# 概述

## coredump

当程序异常退出时，内核会在当前工作目录下生成一个core文件，其本质就是一个关于异常发生时刻内核的映像，同时加上调试信息。

通常情况下，core文件会包含了程序运行时的内存，寄存器状态，堆栈指针，内存管理信息还有各种函数调用堆栈信息等，我们可以理解为是程序工作当前状态存储生成第一个文件，许多的程序出错的时候都会产生一个core文件，通过工具分析这个文件，我们可以定位到程序异常退出的时候对应的堆栈调用等信息，找出问题所在并进行及时解决。

core文件在使用gdb进行调试的时候很有用，可以显示出导致程序出错的代码所在文件和行数。

获取内核转储（core dump）的最大好处是，它能保存问题发生时的状态，只要有问题发生时程序的可执行文件和内核转储，就可以知道进行当时的状态。

内核有问题时的现象：

区分方法 结果

Ps 显示中途停止，状态为D

ping 不返回响应

键盘 键盘无法输入

kill -9 无法结束程序

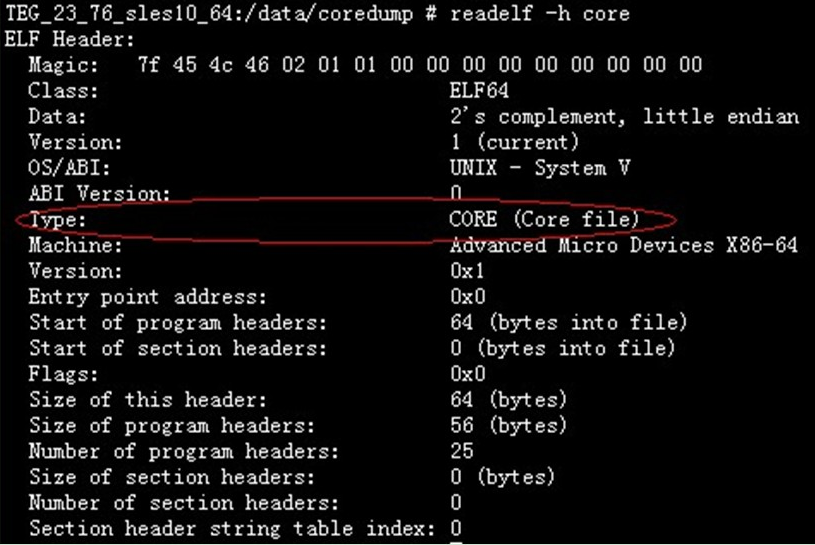
strace 无法附加attach到进程（无响应）

gdb 无法附加attach到进程（无响应）

显示内核信息 softlockup等有输出结果

注：这里添加调试信息需要在gcc编译的时候加上-g参数，否则gdb调试时是没有任何调试信息输出的。

## 格式判断

 在类unix系统下，coredump文件本身主要的格式也是ELF格式，因此，我们可以通过readelf命令进行判断。

可以看到ELF文件头的Type字段的类型是：CORE (Core file)

可以通过简单的file命令进行快速判断：

# 开关

在使用core文件前，需要查看该系统是否已经打开core文件的生成开关，具体指令为：

ulimit –c

该指令表示core文件大小的上限，如果输出结果为0，即core文件大小上限为0，则表示关闭了该功能，不会在指定目录下生成core文件。

打开core文件的方法：

unlimit –a

unlimit –c unlimited

说明：unlimit命令设置后只针对一个终端有效，所以另外启用一个终端时需要重新设置。

generate-core-file 会手动在当前文件生成core文件，不是在系统设置的目录下。

# 大小

## 查看开关

使用ulimit –c命令查看core文件的生成开关，若结果为0（即core文件大小上限为0），则表示关闭此功能，不会生成core文件。

## 修改大小

可以通过调整core文件大小来开启上限，具体指令：

unlimit –c filesize

该指令表示限制core文件的大小上限为filesize Kbyte，如果生成的信息超出此大小，core将会被裁剪，最终生成一个不完整的core文件，在调试此core文件的时候，gdb会提示错误。

如果不设置core文件大小的上限，可以这样设置：

unlimit –c unlimited

# 存储位置

## 位置

core文件默认的存储位置与对应的可执行程序在同一目录下，文件名是core，可以通过下面的命令看到core文件的存在位置：

cat /proc/sys/kernel/core\_pattern

缺省值是core

注意：这里是指在进程当前工作目录的下创建，通常与程序在相同的路径下。但如果程序中调用了chdir函数，则有可能改变了当前工作目录，这时core文件创建在chdir指定的路径下。有好多程序崩溃了，我们却找不到core文件放在什么位置，和chdir函数就有关系。当然程序崩溃了不一定都产生 core文件。

通过下面的命令可以更改coredump文件的存储位置，若你希望把core文件生成到/data/coredump/core目录下：

echo “/data/coredump/core”> /proc/sys/kernel/core\_pattern

## 名称

缺省情况下，内核在coredump时所产生的core文件放在与该程序相同的目录中，并且文件名固定为core。很显然，如果有多个程序产生core文件，或者同一个程序多次崩溃，就会重复覆盖同一个core文件，因此我们有必要对不同程序生成的core文件进行分别命名。

我们通过修改kernel的参数，可以指定内核所生成的coredump文件的文件名。例如，使用下面的命令使kernel生成名字为core.filename.pid格式的core dump文件：

echo “/data/coredump/core.%e.%p” >/proc/sys/kernel/core\_pattern

这样配置后，产生的core文件中将带有崩溃的程序名、以及它的进程ID。上面的%e和%p会被替换成程序文件名以及进程ID。

如果在上述文件名中包含目录分隔符“/”，那么所生成的core文件将会被放到指定的目录中。需要说明的是，在内核中还有一个与coredump相关的设置，就是/proc/sys/kernel/core\_uses\_pid。如果这个文件的内容被配置成1，那么即使core\_pattern中没有设置%p，最后生成的core dump文件名仍会加上进程ID。

# 配置

## /proc文件系统

/proc文件系统是一个伪文件系统，就是说他是一个实际不存在的目录，因为还是一个非常特殊的目录，它以文件系统的方式为访问系统内核数据的操作提供接口，它并不存在于某个磁盘上，而是由核心在内存中产生，这个目录用于提供关于系统的信息。/proc目录的内容为系统启动时自动生成的，可以通过修改/proc的文件微内核参数。

使用vi无法成功编辑/proc/sys/kernel/core\_patterm，只能使用echo命令修改或者命令sysctl修改（可以通过在/etc/sysctl.conf文件中，对sysctl变量kernel\_core\_patterm的设置）。

说明：core文件是运行时内存的信息，proc存储的正是运行的一些信息。

默认是在当前目录下生成core文件，但是一般我们需要设置一个固定的目录存放core文件，以便于后期调试。对应的配置文件为/etc/sysctl.conf。

目前，有三种配置方法，分别是：

## 临时修改

echo ‘/home/project/core/core.%e.%p.%t’ > /proc/sys/kernel/core\_pattern

这种方式只是临时测试使用，因为/proc目录本身是动态加载的，每次主机重启都会重新加载，生产环境不推荐使用这种方式。

## 永久修改

修改/etc/sysctl.conf（该配置文件对应/proc/sys目录及其子目录）文件中的配置项，对sysctl变量kernel.core\_pattern进行设置：

# Allow suid programs to dump core

# Dump core in the location

kenel.core\_pattern = /home/project/core/core.%e.%p.%t

kernel.core\_uses\_pid = 0

配置完后手动生效：

sysctl –p /etc/sysctl.conf

说明：

### kernel.core\_pattern

kernel.core\_pattern配置项用于设置core文件的格式，这里的%e表示可执行文件名称，%p表示被转储进程的PID，%t表示时间，这样就可以唯一确定core文件对应的具体目标进程了。

参数列表包括：

%p 添加PID

%u 添加当前UID

%g 添加当前GID

%s 添加导致产生core的信号

%t 添加core文件生成时的unit时间

%h 添加主机名

%e 添加命令名

举例：

echo “/corefile/core-%e-%p-%t” > /proc/sys/kernel/core\_patterm，可以将core文件统一生成到/corefile目录下，产生的文件命令为core-命令名-pid-时间戳。

### core\_uses\_pid

core\_uses\_pid可以控制core文件的文件名是否添加PID作为拓展，如果该配置项被配置成1，即使core\_pattern中没有设置%p，最后生成的core dump文件名仍会加上进程ID；如果配置为0，则表示生成的core文件统一命名为core。

举例：

echo “1”> “/proc/sys/kernel/core\_uses\_pid”

## sysctl命令配置生效

sysctl -w kenel.core\_pattern = " /home/project/core/core.%e.%p.%t "

综上所述，我们实际生产环境中推荐使用修改配置文件/etc/sysctl.conf的方式。

# 调试

在程序异常终止后生成core文件，这里包含着重要的调试信息，使用gdb可以发现这些异常，具体操作：

gdb 可执行文件 core文件

bt（使用该命令可以看到程序出错的地方）

说明：用gdb同时调试一个运行程序和core文件，core是程序执行后core dump产生的文件。当程序非法崩溃的时候会产生一个core文件，然后使用这个命令会直接定位到发生程序崩溃的地方。

注意有时候需要设置系统命令ulimit –t unlimited才能产生core文件。

# 原因总结

造成程序coredump的原因有很多，这里总结一些比较常用的经验：

1、内存访问越界

a) 由于使用错误的下标，导致数组访问越界。

b) 搜索字符串时，依靠字符串结束符来判断字符串是否结束，但是字符串没有正常的使用结束符。

c) 使用strcpy, strcat, sprintf, strcmp,strcasecmp等字符串操作函数，将目标字符串读/写爆。应该使用strncpy, strlcpy, strncat, strlcat, snprintf, strncmp, strncasecmp等函数防止读写越界。

2、多线程程序使用了线程不安全的函数。

应该使用下面这些可重入的函数，它们很容易被用错：

asctime\_r(3c) gethostbyname\_r(3n) getservbyname\_r(3n)ctermid\_r(3s) gethostent\_r(3n) getservbyport\_r(3n) ctime\_r(3c) getlogin\_r(3c)getservent\_r(3n) fgetgrent\_r(3c) getnetbyaddr\_r(3n) getspent\_r(3c)fgetpwent\_r(3c) getnetbyname\_r(3n) getspnam\_r(3c) fgetspent\_r(3c)getnetent\_r(3n) gmtime\_r(3c) gamma\_r(3m) getnetgrent\_r(3n) lgamma\_r(3m) getauclassent\_r(3)getprotobyname\_r(3n) localtime\_r(3c) getauclassnam\_r(3) etprotobynumber\_r(3n)nis\_sperror\_r(3n) getauevent\_r(3) getprotoent\_r(3n) rand\_r(3c) getauevnam\_r(3)getpwent\_r(3c) readdir\_r(3c) getauevnum\_r(3) getpwnam\_r(3c) strtok\_r(3c) getgrent\_r(3c)getpwuid\_r(3c) tmpnam\_r(3s) getgrgid\_r(3c) getrpcbyname\_r(3n) ttyname\_r(3c)getgrnam\_r(3c) getrpcbynumber\_r(3n) gethostbyaddr\_r(3n) getrpcent\_r(3n)

3、多线程读写的数据未加锁保护

对于会被多个线程同时访问的全局数据，应该注意加锁保护，否则很容易造成coredump

4、非法指针

a) 使用空指针

b) 随意使用指针转换。一个指向一段内存的指针，除非确定这段内存原先就分配为某种结构或类型，或者这种结构或类型的数组，否则不要将它转换为这种结构或类型的指针，而应该将这段内存拷贝到一个这种结构或类型中，再访问这个结构或类型。这是因为如果这段内存的开始地址不是按照这种结构或类型对齐的，那么访问它时就很容易因为bus error而core dump。

5、堆栈溢出

不要使用大的局部变量（因为局部变量都分配在栈上），这样容易造成堆栈溢出，破坏系统的栈和堆结构，导致出现莫名其妙的错误。