **strip** 掉了行号信息，那么堆栈回溯定位行号也将不能够正确使用。

命令格式：

```
set program programname
```

举例说明：

```
(kmem) set program test_leak
Specify a program : test_leak
(kmem)
```

图 7-2 set program 实例

7.2.1.1.2. set libpath 设置库路径

功能：

对于动态链接的应用程序，如果分析日志的环境与程序的运行环境不是同一个环境，则还需要指定库路径，以便客户端分析工具能找到库并加载其符号表。

命令格式：

```
set libpath path
```

举例说明：

```
(kmem) set libpath /home/debugger/kcov/lib/
solib path is /home/debugger/kcov/lib
(kmem)
```

图 7-3 set libpath 命令实例

7.2.1.1.3. set log 设置日志文件路径

功能：

用来指定目标应用程序生成的跟踪日志的文件名称和路径，**kmchktool** 将根据这个日志文件进行内存信息统计与分析。

命令格式：

```
set log filename
```

举例说明：

```
(kmem) set log test_leak_23456_kmchk.log
Specify a log file : test_leak_23456_kmchk.log
(kmem)
```

图 7-4 set log 命令实例

7.2.1.1.4. set offset 设置地址偏移

功能:

在 ARM NOMMU 上, 由于 ELF 符号表文件的地址和实际 FLT 格式在内存中的地址有一定偏移, 造成符号解析工具不能正确地解析符号, 这里需要有人工设置应用程序代码段 TEXT 在内存中的偏移地址。该偏移地址在运行 ARM NOMMU 程序的时候, 由操作系统打印在终端。用户需要记录该地址, 并在分析日志时输入地址。

命令格式:

```
set offset addr
```

7.2.1.2. show 命令

7.2.1.2.1. show all 导出所有信息

功能:

将程序日志中所有的信息, 解析到分析日志文件中。分析日志文件是以程序日志文件名+all_info.txt 命令的文件。

命令格式:

```
show all [-snap number] [-program program] [-log log_file]
[-libpath path] [-offset addr]
```

参数说明:

- -snap number : 对于定时自动获取的内存信息, 该参数用于指定快照编号, 其中 number 为日志编号, 如果不指定, 就显示第一份日志的信息
- -program program: 指定应用程序可执行文件并覆盖之前通过该参数或者 set program 指定的文件, 功能与 set program 相同
- -log log_file: 指定 kmemchk 生成的日志文件并覆盖之前通过该参数或者 set log 指定的文件, 功能与 set log 相同
- -libpath path: 指定动态库路径并覆盖之前通过该参数或者 set libpath 指定的路径, 功能与 set libpath 相同
- -offset addr: 指定 armmommu 代码段在内存中的偏移地址并覆盖之前通过该参数或者 set offset 指定的偏移地址, 功能与 set offset 相同

举例说明:

```
(kmem) show all
All memory info log have been saved to file
test_leak_23456_kmchk.log_all_info.txt
```

图 7-5 show all 用例

```
test_leak_23456_kmchk.log
test_leak_23456_kmchk.log_all_info.txt
```

图 7-6 日志文件和分析结果日志文件

解析得到的结果日志文件格式详见 8.1.2 节。

7.2.1.2.2. show failed 显示内存操作失败情况

功能:

show failed 命令用于显示内存分配或释放时出现失败的信息。

命令格式:

```
show failed [-t tid] [-s low;high] [-a low;high] [-d low;high] [-f funcname] [-o
(size/time/addr)] [-p] [-l logfile] [-n low;high] [-snap number] [-program
program] [-log log_file] [-libpath path] [-offset addr]
```

参数说明:

- -t tid: 只显示线程号 tid 失败情况, 其中 tid 可以指定多个, 以逗号间隔;
- -s low;high: 只显示 low 到 high 大小区间的内存操作失败情况;
- -a low;high: 只显示 low 到 high 地址区间的内存操作失败情况;
- -d low;high: 只显示 low 到 high 时间区间的内存操作失败情况;
 - ✧ 时间格式为: 年-月-日-时:分:秒;年-月-日-时:分:秒
- -f funcname: 只显示函数调用链中包含 funcname 参数指定的函数的内存操作失败情况;
- -o (size/time/addr) : 按大小或时间或地址从小到大排序方式显示内存操作失败信息;
- -p: 逐条显示结果, 没有该参数则按函数调用链进行统计显示;
- -l logfile: 将解析信息输出到指定的日志文件;
- -n low;high: 只显示 low 到 high 序列号 (即内存操作按时间顺序的编号) 区间的内存操作失败

情况;

- -snap number: 对于定时自动获取的内存信息, 该参数用于指定快照编号, 其中 number 为日志编号, 如果不指定, 就显示第一份日志的信息;
- -program program: 指定应用程序可执行文件并覆盖以前通过该参数或者 set program 指定的文件, 功能与 set program 相同;
- -log log_file: 指定 kmemchk 生成的日志文件并覆盖以前通过该参数或者 set log 指定的文件, 功能与 set log 相同;
- -libpath path: 指定动态库路径并覆盖以前通过该参数或者 set libpath 指定的路径, 功能与 set libpath 相同;
- -offset addr: 指定 armnmmu 代码段在内存中的偏移地址并覆盖以前通过该参数或者 set offset 指定的偏移地址, 功能与 set offset 相同。

➡ 提示: 参数中 low 与 high 之间, 需要使用','分号作为分隔符

举例说明:

```
(kmem) show failed
No failed memory operation info found, please check if
KMCHK_OUTPUT_LOG item has been configured to 1, if this item
is 0, then kmemchk will not record memory operation failed info.
(kmem)
```

图 7-7 没有内存操作失败时的信息

```
(kmem) show failed
Memory operation failed occurs !
tid : 23072
time : 2010-07-14-13:35:14.725086
memory operation : malloc
size : 4294967295
start_addr : 0x0

tid : 23072
time : 2010-07-14-13:35:14.725217
memory operation : free
size : 0
start_addr : 0x0
```

图 7-8 存在内存分配或释放失败的信息

显示信息内容说明：

- tid: 线程 id 号
- time: 内存操作时间
- memory operation: 内存操作函数
- size: 内存请求大小
- start_addr : 内存起始地址

注意：默认 kmemchk 只记录未释放内存信息，如果要查看内存操作失败的信息，则必需在运行程序前将配置文件中的 KMCHK_OUTPUT_LOG 一项设置为 1，即打开记录内存操作详细日志功能。

7.2.1.2.3. show slop 显示非实时越界检测信息

功能：

显示内存有越界情况的内存信息。

命令格式：

```
show slop [-t tid] [-s low;high] [-a low;high] [-d low;high] [-f funcname] [-o  
(size/time/addr)] [-p] [-l logfile] [-n low;high] [-snap number] [-program  
program] [-log log_file] [-libpath path] [-offset addr]
```

参数说明：

- -t tid: 只显示线程号 tid 失败情况，其中 tid 可以指定多个，以逗号间隔
- -s low;high : 只显示 low 到 high 大小区间的越界内存情况
- -a low;high : 只显示 low 到 high 地址区间的越界内存情况
- -d low;high : 只显示 low 到 high 时间区间的越界内存情况
 - ◇ 时间的格式为：年-月-日-时:分:秒;年-月-日-时:分:秒
- -f funcname : 只显示函数调用链中包含 funcname 参数指定的函数的越界内存情况
- -o (size/time/addr) : 以大小或时间或地址方式从小到大排序方式显示越界内存信息
- -p : 逐条显示结果，没有该参数则按函数调用链进行统计显示
- -l logfile : 将解析信息输出到指定的日志文件
- -n low;high : 只显示 low 到 high 序列号区间的越界内存情况
- -snap number : 对于定时自动获取的内存信息，该参数用于指定快照编号，其中 number 为日志

编号，如果不指定，就显示第一份日志的信息

- `-program program`: 指定应用程序可执行文件并覆盖以前通过该参数或者 `set program` 指定的文件，功能与 `set program` 相同
- `-log log_file`: 指定 `kmemchk` 生成的日志文件并覆盖以前通过该参数或者 `set log` 指定的文件，功能与 `set log` 相同
- `-libpath path`: 指定动态库路径并覆盖以前通过该参数或者 `set libpath` 指定的路径，功能与 `set libpath` 相同
- `-offset addr`: 指定 `armnommu` 代码段在内存中的偏移地址并覆盖以前通过该参数或者 `set offset` 指定的偏移地址，功能与 `set offset` 相同

注意：此处的越界是指 `kmchk_leak` 模块的非实时越界检测功能检测到的被改写的内存块的信息，与 `kmchk_slop` 的实时越界检测功能无关。

 提示：参数中 `low` 与 `high` 之间，需要使用 `','` 分号作为分隔符

举例说明：

```
(kmem) show slop
Memory slop over occurs !
tid : 18789
time : 2010-07-13-19:59:40.46201
memory operation : malloc
size : 10
start_addr : 0x91f4d58
```

图 7-9 显示覆盖信息

显示信息内容说明：

- `tid`: 线程 id 号
- `time`: 内存操作时间
- `memory operation`: 内存操作函数
- `size` : 内存请求大小
- `start_addr`: 内存起始地址

其中，需要打开配置项 `KMCHK_HEAD_TID`（默认打开）才能正确显示线程号，否则 `tid` 显示为 0；

需要将配置项 `KMCHK_HEAD_TIME` 设置为 2（默认为 1）才能正确显示时间，否则显示默认显示内存操作的按时间顺序的编号；需要将 `KMCHK_HEAD_TACKDEPTH` 配置为非 0（默认打开）才会记录内存操作函数调用链信息。关于配置项的详细说明参见第 5 章。

7.2.1.2.4. show unfree 显示未释放内存信息

功能：

`show unfree` 命令显示日志文件中记录的所有未释放的内存块的信息。

命令格式：

```
show unfree [-t tid] [-s low;high] [-a low;high] [-d low;high] [-f funcname] [-o
(size/time/addr)] [-p] [-l logfile] [-n low;high] [-r] [-snap number] [-program
program] [-log log_file] [-libpath path] [-offset addr]
```

参数说明：

- `-t tid`：只显示线程号 `tid` 失败情况，其中 `tid` 可以指定多个，以逗号间隔
- `-s low;high`：只显示 `low` 到 `high` 大小区间的未释放内存块情况
- `-a low;high`：只显示 `low` 到 `high` 地址区间的未释放内存块情况
- `-d low;high`：只显示 `low` 到 `high` 时间区间的未释放内存块情况
 - ✧ 时间的格式为：年-月-日-时:分:秒;年-月-日-时:分:秒
- `-f funcname`：只显示 `funcname` 参数函数的越界内存情况
- `-o (size/time/addr)`：按大小或时间或地址方式从小到大排序方式显示未分配内存情况
- `-p`：逐条显示结果，没有该参数则按函数调用链进行统计显示
- `-l logfile`：将解析信息输出到指定的日志文件
- `-n low;high`：只显示 `low` 到 `high` 序列号区间的未释放内存块情况
- `-r`：指定本次信息查找从上次查找结果中查找
- `-snap number`：对于定时自动获取的内存信息，该参数用于指定快照编号，其中 `number` 为日志编号，如果不指定，就显示第一份日志的信息
- `-program program`：指定应用程序可执行文件并覆盖以前通过该参数或者 `set program` 指定的文件，功能与 `set program` 相同
- `-log log_file`：指定 `kmemchk` 生成的日志文件并覆盖以前通过该参数或者 `set log` 指定的文件，功能与 `set log` 相同
- `-libpath path`：指定动态库路径并覆盖以前通过该参数或者 `set libpath` 指定的路径，功能与 `set`

libpath 相同

- -offset addr: 指定 armnmmu 代码段在内存中的偏移地址并覆盖以前通过该参数或者 set offset 指定的偏移地址，功能与 set offset 相同

➡ 提示：参数中 low 与 high 之间，需要使用','分号作为分隔符

举例说明：

在不加任何参数的情况下，show unfree 默认按函数调用链对内存块进行统计显示，如下：

```
RecordNo : 2
type: malloc
count: 10
size: 550
call trace :
0 : 0x806e19f  <__libc_malloc>
1 : 0x8048310  <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a  <start_thread>
3 : 0x807276e  <clone>
```

图 7-10 show unfree 基本信息

其中各字段函数如下：

- RecordNo: 信息编号，该编号是解析工具为每条信息编的号，不是 kmemchk 按时间顺序对内存操作编的号，此处为 2，表示第 2 条信息；
- type: 内存块是通过哪个标准内存申请库函数申请的，上例中为 malloc；
- count: 在同一个函数调用链下有多少未释放的内存块，上例中为 10 块；
- size: 在同一个函数调用链下未释放的内存块的总大小，上例中为 550 个字节；
- call trace: 内存块是在哪个函数调用链下申请的；

如果要显示每块内存的详细信息，可加上-p 参数，如下：

```
RecordNo : 2
tid : 11140
seq_num : 3
memory operation : malloc
size : 10
start_addr : 0x8f6b040
call trace :
0 : 0x806e19f  <__libc_malloc>
1 : 0x8048310  <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a  <start_thread>
3 : 0x807276e  <clone>
```

图 7-11 show unfree -p 详细信息

其中各字段函数如下：

- RecordNo: 信息编号，该编号是解析工具为每条信息编的号，不是 kmemchk 按时间顺序对内存操作编的号，此处为 2，表示第 2 条信息；
- seq_num: 序列号，该序列号是 kmemchk 按时间顺序对内存操作编的号；
- tid: 线程号，申请该内存块的线程 id；
- memory operation: 内存块是通过哪个标准内存申请库函数申请的，上例中为 malloc；
- size: 该内存块的大小，上例中为 10 个字节。注意该字段与不加-p 参数时的 size 字段意义不同；
- start_addr: 该内存块的起始地址；
- call trace: 内存块是在哪个函数调用链下申请的；

如果输出信息比较多，不便于在在终端上查看，可使用-l 参数指定将 show unfree 的执行结果保存到一个文件中，同时只输出未释放内存按大小进行的一个简单统计信息，如下：

```
(kmem) show unfree -l unfree.txt
There are unfree memory blocks.

Prof info of memory blocks is as follows:

Level   size(byte)      unfree info num
0        0 - 99          9
1       100 - 999       2
2      1000 - 9999      0
3     10000 - 99999     0
4      Large          1
All info of unfreed memory blocks have been saved to file
```

图 7-12 日志信息输出到文件，只打印未释放内存按大小统计的信息

其中的统计信息意义如下：

未释放的内存块大小在 0-99 字节范围内的有 9 块，在 100-999 字节范围内内存的有 2 块；在 1000-9999 字节范围内的有 0 块，10000-99999 字节范围内的有 0 块，大于 100000 字节的有 1 块。

执行 show unfree 命令时还可以指定大小、起始地址、时间或序列号等过滤参数，以便按需查看信息，如下：

```
(kmem) show unfree -s 10;20
There are unfree memory blocks.

RecordNo : 1
type: malloc
count: 2
size: 30
call trace :
0 : 0x806e19f <__libc_malloc>
1 : 0x8048310 <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a <start_thread>
3 : 0x807276e <clone>
```

图 7-13 show unfree 按大小过滤

```
(kmem) show unfree -a 0x8f6b278;0x8f6b2f8
There are unfree memory blocks.

RecordNo : 1
type: malloc
count: 2
size: 150
call trace :
0 : 0x806e19f <__libc_malloc>
1 : 0x8048310 <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a <start_thread>
3 : 0x807276e <clone>
```

图 7-14 show unfree 按起始地址过滤

```
(kmem) show unfree -n 7;8
There are unfree memory blocks.

RecordNo : 1
type: malloc
count: 2
size: 110
call trace :
0 : 0x806e19f <__libc_malloc>
1 : 0x8048310 <thread1 /opt/test_leak.c:31>
2 : 0x8054f6a <start_thread>
3 : 0x807276e <clone>
```

图 7-15 show unfree 按序列号过滤

以上信息中，需要打开配置项 KMCHK_HEAD_TID（默认打开）才能正确显示线程号，否则 tid 显示为 0；需要将配置项 KMCHK_HEAD_TIME 设置为 2（默认为 1）才能显示时间，否则显示默认显示内存操作的按时间顺序的编号；需要将 KMCHK_HEAD_TACKDEPTH 配置为非 0（默认打开）才会记录内存操作函数调用链信息。关于配置项的详细说明参见第 5 章。

7.2.1.2.5. show prof 显示内存统计信息

功能:

show prof 命令用于显示应用程序统计相关的信息。

命令格式:

```
show prof [-t tid] [-snap number] [-F] [-program program] [-log log_file]
[-libpath path] [-offset addr]
```

参数说明:

- -t tid: 只显示线程号 tid 失败情况, 其中 tid 可以指定多个, 以逗号间隔
- -snap number: 对于定时自动获取的内存信息, 该参数用于指定快照编号, 其中 number 为日志编号, 如果不指定, 就显示第一份日志的信息
- -F: 显示从文件 mmap/munmap 的统计信息
- -program program: 指定应用程序可执行文件并覆盖以前通过该参数或者 set program 指定的文件, 功能与 set program 相同
- -log log_file: 指定 kmemchk 生成的日志文件并覆盖以前通过该参数或者 set log 指定的文件, 功能与 set log 相同
- -libpath path: 指定动态库路径并覆盖以前通过该参数或者 set libpath 指定的路径, 功能与 set libpath 相同
- -offset addr: 指定 armnmmu 代码段在内存中的偏移地址并覆盖以前通过该参数或者 set offset 指定的偏移地址, 功能与 set offset 相同

举例说明:


```
(kmem) show prof
##### Process prof log #####
malloc      214      calls count      failed count      total size(byte)
realloc     200      0              0              57682193
free        21      0              0              15150
memalign    110      0              0              115343365
mmap         5      0              0              57676730
munmap       2      0              0              33566848
unfree mem size 32548980
unfree mem peak 43014540
NOTES
The failed count item include the following conditions:
allocate memory failed,allocate 0 byte memory and free a NULL pointer.
```

图 7-16 show prof 不带参数时显示的信息

显示内容说明:

show prof 显示的信息包括进程统计信息和线程统计信息，其格式是相同的，只不过进程统计信息针对的是整个进程，线程针对单个线程。

- 第 1 列，显示函数：malloc、realloc、free、memalign、mmap、munmap
- 第 2 列，calls count，每个函数的调用次数；
- 第 3 列，failed count，内存操作返回失败（包括申请或释放内存失败、申请 0 字节以及释放空指针的情况）的次数；
- 第 4 列，每个函数操作的动态内存总大小，total size，单位 byte
- unfree mem size，应用程序当前使用动态内存的值，单位 byte
- unfree mem peak，应用程序运行过程中使用的动态内存最大值，单位 byte

 提示：其中 failed count 包含以下情况：内存分配或释放失败，分配 0 字节内存和释放空指针。

7.2.1.2.6. show block 显示内存块跟踪信息

功能：

show block 用于显示内存操作的跟踪信息，该命令在关闭记录详细内存操作日志功能时（即 KMCHK_OUTPUT_LOG 设置为 0）与 show unfree 功能一样，如果将 KMCHK_OUTPUT_LOG 设置为 1，则该命令还能查看已经释放的内存块的信息以及释放内存的操作的信息。

命令格式：

```
show block [-t tid] [-s low;high] [-a low;high] [-d low;high] [-f funcname] [-o (size/time/addr)] [-p] [-l logfile] [-n low;high] [-snap number] [-program program] [-log log_file] [-libpath path] [-offset addr]
```

参数说明：

- -t tid：只显示线程号 tid 失败情况，其中 tid 可以指定多个，以逗号间隔
- -s low;high：只显示 low 到 high 大小区间的内存未释放情况
- -a low;high：只显示 low 到 high 地址区间的内存未释放情况
- -d low;high：只显示 low 到 high 时间区间的内存未释放情况
 - ✧ 时间的格式为：年-月-日-时:分:秒;年-月-日-时:分:秒
- -f funcname：只显示 funcname 参数函数的内存未释放情况
- -o (size/time/addr)：以大小或时间或地址方式从小到大排序方式显示未分配内存情况

- -p: 逐条显示结果, 没有该参数则按函数调用链进行统计显示
- -l logfile: 将解析信息输出到指定的日志文件
- -n low;high: 只显示 low 到 high 序列号区间的未释放内存情况
- -snap number: 对于定时自动获取的内存信息, 该参数用于指定快照编号, 其中 number 为日志编号, 如果不指定, 就显示第一份日志的信息
- -program program: 指定应用程序可执行文件并覆盖以前通过该参数或者 set program 指定的文件, 功能与 set program 相同
- -log log_file: 指定 kmemchk 生成的日志文件并覆盖以前通过该参数或者 set log 指定的文件, 功能与 set log 相同
- -libpath path: 指定动态库路径并覆盖以前通过该参数或者 set libpath 指定的路径, 功能与 set libpath 相同
- -offset addr: 指定 armnmmu 代码段在内存中的偏移地址并覆盖以前通过该参数或者 set offset 指定的偏移地址, 功能与 set offset 相同

 提示: 参数中 low 与 high 之间, 需要使用','分号作为分隔符

举例说明:

```
(kmem) show block -a 0x8f143c8;0x8f143c8 -p
tid : 24684
time : 2010-08-12-10:05:42.288526
memory operation : malloc
size : 136
start_addr : 0x8f143c8
call trace :
0 : 0x804c3f6 <_kmchk_malloc_hook_1 /home/tangyk/kmemcheck/srcs/kmchk/kmemchk.c:3130 >
1 : 0x806b7be <__libc_malloc >
2 : 0x8070faa <allocate_dtv kmemchk.c:0 >
3 : 0x807126c <_dl_allocate_tls >
4 : 0x80541fa <__pthread_create_2_1 >
5 : 0x8048301 <main /home/tangyk/kmemcheck/target/x86/glibc/last_test.c:60 >
```

图 7-17 显示内存块信息

显示信息内容说明:

- tid: 线程 id 号
- time: 内存操作时间
- memory operation: 内存操作函数

- size: 内存请求大小
- start_addr: 内存起始地址
- call trace: 内存操作的函数调用链信息

以上信息中，需要打开配置项 `KMCHK_HEAD_TID`（默认打开）才能正确显示线程号，否则 `tid` 显示为 0；需要将配置项 `KMCHK_HEAD_TIME` 设置为 2（默认为 1）才能显示时间，否则显示默认显示内存操作的按时间顺序的编号；需要将 `KMCHK_HEAD_TACKDEPTH` 配置为非 0（默认打开）才会记录内存操作函数调用链信息。关于配置项的详细说明参见第 5 章。

7.2.1.2.7. show same 比较两份日志的相同部分

功能:

`show same` 用于比较程序运行过程中，不同时刻手动发送信号得到的日志文件或者不同时刻自动定时获取的内存信息的相同部分，并将比较结果保存在指定的文件中，如果没有指定保存结果的文件，则保存到临时文件“`same_result.txt`”中，通过该命令可以得到那些在程序运行过程中一直未释放的内存信息，帮助判断是否发生内存泄露。

命令格式:

```
show same file1 file2 [-snap number1 number2] [-f funcname] [-p] [-l logfile]
```

参数说明:

- file1 file2: 为同一个进程在运行过程中不同时刻分别发送信号获取的快照日志文件，两个文件的输入顺序随意
- -snap number1 number2: 对于定时自动获取的内存信息，该参数用于指定快照编号，其中 `number1`, `number2` 为快照编号
- -f funcname: 只显示函数调用链中包含 `funcname` 参数指定的函数的内存信息
- -l logfile: 将解析信息输出到指定的日志文件，不带该参数则将比较结果打印输出到终端
- -p: 逐条显示结果，没有该参数则按函数调用链进行统计显示

举例说明:

```
(kmem) show same 1.log 2.log -l same.txt
Prof info of memory blocks is as follows:

Level    size(byte)      same info num
0         0      - 99      82
1        100     - 999       3
2       1000    - 9999      40
3      10000   - 99999       0
4       Large
All compare info has been saved to log file : same.txt
(kmem)
```

图 7-18 比较文件，得到两个日志文件的相同部分

7.2.1.2.8. show diff 比较两份日志的不同部分

功能:

show diff 用于比较程序运行过程中，不同时刻手动发送信号得到的日志文件或者不同时刻自动定时获取的内存信息的不同部分，并将比较结果保存在指定的文件中，如果没有指定保存结果的文件，则保存到临时文件“differ_result.txt”中，通过该命令可以得到那些在程序运行过程中某一段时间内或者程序在执行某个操作过程中分配的并且还未释放的内存信息，帮助判断是否发生内存泄露。

命令格式:

```
show diff file1 file2 [-snap number1 number2] [-f funcname] [-p] [-l logfile]
```

参数说明:

- file1 file2: 为同一个进程在运行过程中不同时刻分别发送信号获取的快照日志文件，两个文件的输入顺序随意
- -snap number1 number2: 对于定时自动获取的内存信息，该参数用于指定快照编号，其中 number1, number2 为快照编号
- -f funcname: 只显示函数调用链中包含 funcname 参数指定的函数的内存信息
- -l logfile: 将解析信息输出到指定的日志文件
- -p: 逐条显示结果，没有该参数则按函数调用链进行统计显示

举例说明:

```
(kmem) show diff 1.log 2.log -l diff.txt
Prof info of memory blocks is as follows:

Level   size(byte)      differ info num
0        0      - 99      92
1       100     - 999       2
2      1000    - 9999      54
3     10000   - 99999       0
4      Large                2
All compare info has been saved to log file : diff.txt
(kmem)
```

图 7-19 比较文件，得到两个日志文件的不同部分

➡ 提示：以上各 show 命令的过滤参数中，对于 -t, -s, -a, -f, -n, -d，如果加上前缀 i 则表示非的过滤条件，比如 -t 760，表示只获取线程号为 760 的信息，而加上前缀 i，即 -it 760，则表示获取除线程号为 760 以外的其它线程的信息。

➡ 提示：对于排序选项“-o”，默认按升序排列，如果加上前缀“i”，即“-io”，则按降序排列。

7.2.1.2.9. show snap 显示自动定时快照相关信息

功能：

show snap 命令用于查看自动定时快照相关统计信息，包括快照列表以及程序内存消耗变化走势图。

其中如何启用定时自动获取内存快照功能请参见 7.1.2 节。

命令格式：

```
show snap [-g] [-t tid] [-s low;high] [-program program] [-log log_file] [-libpath path] [-offset addr]
```

参数说明：

- -g: 显示程序内存消耗变化走势图
- -t: 在输入-g 的情况下，显示程序指定线程内存消耗变化走势图
- -s: 显示内存即时消耗在指定范围内的快照信息
- -program program: 指定应用程序可执行文件并覆盖以前通过该参数或者 set program 指定的文件，

功能与 set program 相同

- -log log_file: 指定 kmemchk 生成的日志文件并覆盖以前通过该参数或者 set log 指定的文件，功能与 set log 相同，对于 show snap 命令应该指定文件名为“程序名_进程号
- _kmchk_leakautosnap.log”的日志文件，其中才是保存的定时自动获取的日志信息
- -libpath path: 指定动态库路径并覆盖以前通过该参数或者 set libpath 指定的路径，功能与 set libpath 相同
- -offset addr: 指定 armnommu 代码段在内存中的偏移地址并覆盖以前通过该参数或者 set offset 指定的偏移地址，功能与 set offset 相同

举例说明：

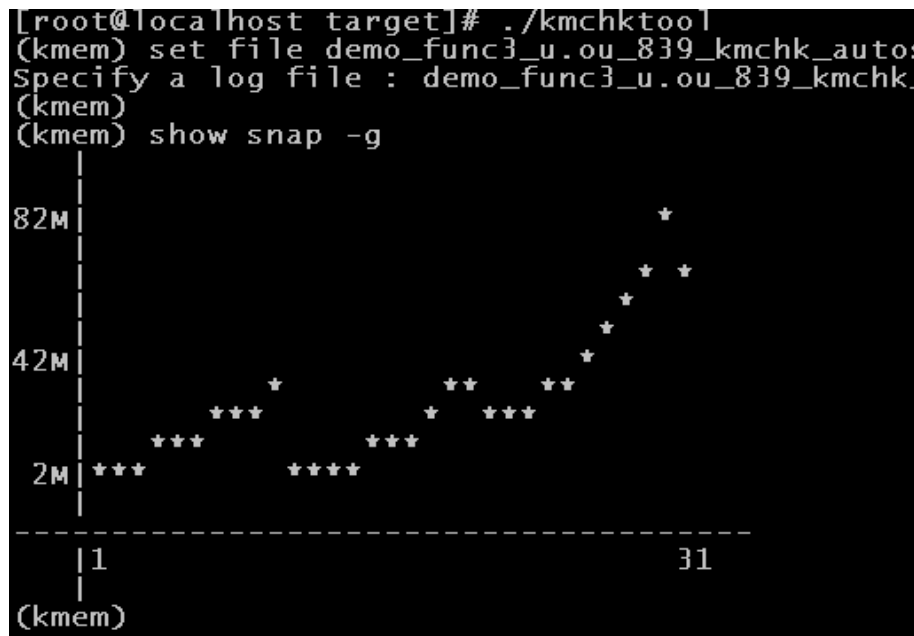


图 7-20 内存消耗变化走势图

```
(kmem) show snap
```

1	3233468	2011-02-17-15:37:26.0
2	6379196	2011-02-17-15:37:27.0
3	9524924	2011-02-17-15:37:28.0
4	12670652	2011-02-17-15:37:29.0
5	15816380	2011-02-17-15:37:30.0
6	18962108	2011-02-17-15:37:31.0
7	22107836	2011-02-17-15:37:32.0
8	25253564	2011-02-17-15:37:33.0
9	28399292	2011-02-17-15:37:34.0
10	31545020	2011-02-17-15:37:35.0
11	2184892	2011-02-17-15:37:36.0
12	4282044	2011-02-17-15:37:37.0
13	7427772	2011-02-17-15:37:38.0
14	10573500	2011-02-17-15:37:39.0
15	12670652	2011-02-17-15:37:40.0
16	14767804	2011-02-17-15:37:41.0
17	16864956	2011-02-17-15:37:42.0
18	23156412	2011-02-17-15:37:43.0
19	31545020	2011-02-17-15:37:44.0
20	36787900	2011-02-17-15:37:45.0
21	21059260	2011-02-17-15:37:46.0
22	24204988	2011-02-17-15:37:47.0
23	27350716	2011-02-17-15:37:48.0
24	32593596	2011-02-17-15:37:49.0
25	38885052	2011-02-17-15:37:50.0
26	46225084	2011-02-17-15:37:51.0
27	54613692	2011-02-17-15:37:52.0
28	64050876	2011-02-17-15:37:53.0
29	74536636	2011-02-17-15:37:54.0
30	86070972	2011-02-17-15:37:55.0
31	75535916	2011-02-17-15:37:55.0

图 7-21 内存信息列表

7.2.1.2.10. show thread 显示日志文件中记录的线程列表

功能:

show thread 用于显示日志文件中记录的线程列表。

命令格式:

```
show thread [-snap number] [-log log_file]
```

参数说明:

- -snap number: 对于定时自动获取的内存信息, 该参数用于指定快照编号, 其中 number 为日志编号, 如果不指定, 就显示第一份日志的信息
- -log log_file: -log log_file: 指定 kmemchk 生成的日志文件并覆盖以前通过该参数或者 set log 指定的文件, 功能与 set log 相同

举例说明：

```
(kmem) show thread
##### Thread list #####
Number   tid      name
1         6747    demo_malloc
2         6748    demo_malloc
3         6749    demo_malloc
(kmem)
```

图 7-22 显示线程列表

7.2.1.2.11. show current 显示当前分析的日志和应用程序

功能：

show cuurrent 用于显示当前分析的日志和应用程序。

命令格式：

```
show current
```

参数说明：

无

举例说明：

```
(kmem) show current
Program: demo_malloc
Log: demo_malloc_6780_kmchk.log
(kmem)
```

图 7-23 显示当前分析的日志和应用程序

7.2.1.3. rtconfig 命令

功能：

rtconfig 命令是 **kmchktool** 工具与应用程序建立通信连接的命令，建立连接后，才能使用例如 config 命令等需要通信的命令。

如果应用程序启动打开了配置项 **KMCHK_DEMO_THREAD**，则启动应用后，kmemchk 会创建一个线程在 9000 端口监听连接请求，如下

```
# ./target/demo_malloc &  
# waiting for connect at port [ 9000 ]
```

图 7-24 启动应用程序，建立监听

然后 kmchktool 就可以通过 rtconfig 命令与应用建立连接。

命令格式：

```
rtconfig (ip) (port)
```

参数说明：

- ip 是目标机 ip
- port 是端口号，默认是 9000

举例说明：

```
(kmem) rtconfig 127.0.0.1 9000
```

图 7-25 使用 rtconfig 命令进行连接

7.2.1.4. disconnect 命令

功能：

关闭 kmchktool 与应用程序的连接。

命令格式：

```
disconnect
```

7.2.1.5. config 命令

功能：

动态配置项设置内存泄露检测功能，需要使用 socket 与应用程序通信，链接后才能使用，详见 rtconfig 命令的使用。

命令格式：

```
config 配置项名 on|off
```

其中可以配置的项目参见 5.1.1 和 5.1.2 中列举的允许动态配置的配置项，例如：

```
config KMCHK_MEM_HEAD on
```

目前提供的 config 项如下所示：

```
KMCHK_ALL
KMCHK_PROF
KMCHK_THREAD
KMCHK_RANDOM_FAIL
KMCHK_MEM_HEAD
KMCHK_HEAD_TID
KMCHK_HEAD_TIME
KMCHK_HEAD_STACKDEPTH
KMCHK_SLOPOVER
```

7.2.1.6. help 命令

功能：

help 命令是用于快速熟悉 kmchktool 工具提供给用户的相关命令。

命令格式：

```
help
```

或者

```
help command
```

在 kmem 命令提示符中输入 **help** 即可看到如下的提示：

```
(kmem) help
disconnect          Disconnect with the manager thread of kmemchk.
rtconfig ip port    Connect to the manager thread of kmemchk to modify
                    the configuration of kmemchk.
q                   Exit the program.
help [command]      Help command.
config configure_command parameter config the target configuration
show (all/unfree/slop/failed/prof/block/diff/common/snap) parameter Show the memory
set (program/file/offset/libpath) path |offset Set the target file or offset addr
(kmem)
```

图 7-26 help 命令帮助信息

输入 **help command** 可以看到相应命令的详细帮助信息：


```
(kmem) help q
NAME
    q -- Exit the program

USAGE
    q

DESCRIPTION
    Exit the program.
```

图 7-27 单个命令的 help 信息

7.2.1.7. q 退出命令

功能:

退出 kmchktool 工具。

命令格式:

```
q
```

举例说明:

```
(kmem) q
Exit the Program.
[root@localhost kmemcheck]#
```

图 7-28 退出 kmchktool

7.2.1.8. kmchktool 快速解析命令

kmchktool 的使用方式有两种，一种是启动后进入交互界面，然后可输入命令并执行；另一种方式是在启动参数中指定要执行的 show 系列查看命令以及相关的程序文件和日志文件，kmchktool 执行完命令后就退出，不再进入交互界面，该功能可在程序的长时间运行过程中实现自动检测，通过脚本指定检查命令后，无需人工干预。支持的启动参数如下：

- -program program: 指定应用程序可执行文件，功能与 set program 相同
- -log log_file: 指定 kmemchk 生成的日志文件，功能与 set log 相同
- -libpath path: 指定动态库路径，功能与 set libpath 相同

- -offset addr: 指定 armnommu 代码段在内存中的偏移地址，功能与 set offset 相同
- show cmd args: cmd 和 args 分别为需要执行的命令及其参数，cmd 可以为 7.2.1.2 中介绍的任何
一个命令

举例说明：

```
[root@localhost opt]# ./kmchktool -program test_leak
k -log test_leak_23456_kmchk.log show unfree -l unfree.txt
Specify a program : test_leak
Specify a log file : test_leak_23456_kmchk.log
There are unfree memory blocks.

Prof info of memory blocks is as follows:

Level   size(byte)      unfree info num
0        0 - 99          9
1       100 - 999        2
2      1000 - 9999        0
3     10000 - 99999       0
4      Large             1
All info of unfreed memory blocks have been saved to file unfree.txt
```

图 7-29 kmchktool 通过启动参数执行命令的输出信息

7.2.2. kmchk_slop 命令

7.2.2.1. set 命令

■ set program 设置符号表文件

功能：

set program 命令用于设置目标应用程序可执行文件的本地路径。由于在进行堆栈信息解析的时候，需要指定程序的符号表进行符号分析，如果没有指定对应的目标程序路径，那么堆栈回溯信息中就只有地址没有符号。另外，如果应用程序被 **strip** 掉了行号信息，那么堆栈回溯定位行号也将不能够正确使用。

命令格式：

```
set program programname
```

■ set libpath 设置库路径

功能：

对于动态链接的应用程序，如果分析日志的环境与程序的运行环境并非同一个环境，则还需要指定库路径，以便客户端分析工具能找到库并加载其符号表。

命令格式：

```
set libpath path
```

■ set log 设置日志文件路径

功能：

set log 命令用来指定目标应用程序生成的跟踪日志 `kmchk_$(pid)_$(prog).log` 的文件名称和路径，`kmchktool` 将根据这个日志文件进行内存信息统计与分析。命令格式如下：

命令格式：

```
set log filename
```

■ set offset 设置地址偏移

功能：

在 ARM NOMMU 上，由于 ELF 符号表文件的地址和实际 FLT 格式在内存中的地址有一定偏移，造成符号解析工具不能正确地解析符号，这里需要有人工设置应用程序代码段 **TEXT** 在内存中的偏移地址。该偏移地址在运行 ARM NOMMU 程序的时候，由操作系统打印在终端。用户需要记录该地址，并在分析日志时输入地址。

命令格式：

```
set offset addr
```

7.2.2.2. save 命令

功能：

将所有违法信息解析（插入符号信息）后保存到文件中。

命令格式：

```
save [-n filepath]
```

参数说明：

- **n**：指定保存信息的文件名（可以含路径，如无路径则保存在后端工具的目录下）

举例说明：

```
save
save -n self_name
```

备注:

如果没有指定-n 参数，只使用“save”命令，则信息保存在后端工具的目录下的默认文件中。文件名=“可执行程序名+_all_info.txt”。

7.2.2.3. getsy 命令

功能:

根据输入的地址查找对应的符号，方便用户手动解析堆栈回溯等信息。

命令格式:

```
getsy addr(hex)
```

举例说明:

```
getsy 0xff080920
```

7.2.2.4. filter 命令

功能:

根据指定内容过滤违法信息。

命令格式:

```
filter [-R] [-n filename] [-a addr(hex)] [-t num(hex)] [-f func] [-i tid] [-c file]
```

参数说明:

- -R: 从上一次过滤结果过滤，不设置则从原始信息过滤;
- -n: 将过滤结果保存到文件中。后面可以跟文件名，如不指定文件名则保存到后端工具所在目录下的默认文件中，文件名=“可执行程序名+_now_info.txt”;
- -a: 过滤违法操作的内存，若操作范围包含过滤地址则命中;
- -t: 过滤违法类型;
- -f: 按照函数名过滤违法信息， 违法操作涉及此函数就命中;
- -i: 过滤违法信息的线程号;
- -c: 按照源文件过滤，违法操作涉及此文件就命中（此过滤项必须有调试信息，否则过滤失败）。

备注:

- 1) 过滤完成后会显示过滤结果的条数，如果数目在 5 条以下自动显示违法信息，否则不显示；
- 2) 在参数前加！可以实现非过滤。例如 filter -!t 1249，表示过滤掉 tid 是 1249 的违法信息；
- 3) 一条语句支持多种过滤条件，各条件间为与关系。例如：filter -i 129 -!t 0x1；则过滤结果是 tid 为 129，并且违法类型不是 1 的违法信息。
- 4) 违法类型详见 8.2 节的表 8-1。

举例说明：

```
filter -R -n tem_file -f main -t 0x1
```

7.2.2.5. help 命令

功能：

用于快速熟悉 kmchktool 工具提供给用户的相关命令。

命令格式：

```
help command
```

参数说明：

- command：查看相应命令的详细帮助信息

7.2.2.6. version 命令

功能：

显示 kmchktool 版本。

命令格式：

```
version
```

7.2.2.7. switchp 命令

功能：

堆栈回溯定位的源文件路径格式有两种：文件名或全路径+文件名，默认为前者。通过该命令可以进行切换。

命令格式：

```
switchp
```

7.2.2.8. switcho 命令

功能:

输出越界信息的顺序有两种: 按越界严重程度排序或按越界发生时间排序, 默认为前者。通过该命令可以进行切换。

命令格式:

```
switcho
```

7.2.2.9. q 退出命令

功能:

退出 kmchktool。

命令格式:

```
q
```

8. 检测信息详解

内存检测工具在使用时, 主要是以输出信息的方式来让用户了解应用程序的内存使用情况。这些信息提供了应用程序内存使用上的详细记录, 用户通过查看这些信息, 来定位问题。内存检测工具输出的信息包括 kmemchk 打印输出的信息和通过日志解析工具 **kmchktool** 输出的信息。下文通过对这些信息的详细描述, 来检测内存是否有覆盖、泄露等情况的发生, 以及发生这些情况后, 如何定位问题。

8.1. kmchk_leak 信息详解

8.1.1. 打印输出的信息

8.1.1.1. 内存统计信息

内存的使用情况统计信息, 可以查看的时间点如下:

- 1) 程序运行结束, 退出时, 将打印统计信息;

2) 程序运行过程中, 通过发送 SIGURG 信号给进程时, 将打印统计信息。

内存统计信息需要配置项 **KMCHK_PROF** 支持, 在配置了 **KMCHK_PROF** 的情况下, 统计信息的内容如图 8-1 所示:

```
##### Program memory prof info #####
calls count    failed count    total size(byte)
malloc          214             0          57682193
realloc         200             0          15150
free            21             0         115343365
memalign        110             0         57676730
mmap             5             0         33566848
munmap          2             0          1048576
unfree mem size 32548980
unfree mem peak 43014540
```

图 8-1 打印在终端的统计信息

内容说明:

打印在终端的统计信息如图 8-1 范例所示, 这些统计信息是应用进程对内存操作的一个粗略的统计信息。这个统计信息中的内容分为二个部分:

第一部分, 对内存操作函数的统计, 显示内容如下:

- 第 1 列, 显示函数: malloc、realloc、free、memalign、mmap、munmap
- 第 2 列, calls count, 每个函数的调用次数
- 第 3 列, failed count, 调用返回失败的次数
- 第 4 列, total size, 每个函数操作的动态内存总大小, 单位 byte

第二部分, 是对应用进程的统计信息, 包括如下内容:

- unfree mem size, 应用程序当前使用动态内存的值, 单位 byte
- unfree mem peak, 应用程序运行过程中使用的动态内存最大值, 单位 byte

8.1.1.2. 内存非实时越界检测信息

内存泄露检测功能附带的非实时越界检测功能, 用于一些由于对性能要求较高而不方便使用实时越界检测功能的场景。当检测到内存越界时, 内存检测工具将打印出发生越界的内存块的详细信息, 暂停线程的运行, 并将此信息写入日志中。

举例说明:

```
##### slop over occurs at tail of a memory block whose  
tid : 17862  
time : 1279009593.445945  
memory operation : malloc  
result : memory slop over at tail  
stack_depth : 6  
size : 10  
start_addr : 0x9e60d30  
slop over addr : 0x9e60d3a  
call trace :  
0 : 0x804aff8  
1 : 0x80690ff  
2 : 0x804823a  
3 : 0x80484bd  
4 : 0x8050dfa  
5 : 0x806d8ae
```

图 8-2 非实时越界检测输出信息

显示内容包括：

- tid: 线程号
- time: 线程执行时间
- memory operation: 内存操作函数
- result: 内存操作结果，指内存块是头部还是尾部被改写
- stack_depth: 堆栈层数
- size: 内存大小
- start_addr: 内存起始地址
- slop over addr: 内存被改写的地址
- call trace: 被改写的内存块申请时的函数调用链

8.1.1.3. 未释放内存信息

在应用程序收到信号记录未释放内存信息到文件时，以及程序退出时，如果有申请的内存没有释放，则打印输出对未释放的内存块的一个简单统计。

举例说明：

Prof info of unfreed memory blocks is as follows:

Level	size(byte)	Unfree Memory num
0	0 - 99	222
1	100 - 999	82
2	1000 - 9999	0
3	10000 - 99999	0
4	Large	3

图 8-3 程序退出时打印的内存泄露信息

通过这些统计信息，可以大致了解此时程序占用的动态内存情况。

8.1.2. 内存泄露检测统计/跟踪日志信息

在得到程序统计/日志后，可以使用第 7 章中介绍的工具命令，将二进制格式的日志解析成可读的分析日志文件。其中分析日志文件包含了如下内容：

- 文件头
- 进程统计信息
- 线程统计信息
- 内存块的信息

8.1.2.1. 文件头信息

日志头文件中，是进程的一些基本信息，提示信息包括：

- 用户程序大小端信息
- 用户程序支持的 CPU 信息
- 用户程序进程名
- 用户程序进程 pid 信息

```
#### Leak log header ####  
  
endian : little  
cpu_arch : i386  
program_name : test_leak  
pid : 23456
```

图 8-4 日志分析文件头

8.1.2.2. 进程统计信息

日志分析文件的第二部分是进程统计信息。其内容与 8.1.1 是一致的。

```
(kmem) show prof
##### Process prof log #####
calls count      failed count      total size(byte)
malloc           214              0                57682193
realloc          200              0                15150
free             21              0               115343365
memalign         110              0               57676730
mmap              5                0               33566848
munmap           2                0               1048576
unfree mem size  32548980
unfree mem peak  43014540
NOTES
The failed count item include the following conditions:
allocate memory failed,allocate 0 byte memory and free a NULL pointer.
```

图 8-5 日志分析文件的进程统计信息

8.1.2.3. 线程统计信息

线程统计信息是对线程使用内存的信息进行统计，例如对于 malloc 函数，某线程一共执行过多少次，该线程共分配了多少空间，等等，线程统计信息实例如下图所示：

```
thread id        3710
calls count      failed count      total size(byte)
malloc           0                0                0
realloc          0                0                0
free            10                0               57671680
memalign         110              0               57676730
mmap              1                0               2097152
munmap           2                0               1048576
unfree mem size  1053626
unfree mem peak  11539386

thread id        3711
calls count      failed count      total size(byte)
malloc          100                0                5050
realloc         200                0               15150
free            0                0                0
memalign        0                0                0
mmap            0                0                0
munmap          0                0                0
unfree mem size  20200
unfree mem peak  20200
```

图 8-6 日志分析文件线程统计信息

其中包含的内容有：

- thread id 字段：线程 ID 号
- calls count 字段：各分配或释放函数调用次数统计
- failed count 字段：各分配或释放函数调用失败次数统计
- total size 字段：各分配或释放函数总共操作的内存大小统计
- unfree mem size：该线程当前占用的总内存大小
- unfree mem peak：该线程占用的最大内存时的大小

线程信息的显示需要将配置项 KMCHK_THREAD 设置为 1（默认为 0），如果为 0 则不按线程进行统计。


8.1.2.4. 内存操作信息

堆栈回溯信息中，包含了内存操作函数的调用关系，每一个堆栈回溯信息包括如下内容

- 线程号 tid
- 时间 time
- 内存操作类型
- 内存操作结果：分别有如下几种情况：
 - ✧ 内存增加 increase memory
 - ✧ 内存减少 decrease memory
 - ✧ 分配或释放失败：operation failed
 - ✧ 内存泄露：memory leak
 - ✧ 内存上溢：slop over at head
 - ✧ 内存下溢：slop over at tail
- 操作内存的大小
- 操作内存的起始地址
- 堆栈回溯信息。其中第 0 层为顶层堆栈。

```
tid : 17862
time : 2010-07-13-16:26:33.445945
memory operation : malloc
result : increase memory
size : 10
start_addr : 0x9e60d30
call trace :
0 : 0x804aff8 <_kmchk_malloc_hook_1 /home/ze
1 : 0x80690ff <__libc_malloc >
2 : 0x804823a <demo_task /home/zengyi/userca
3 : 0x80484bd <task_starter /home/zengyi/use
4 : 0x8050dfa <start_thread kmemchk.c:0 >
5 : 0x806d8ae <clone >
```

图 8-7 日志分析文件中的堆栈回溯信息

 提示：在编译时，如果使用了优化参数，例如-O1，-O2，-Os等，应用程序的堆栈回溯可能出现回溯信息不完整的情况。特别是在 ARM 架构应用的编译中，优化参数将导致函数不建立栈帧信息，给堆栈回溯带来一定的困难，建议 ARM 下使用-fno-omit-frame-pointer 选项保证应用程序建立栈帧。

8.2. kmchk_slop 信息详解

kmchk_slop 的日志文件主要记录了越界类型、越界内存地址、堆栈回溯信息等内存越界相关信息，便于用户进行越界定位和故障分析。其中 kmchk_slop 定义的内存越界违法类型如下：

表 8-1 违法类型列表

值	类型	意义
0x00	unknown	预留，将来备用
0x01	overflow read(check)	越界读内存
0x02	overflow write(check)	越界写内存
0x03	redefinition	重复申明变量
0x04	wrong free	释放内存违法（释放范围包含了其它内存块，释放内存与申明大小不一致）
0x05	watch : unknown	wactch 跟踪的内存操作
0x06	uninitialize read	动态内存未初始化就读

0x81	unknown read	找不到所读的内存信息（可能是其它没有添加 kmchk_slop 参数构建的模块申明的）
0x82	unknown write	找不到所写的内存信息（可能是其它没有添加 kmchk_slop 参数构建的模块申明的）
0x83	unknown free	找不到待释放的内存信息（可能是其它没有添加 kmchk_slop 参数构建的模块申明的）

8.2.1. 打印输出的信息

以下为一个实时越界信息打印输出的示例：

kmchk_slop violation 1	#检测信息编号
violation type: overflow read (check)	#违法类型（本例为读检测违法）
violation rang: 0x81035fc -0x810360b size: 16	#越界读的内存范围和大小
pc=0x8048325 location='test.c:9 (main)'	#越界操作的 PC、源文件名、行号
backtrace:	#堆栈回溯信息
0x804a3e2	
0x804b707	
0x8048325	
0x804a9e6	
0x80528b9	
0x8048131	

说明：location 信息是 _mf_check 函数的一个参数传入的。来源分两种情况：

- 封装的库函数，例如 memcpy，在调用 _mf_check 时，显式写入的“memcpy source”或“memcpy dest”字段，以作越界时提示信息；
- 编译时嵌入到应用中 _mf_check 的参数，编译时就生成的，包含源码信息，方便定位。如“test.c:6 (main)”。

8.2.2. 工具解析信息

使用 kmchktool 工具加载日志文件后，可以使用 save 命令（命令的使用详见 7.2.2.2 节）获得完整的检测信息，如下：

```
*****
```

```
Log type: kmchk_slop          #slop 日志标识
endian: little endian        #标识被测程序 CPU 大小端
pointer size: 32             #标识程序是 32 位还是 64 位
slop count: 4                #越界信息条数
overflow read                : 1    #一条溢出类型的读越界（越界类型详见表 8-1）
overflow write                : 2    #两条溢出类型的写越界
unknown read                  : 1    #一条未知内存类型的读越界
```

```
kmchk_slop num: 1
slop type: overflow write    #越界类型
slop range: 0x83e5ac0 -- 0x83e5aeb size: 44 #越界操作的内存范围和大小
variable name: array         #越界操作的对象名称
source info: main.c:39 (vt_2) #越界操作的源文件行信息
tid: 11800                   #线程号
backtrace info:              #越界操作的堆栈回溯信息
    0x8052a7b (vt_2           main.c:32)
    0x805004a (start_thread   pthread_create.c:297)
    0x8087afe (clone          clone.S:130)
-----definition info----- #被越界的内存定义信息
memory type: global or static variable #被越界内存类型
memory range: 0x83e5ac0 -- 0x83e5ae7 size: 40 #被越界内存地址和大小
variable name: array         #被越界内存对象名称
source info: main.c:11       #被越界内存定义的位置
tid: 11752
```

```
kmchk_slop num: 2
slop type: overflow write
slop range: 0x88ecbc8 -- 0x88ecbcb size: 4
variable name: unknown var
source info: main.c:57 (vt_82)
tid: 11802
backtrace info:
    0x8052bea (vt_82         main.c:48)
```

```
0x805004a (start_thread      pthread_create.c:297)
0x8087afe (clone              clone.S:130)
```

-----definition info-----

```
memory type: heap
memory range: 0x88ecba0 -- 0x88ecbc9 size: 42
source info: (malloc region)
tid: 11802
```

backtrace info:

```
0x8052b70 (vt_82              main.c:51)
0x805004a (start_thread      pthread_create.c:297)
0x8087afe (clone              clone.S:130)
```

```
kmchk_slop num: 3
slop type: overflow read
slop range: 0x83e5ac0 -- 0x83e5aeb size: 44
variable name: array
source info: main.c:24 (vt_1)
tid: 11799
```

backtrace info:

```
0x8052928 (vt_1              main.c:17)
0x805004a (start_thread      pthread_create.c:297)
0x8087afe (clone              clone.S:130)
```

-----definition info-----

```
memory type: global or static variable
memory range: 0x83e5ac0 -- 0x83e5ae7 size: 40
variable name: array
source info: main.c:11
tid: 11752
```

```
kmchk_slop num: 4
slop type: unknown read
```

```
slop range: 0x88ecb20 -- 0x88ecb23 size: 4
variable name: unknown var
source info: main.c:76 (vt_81)
tid: 11801
backtrace info:
    0x8052dec (vt_81                main.c:65)
    0x805004a (start_thread         pthread_create.c:297)
    0x8087afe (clone                 clone.S:130)
*****
```

9. CGSL 环境下使用 kmemchk

在[第6节运行使用](#)一节中已经阐述过，为使动态程序运行环境的 libc 等动态库与编译程序时的工具链保持一致，在嵌入式环境下，一般采用用工具链中的动态库直接覆盖运行环境对应文件的方法。但是对于运行在 CGSL 环境下的程序，CGSL 作为服务器，其库不能在重启后还原，不能随意覆盖替换，需要采用设定运行时库等方法。

为简化工具使用步骤，CGSL custom 工具链提供了 mk_kmchk_leak.sh 和 mk_kmchk_slop.sh 两个脚本，这两个脚本分别封装了在 CGSL 环境上使用 kmemchk leak 和 slop 的各种设置，具体使用方法如下：

step1：如果应用程序原本是采用 CGSL 交叉工具链构建，则需要先修改 makefile，转变成本地构建；如果应用程序原本是采用本地 gcc 构建，则跳过这步；

step2：将带有 kmemchk 功能的 CGSL custom 工具链放置到 CGSL 服务器上；

step3：使用 source 命令执行 custom 工具链中的对应封装脚本；

例如：

```
source ./toolchain_path/bin/kmchk_leak.sh
```

注意：不能将脚本 copy 到其它路径执行；其中 toolchain_path 为交叉工具链路径。

step4：按照程序原有的构建过程进行构建。

注意：构建过程依赖 step1 设置的环境变量，所以必须在执行 step1 的会话环境中进行，不能另外开启一个会话窗口构建。

附录

版权说明

成研所基于开源社区提供的 Linux 内核源代码提供适用于各产品线的嵌入式 Linux 产品，从总体而言，主要应该遵循 GPL 许可协议的条款。

GPL 是 GNU 通用公共许可证的简称，为大多数的 GNU 程序和超过半数的自由软件所采用，Linux 内核就是基于 GPL 协议而开源。GPL 许可协议对在 GPL 条款下发布的程序以及基于程序的衍生著作的复制与发布行为提出了保留著作权标识及无担保声明、附上完整、相对应的机器可判读源码等较为严格的要求。GPL 规定，如果将程序做为独立的、个别的著作加以发布，可以不要求提供源码。但如果作为基于源程序所生著作的一部分而发布，就要求提供源码。

为尽可能地保护公司在 Linux 方面的自有研发成果，特别是产品线的应用程序，同时顺从于 GPL 协议条款的规定，成研所对 Linux 的启动方式做了设计，产品线通过启动方式的选择，可以有效地规避由于顺从 GPL 条款所带来的风险。

目前，成研所的 Linux 产品提供了模块加载和静态链接两种启动方式，如果产品线要完全规避 GPL 开源的风险，可以选择模块加载方式。两种启动方式介绍如下：

模块加载方式：将上层应用独立地编译链接成标准 Linux 所支持的模块文件 (*.mod)。运行时，首先启动 Linux 内核，然后通过 insmod 命令将各模块按照彼此间的依赖关系插入到内核。该方式下上层应用以及 tldagent 等独立于内核，可以不用开放源码，顺从于 GPL 的条款。

静态链接方式：将所有的模块（包括应用）都编译到内核中，形成一个较大的内核映像文件。该方式下上层应用以及 tldagent 等均做为 Linux 内核的一部分发布，如果要顺从 GPL 的规定，则开放源码的风险较高。

另一方面，成研所推出的集成开发环境 KIDE 是基于 eclipse.Org 所提供的 Eclipse 软件而开发的，成研所所做的工作是在 Eclipse 的框架下，增加满足嵌入式软件开发、调试所需的插件模块。按照 eclipse.org 的规定，采用 Eclipse 软件应该由 CPL (Common Public License)协议的条款来提供许可。

在 CPL 协议中规定，如果编写了一个模块，添加到一个根据 CPL 得到许可的程序中，并将该模块的对象代码与程序的其他部分一起发布，由于该模块不是程序的继承产物，所以不必提供该模块的源代码。因此，成研所的 KIDE 产品符合 CPL 中该条款的规定，也不用提供源代码。