linux下生成core dump文件方法及设置【转】 - sky-heaven - 博客园

sky-heaven 关注 - 15 粉丝 - 239

转自：http://blog.csdn.net/mrjy1475726263/article/details/44116289

源自：http://andyniu.iteye.com/blog/1965571

core dump的概念：

A **core dump** is the recorded state of the working memory of a computer program at a specific time, generally when the program has terminated abnormally (crashed). In practice, other key pieces of program state are usually dumped at the same time, including the processor registers, which may include the program counter and stack pointer, memory management information, and other processor and operating system flags and information. The name comes from the once-

1

standard memory technology core memory. Core dumps are often used to diagnose or debug errors in computer programs.

On many operating systems, a fatal error in a program automatically triggers a core dump, and by extension the phrase "to dump core" has come to mean, in many cases, any fatal error, regardless of whether a record of the program memory is created.

在linux平台下，设置core dump文件生成的方法：

linux coredump调试

1 ）如何生成 coredump 文件 ?

登陆 LINUX 服务器，任意位置键入

```
echo "ulimit -c 1024" >> /etc/profile
```

退出 LINUX 重新登陆 LINUX

键入 ulimit -c

2

如果显示 1024 那么说明 coredump 已经被开启。

1024 限制产生的 core 文件的大小不能超过 1024kb，可以使用参数 unlimited，取消该限制

ulimit -c unlimited

2 ）. core 文件的简单介绍

在一个程序崩溃时，它一般会在指定目录下生成一个 core 文件。 core 文件仅仅是一个内存映象 (同时加上调试信息)，主要是用来调试的。

3 ）. 开启或关闭 core 文件的生成

用以下命令来阻止系统生成 core 文件 :

ulimit -c 0

下面的命令可以检查生成 core 文件的选项是否打开 :

ulimit -a

3

该命令将显示所有的用户定制，其中选项 -a 代表“ all ”。

也可以修改系统文件来调整 core 选项

在 /etc/profile 通常会有这样一句话来禁止产生 core 文件，通常这种设置是合理的 :

No core files by default

ulimit -S -c 0 > /dev/null 2>&1

但是在开发过程中有时为了调试问题，还是需要在特定的用户环境下打开 core 文件产生的设置。

在用户的 ~/.bash_profile 里加上 ulimit -c unlimited 来让特定的用户可以产生 core 文件。

如果 ulimit -c 0 则也是禁止产生 core 文件，而 ulimit -c 1024 则限制产生的 core 文件的大小不能超过 1024kb

4 ）. 设置 Core Dump 的核心转储文件目录和命名规则

4

/proc/sys/kernel/core_uses_pid 可以控制产生的 core 文件的文件名中是否添加 pid 作为扩展，如果添加则文件内容为 1，否则为 0

proc/sys/kernel/core_pattern 可以设置格式化的 core 文件保存位置或文件名，比如原来文件内容是 core-%e

可以这样修改 :

echo "/corefile/core-%e-%p-%t" > core_pattern

将会控制所产生的 core 文件会存放到 /corefile 目录下，产生的文件名为 core- 命令名 -pid- 时间戳

以下是参数列表 :

%p - insert pid into filename 添加 pid

%u - insert current uid into filename 添加当前 uid

%g - insert current gid into filename 添加当前 gid

%s - insert signal that caused the coredump into the filename 添加导致产生

5

core 的信号

%t - insert UNIX time that the coredump occurred into filename 添加 core 文件生成时的 unix 时间

%h - insert hostname where the coredump happened into filename 添加主机名

%e - insert coredumping executable name into filename 添加命令名

6 ）. 一个小方法来测试产生 core 文件

直接输入指令 :

kill -s SIGSEGV $$

如何产生Core Dump

发生doredump一般都是在进程收到某个信号的时候，Linux上现在大概有60多个信号，可以使用 kill -l 命令全部列出来。

```
sagi@sagi-laptop:~$ kill -l
     1) SIGHUP   2) SIGINT   3) SIGQUIT  4) SIGILL   5) SIGTRAP
```


6

```
 6) SIGABRT     7) SIGBUS   8) SIGFPE   9) SIGKILL 10) SIGUSR1
11) SIGSEGV   12) SIGUSR2 13) SIGPIPE 14) SIGALRM 15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP
21) SIGTTIN   22) SIGTTOU 23) SIGURG  24) SIGXCPU 25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF 28) SIGWINCH    29) SIGIO   30) SIGPWR
31) SIGSYS    34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4    39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTM
43) SIGRTMIN+9    44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTM
48) SIGRTMIN+14   49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTM
53) SIGRTMAX-11   54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTM
58) SIGRTMAX-6    59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTM
63) SIGRTMAX-1    64) SIGRTMAX
```

针对特定的信号，应用程序可以写对应的信号处理函数。如果不指定，则采取默认的处理方式, 默认处理是coredump的信号如下：

```
3)SIGQUIT   4)SIGILL    6)SIGABRT   8)SIGFPE    11)SIGSEGV    7)SIGBUS    31)SIGSY
5)SIGTRAP   24)SIGXCPU  25)SIGXFSZ  29)SIGIOT
```

我们看到SIGSEGV在其中，一般数组越界或是访问空指针都会产生这个信号。另外虽然默认是这样的，但是你也可以写自己的信号处理函数改变默认行为，更多信号相关可以看参考链接3[3]。

上述内容只是产生coredump的必要条件，而非充分条件。要产生core文件还

7

依赖于程序运行的shell，可以通过ulimit -a命令查看，输出内容大致如下：

```
sagi@sagi-laptop:~$ ulimit -a
    core file size          (blocks, -c) 0
    data seg size           (kbytes, -d) unlimited
    scheduling priority             (-e) 20
    file size               (blocks, -f) unlimited
    pending signals                 (-i) 16382
    max locked memory       (kbytes, -l) 64
    max memory size         (kbytes, -m) unlimited
    open files                      (-n) 1024
    pipe size            (512 bytes, -p) 8
    POSIX message queues     (bytes, -q) 819200
    real-time priority              (-r) 0
    stack size              (kbytes, -s) 8192
    cpu time               (seconds, -t) unlimited
    max user processes              (-u) unlimited
    virtual memory          (kbytes, -v) unlimited
    file locks                      (-x) unlimited
```

看到第一行了吧，core file size，这个值用来限制产生的core文件大小，超过这个值就不会保存了。我这里输出是0，也就是不会保存core文件，即使产生了，也保存不下来==! 要改变这个设置，可以使用ulimit -c unlimited。

8

OK, 现在万事具备，只缺一个能产生Core的程序了，介个对C程序员来说太容易了。

1. #include <stdio.h>;
2. #include <stdlib.h>;
3.
4. int crash()
5. {
6. char <span style="margin:0px;padding:0px;

9

border:0px;color:rgb(51,153,51);vertical-align:baseline;background-color:transparent;">*xxx = "crash!!";

7. xxx[1] = 'D'; <span style="margin:0px;padding:0px;border:0px;color:rgb(102,102,102);font-style:italic;vertical-align:baseline;

10

background-color:transparent;">
8. return 2;
9. }
10.
11. int foo()
12. {
13. <span style="margin:0px;padding:0px;border:0px;color:rgb(177,177,0);vertical-align:baseline;background-

11

color:transparent;">return crash<span style="margin:0px;
padding:0px;border:0px;color:rgb(0,153,0);vertical-align:baseline;
background-color:transparent;">(<span style="margin:0px;
padding:0px;border:0px;color:rgb(0,153,0);vertical-align:baseline;
background-color:transparent;">)<span style="margin:0px;
padding:0px;border:0px;color:rgb(51,153,51);vertical-align:baseline;
background-color:transparent;">;

14. <span style="margin:0px;padding:0px;border:0px;color:rgb(0,153,0);
vertical-align:baseline;background-color:transparent;">}
15.
16. <span style="margin:0px;padding:0px;border:0px;color:rgb(153,51,51);
vertical-align:baseline;background-
color:transparent;">int main<span style="margin:0px;padding:0px;
border:0px;color:rgb(0,153,0);vertical-align:baseline;background-
color:transparent;">(<span style="margin:0px;padding:0px;
border:0px;color:rgb(0,153,0);vertical-align:baseline;background-
color:transparent;">)
17. <span style="margin:0px;padding:0px;border:0px;color:rgb(0,153,0);
vertical-align:baseline;background-color:transparent;">{
18. <span style="margin:0px;padding:0px;border:0px;
color:rgb(177,177,0);vertical-align:baseline;background-
color:transparent;">return foo<span style="margin:0px;

12

padding:0px;border:0px;color:rgb(0,153,0);vertical-align:baseline;
background-color:transparent;">(<span style="margin:0px;
padding:0px;border:0px;color:rgb(0,153,0);vertical-align:baseline;
background-color:transparent;">)<span style="margin:0px;
padding:0px;border:0px;color:rgb(51,153,51);vertical-align:baseline;
background-color:transparent;">;

19. <span style="margin:0px;padding:0px;border:0px;color:rgb(0,153,0);
vertical-align:baseline;background-color:transparent;">}

上手调试

测试如下代码

```
1
#include <stdio.h>
2
{
3
}
4
{
```


13

```
5  }
6
7
8
9
10
11
12
```

生成可执行文件并运行

gcc -o main a.c

root@ubuntu:~# ./main
Segmentation fault (core dumped)

14

<-----这里出现段错误并生成core文件了。

在/tmp目录下发现文件core-main-10815

如何查看进程挂在哪里了?

我们可以用

gdb main /tmp/core-main-10815

查看信息，发现能定位到函数了

Program terminated with signal 11, Segmentation fault.
#0 0x080483ba in func ()

如何定位到行?

在编译的时候开启-g调试开关就可以了

gcc -o main -g a.c

gdb main /tmp/core-main-10815

15

最终看到的结果如下，好棒。

Program terminated with signal 11, Segmentation fault.
#0 0x080483ba in func (p=0x0) at a.c:5
5   *p = 0;

总结一下，需要定位进程挂在哪一行我们只需要4个操作，

ulimit -c unlimited

echo "/tmp/core-%e-%p" > /proc/sys/kernel/core_pattern

gcc -o main -g a.c

gdb main /tmp/core-main-10815

就可以啦。

上边的程序编译的时候有一点需要注意，需要带上参数-g, 这样生成的可执行程序中会带上足够的调试信息。编译运行之后你就应该能看见期待已久的“Segment Fault(core dumped)”或是“段错误 (核心已转储)”之类的字眼了。看看当前目录下是不是有个core或是core.xxx的文件。祭出linux下经典的调试器

16

GDB，首先带着core文件载入程序：gdb exefile core，这里需要注意的这个core文件必须是exefile产生的，否则符号表会对不上。载入之后大概是这个样子的：

```
sagi@sagi-laptop:~$ gdb coredump core
Core was generated by ./coredump'.
    Program terminated with signal 11, Segmentation fault.
#0  0x080483a7 in crash () at coredump.c:8
    8       xxx[1] = 'D';
(gdb)
```

我们看到已经能直接定位到出core的地方了，在第8行写了一个只读的内存区域导致触发Segment Fault信号。在载入core的时候有个小技巧，如果你事先不知道这个core文件是由哪个程序产生的，你可以先随便找个代替一下，比如/usr/bin/w就是不错的选择。比如我们采用这种方法载入上边产生的core，gdb会有类似的输出：

```
sagi@sagi-laptop:~$ gdb /usr/bin/w core
Core was generated by ./coredump'.
    Program terminated with signal 11, Segmentation fault.
#0  0x080483a7 in ?? ()
    (gdb)
```


17

可以看到GDB已经提示你了，这个core是由哪个程序产生的。

GDB 常用操作

上边的程序比较简单，不需要另外的操作就能直接找到问题所在。现实却不是这样的，常常需要进行单步跟踪，设置断点之类的操作才能顺利定位问题。下边列出了GDB一些常用的操作。

- 启动程序：run
- 设置断点：b 行号|函数名
- 删除断点：delete 断点编号
- 禁用断点：disable 断点编号
- 启用断点：enable 断点编号
- 单步跟踪：next 也可以简写 n
- 单步跟踪：step 也可以简写 s
- 打印变量：print 变量名字
- 设置变量：set var=value
- 查看变量类型：ptype var
- 顺序执行到结束：cont
- 顺序执行到某一行： util lineno

18

- 打印堆栈信息：bt

19