Linux 内核崩溃原因分析及错误跟踪技术

随着嵌入式 Linux 系统的广泛应用，对系统的可靠性提出了更高的要求，尤其是涉及到生命财产等重要领域，要求系统达到安全完整性等级 3 级以上[1]，故障率（每小时出现危险故障的可能性）为 10-7 以下，相当于系统的平均故障间隔时间（MTBF）至少要达到 1141 年以上，因此提高系统可靠性已成为一项艰巨的任务。对某公司在工业领域 14 878 个控制器系统的应用调查表明，从 2004 年初到 2007 年 9 月底，随着硬软件的不断改进，根据错误报告统计的故障率已降低到 2004 年的五分之一以下，但查找错误的时间却增加到原来的 3 倍以上。

这种解决问题所需时间呈上升的趋势固然有软件问题，但缺乏必要的手段以辅助解决问题才是主要的原因。通过对故障的统计跟踪发现，难以解决的软件错误和从发现到解决耗时较长的软件错误都集中在操作系统的核心部分，这其中又有很大比例集中在驱动程序部分[2]。因此，错误跟踪技术被看成是提高系统安全完整性等级的一个重要措施[1]，大多数现代操作系统均为发展提供了操作系统内核“崩溃转储”机制，即在软件系统宕机时，将内存内容保存到磁盘[3]，或者通过网络发送到故障服务器[3]，或者直接启动内核调试器[4]等，以供事后分析改进。

基于 Linux 操作系统内核的崩溃转储机制近年来有以下几种：

(1) LKCD（Linux Kernel Crash Dump）机制[3]；

(2) KDUMP（Linux Kernel Dump）机制[4]；

(3) KDB 机制[5]；

(4) KGDB 机制[6]。

综合上述几种机制可以发现，这四种机制之间有以下三个共同点：

(1) 适用于为运算资源丰富、存储空间充足的应用场合；

(2) 发生系统崩溃后恢复时间无严格要求；

(3) 主要针对较通用的硬件平台，如 X86 平台。

在嵌入式应用场合想要直接使用上列机制中的某一种，却遇到以下三个难点无法解决：

(1) 存储空间不足

嵌入式系统一般采用 Flash 作为存储器，而 Flash 容量有限，且可能远远小于嵌入式系统中的内存容量。因此将全部内存内容保存到 Flash 不可行。

(2) 记录时间要求尽量短

嵌入式系统一般有复位响应时间尽量短的要求，有的嵌入式操作系统复位重启时间不超过 2s，而上述几种可用于 Linux 系统的内核崩溃转储机制耗时均不可能在 30s 内。写 Flash 的操作也很耗时间，实验显示，写 2MB 数据到 Flash 耗时达到 400ms 之多。

(3) 要求能够支持特定的硬件平台

嵌入式系统的硬件多种多样，上面提到的四种机制均是针对 X86 平台提供了较好的支持，而对于其他体系的硬件支持均不成熟。

由于这些难点的存在，要将上述四种内核崩溃转储机制中的一种移植到特定的嵌入式应用平台是十分困难的。因此，针对上述嵌入式系统的三个特点，本文介绍一种基于特定平台的嵌入式 Linux 内核崩溃信息记录机制 LCRT(Linux Crash Record and Trace)，为定位嵌入式 Linux 系统中软件故障和解决软件故障提供辅助手段。

1 Linux 内核崩溃的分析

分析 Linux 内核对于运行期间各种“陷阱”的处理可以得知，Linux 内核对于应用程序导致的错误可以予以监控，在应用程序发生除零、内存访问越界、缓冲区溢出等错误时，Linux 内核的异常处理例程可以对这些由应用程序引起的异常情况予以处理。当应用程序产生不可恢复的错误时，Linux 内核可以仅仅终止产生错误的应用程序，其他应用程序仍然可以正常运行。

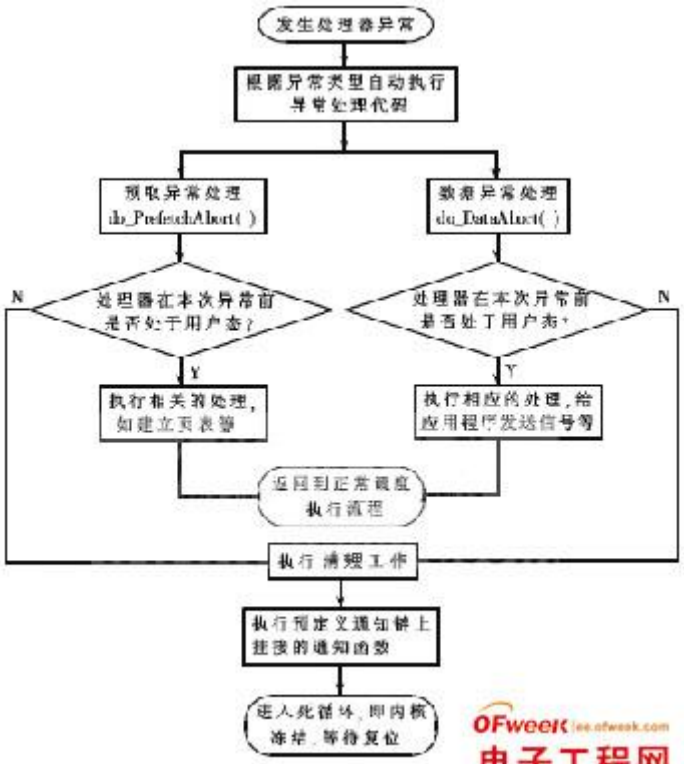

图 1 Linux 内核对异常的处理过程

如果 Linux 内核本身或者新开发的 Linux 内核模块存在 bug，产生了“除零”，“内存访问越界”、“缓冲区溢出”等错误，同样会由 Linux 内核的异常处理例程来处理。Linux 内核通过在异常处理程序中判断，如果发现是“严重的不可恢复”的内核异常，则会导致“内核恐慌”（kernel panic），即 Linux 内核崩溃。图 1 所示为 Linux 内核对异常情况的处理流程。

2 LCRT 机制的设计与实现

通过对 Linux 内核代码的分析可知，Linux 内核本身提供了一种“内核通知机制”[7-8]，并预定义了“内核事件通知链”，使得 Linux 内核扩展开发人员可以通过这些预定义的内核事件通知链在特定的内核事件发生时执行附加的处理流程。通过对 Linux 内核源代码的研究发现，对于上文中提到的“严重不可恢复的内核异常”，预定义了一个通知链和通知点，使得在发生 Linux 内核崩溃之后，可以在 Linux 内核的 panic 函数中预定义的一个“内核崩溃通知链”[7]上挂接 LCRT 机制来获得 Linux 内核崩溃现场的一些信息并记录到非易失性存储器中，以便分析引起 Linux 内核崩溃的原因。

2.1 设计要点

LCRT 机制的设计和实现基于如下特定的机制：

(1) 编译器选项与内核依赖

Linux 内核及相应的驱动程序都采用 GNU[9]的开源编译器 GCC[9]编译，为了结合 LCRT 机制方便地提取信息和记录信息，需要采用特定的 GCC 编译器选项来编译 Linux 内核和相关的驱动程序以及应用程序。用到的选项为：-mpoke-function-name[9]。使用这个选项编译出的二进制程序中可以包含 C 语言函数名称的信息，以方便函数调用链回溯时记录信息的可读性。

(2) Linux 内核 notify_chain 机制[8]

Linux 内核提供“通知链”功能，并预定义了一个内核崩溃通知链，在 Linux 内核的异常处理例程中判断出系统进入“不可恢复”状态时，会沿预定义的通知链顺序调用注册到相应链中的通知函数。

(3) 函数调用的栈布局

Linux 内核的绝大部分由 C 语言实现，而且 C 语言也多用来进行 Linux 内核开发。Linux 内核及使用 LKM 扩展而加入 Linux 内核执行环境的代码是有规律可循的，这些代码在执行过程中产生的栈布局和这些规律的代码相关联。例如，这些函数在执行函数之前会保存本函数调用后的返回地址、本函数被调用时传递过来的参数及调用本函数的函数所拥有的栈帧的栈底。

2.2 LCRT 机制的设计思想

LCRT 机制分为 Linux 内核模块[8]部分和 Linux 用户程序部分。内核模块部分的设计采用了 Linux 内核模块的模式而不是直接修改 Linux 内核。这样的设计降低了 Linux 内核和 LCRT 机制之间的耦合度，同时满足了 Linux 内核和 LCRT 机制独立升级完善的便利性。用户程序部分完成从非易失性存储器中读取、清除 LCRT 机制保存的信息等相关功能。

在 LCRT 机制的设计中，针对嵌入式系统的特点，其设计决策有：

(1) 将对于解决和定位问题最具辅助意义的函数调用关系链记录下来。

(2) 为了不占用过多的存储空间，有选择性地将函数调用序列上的函数各自用到的栈内容保存起来，而不是保存全部内容。

(3) 将记录的信息保存到非易失性存储器中，这样既达到了掉电保存的目的、又缩短了写入时间。

LCRT 机制的设计包括以下五个方面。

(1) 设计 Linux 内核模块、动态地加载 LCRT 机制、尽量少地修改 Linux 内核代码。

(2)在相应、预定义的 Linux 内核通知链上挂接 LCRT 的通知函数。

(3) 在 LCRT 机制的通知处理函数中进行堆栈回溯得到函数调用信息。

(4) 记录回溯到的函数调用信息和堆栈空间内容到非易失性存储器。

(5) 开发用户空间的工具，可以从非易失性存储器中读取保存的信息。

2.3 LCRT 机制的实现

LCRT 机制的实现可参照 2.2 节的设计思想，分步予以实现。限于篇幅，本文不过多涉及 Linux 内核模块的原理和实现相关的细节，仅仅给出 LCRT 机制的内核模块实现伪代码。用伪代码描述 LCRT 机制的加载函数如下:

```
int lcrt_init(void)

{

printk("Registering my__panic notifier. ");

bt_nvram_ptr=(volatile unsigned char*)ioremap_

nocache (BT_NVRAM_BASE,BT_NVRAM_LENGTH);

bt_nvram_index+=sizeof(struct bt_info);

*)bt_nvram_ptr,BT_NVRAM_LENGTH);

notifier_chain_register(&panic_notifier_list,&my_

panic_block);

return 0;

}
```

LCRT 机制的通知处理函数完成函数调用关系回溯、得到函数名称、函数栈内容等工作，限于篇幅，在这里用下面伪代码说明：

```
void ll_bt_information(struct pt_regs *pr)

{

变量定义等初始化工作

do {

reglist=*(unsigned long *)(*myfp-8);

//从函数栈帧的顶部获取函数开始执行时保存的寄存器信息
```

//从函数的代码区中取得函数的名称

//从函数的栈帧里取出函数执行函数体代码之前保存的函数参数信息

//从本函数的栈帧中得到调用本函数的代码所在位置和调用本函数的函数栈帧的栈底

}while(直到函数调用链的链头);

//取得函数调用栈帧的内容

//填充信息记录的记录头部

//将上面的循环中取得的信息保存到非易失性存储器中

write_to_nvram((void *)bt_nvram_ptr,&bt_record_header,sizeof(bt_info_t));

}

3 验证评估 LCRT 机制

3.1 部署 LCRT 机制

部署 LCRT 机制，使 LCRT 机制发挥作用前需要做的相关工作有：

(1)针对目标 Linux 内核编译 LCRT 机制的 Linux 内核模块部分；

(2) 将 LCRT 机制的内核模块部分载入 Linux 内核。

3.2 实验结果

为了实验 LCRT 机制的作用效果，构造一个会造成 Linux 内核崩溃的设备驱动模块，记这个内核驱动模块为 bugguy.ko,列出如下所示的 bugguy.ko 中会引起 Linux 内核崩溃的代码如下所示：

irqreturn_t my_timer_interrupt(int irq,void *dev_id,struct pt_regs* regs)

{

确认硬件状态并清除中断状态

if(ujiffies > 5000) {

void * ill_pointer=NULL;

```
*(unsigned long *)ill_pointer=0;
}
else {
ujiffies++;
}
return IRQ_HANDLED;
}
```

说明：用黑体标出的代码即为产生 bug 的代码

从上面的代码可以看出，这个错误是对空指针进行解析而造成的。在一个中断处理函数中如果发生对空指针的解析，将会引起 Linux 内核的崩溃。在部署完成 LCRT 机制的嵌入式 linux 系统上将这个 bugguy.ko 载入 Linux 内核，使得会引起 Linux 内核崩溃的中断处理程序得以运行，LCRT 机制可以将相关的信息保存到非易失性存储器中，在系统复位后，通过 LCRT 机制的用户空间工具，可以将保存的信息读取出来。实验结果显示，可以得到如图 2 所示的函数调用链信息。

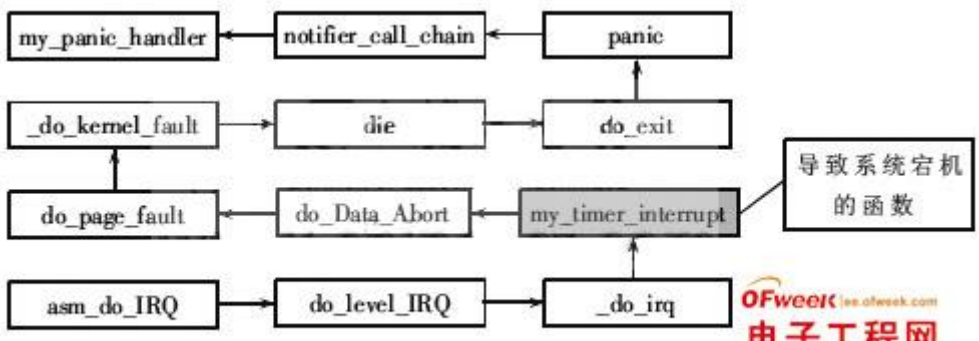

图 2 LCRT 机制记录下的函数调用链

图 2 标注即为会引起 Linux 内核崩溃的错误代码的中断处理函数即真正引起系统宕机的“罪魁祸首”。而记录下的所有信息仅仅占用了不到 1KB 的存储空间，写入非易失性存储器所耗用的时间控制在 50ms 以内。在使用少量空间和少量时间的情况下，所记录下的信息对于查找问题和解决问题都有较大的帮助。

实验结果表明，在 LCRT 机制的作用下，可以快速地定位到嵌入式 Linux 系统中隐藏的可能会导致系统宕机的软件缺陷。这就为后续的故障解决和软件完善提供了关键的辅助信息。对嵌入式 Linux 内核而言，即是为提高 Linux 内核的稳定性和可靠性提供了帮助。

在基于 ARM 的嵌入式 Linux 应用中，开发 LCRT 机制来记录系统内核发生崩溃时引起崩溃的函数调用链和栈信息到非易失性存储器中，截至目前为止，LCRT 机制可以记录基于 ARM 的嵌入式 Linux 内核发生崩溃时的函数调用链信息，可直接得到函数名称、函数调用链中单个函数被调用时的参数信息以及函数调用链中的函数各自的栈帧信息。这些记录下来的信息对于完善和发展基于 ARM 的嵌入式 Linux 应用具有重要的辅助意义。