根据core文件利用gdb定位问题

1. 进gdb,查看挂死的具体地址，将core文件拷贝到/home/bin目录下，然后进gdb，此时进程ipmc已经挂了。

(gdb) core-file core.ipmc.1495.11.1347890990

(no debugging symbols found)

Core was generated by `./ipmc -d'.

Program terminated with signal 11, Segmentation fault.

#0 0x1000522c in ?? ()

(gdb)

1. 找到挂死的地址0x1000522c

退出gdb，到shell下用 readelf查看core文件

[root@localhost:bin]#./readelf -W -l core.ipmc.1495.11.1347890990

Elf file type is CORE (Core file)

Entry point 0x0

There are 40 program headers, starting at offset 52

Program Headers:

Type Offset VirtAddr PhysAddr FileSiz MemSiz Flg Align

NOTE 0x000534 0x00000000 0x00000000 0x00824 0x00000 0

LOAD 0x001000 0x00100000 0x00000000 0x03000 0x03000 R E 0x1000

LOAD 0x004000 0x0fbe9000 0x00000000 0x00000 0x13c000 R E 0x1000

LOAD 0x004000 0x0fd25000 0x00000000 0x00000 0x3f000 0x1000

LOAD 0x004000 0x0fd64000 0x00000000 0x02000 0x02000 R 0x1000

LOAD 0x006000 0x0fd66000 0x00000000 0x03000 0x03000 RWE 0x1000

LOAD 0x009000 0x0fd69000 0x00000000 0x03000 0x03000 RWE 0x1000

LOAD 0x00c000 0x0fd7c000 0x00000000 0x00000 0x14000 R E 0x1000

LOAD 0x00c000 0x0fd90000 0x00000000 0x00000 0x40000 0x1000

LOAD 0x00c000 0x0fdd0000 0x00000000 0x01000 0x01000 R 0x1000

LOAD 0x00d000 0x0fdd1000 0x00000000 0x01000 0x01000 RWE 0x1000

LOAD 0x00e000 0x0fdd2000 0x00000000 0x02000 0x02000 RWE 0x1000

LOAD 0x010000 0x0fde4000 0x00000000 0x00000 0xa0000 R E 0x1000

LOAD 0x010000 0x0fe84000 0x00000000 0x00000 0x3f000 0x1000

LOAD 0x010000 0x0fec3000 0x00000000 0x01000 0x01000 R 0x1000

LOAD 0x011000 0x0fec4000 0x00000000 0x03000 0x03000 RWE 0x1000

LOAD 0x014000 0x0fed7000 0x00000000 0x00000 0x07000 R E 0x1000

LOAD 0x014000 0x0fede000 0x00000000 0x00000 0x3f000 0x1000

LOAD 0x014000 0x0ff1d000 0x00000000 0x01000 0x01000 R 0x1000

LOAD 0x015000 0x0ff1e000 0x00000000 0x01000 0x01000 RWE 0x1000

LOAD 0x016000 0x0ff2f000 0x00000000 0x00000 0x05000 R E 0x1000

LOAD 0x016000 0x0ff34000 0x00000000 0x00000 0x3f000 0x1000

LOAD 0x016000 0x0ff73000 0x00000000 0x01000 0x01000 R 0x1000

LOAD 0x017000 0x0ff74000 0x00000000 0x01000 0x01000 RWE 0x1000

LOAD 0x018000 0x0ff75000 0x00000000 0x00000 0x27000 RWE 0x1000

LOAD 0x018000 0x0ffac000 0x00000000 0x00000 0x04000 R E 0x1000

LOAD 0x018000 0x0ffb0000 0x00000000 0x00000 0x3f000 0x1000

LOAD 0x018000 0x0ffef000 0x00000000 0x01000 0x01000 RWE 0x1000

LOAD 0x019000 0x10000000 0x00000000 0x00000 0x56000 R E 0x1000

LOAD 0x019000 0x10095000 0x00000000 0x07000 0x07000 RWE 0x1000

LOAD 0x020000 0x1009c000 0x00000000 0x57000 0x57000 RWE 0x1000

LOAD 0x077000 0x48000000 0x00000000 0x00000 0x1d000 R E 0x1000

LOAD 0x077000 0x4801d000 0x00000000 0x04000 0x04000 RW 0x1000

LOAD 0x07b000 0x4805c000 0x00000000 0x01000 0x01000 R 0x1000

LOAD 0x07c000 0x4805d000 0x00000000 0x01000 0x01000 RWE 0x1000

LOAD 0x07d000 0x4805e000 0x00000000 0x01000 0x01000 0x1000

LOAD 0x07e000 0x4805f000 0x00000000 0x7ff000 0x7ff000 RW 0x1000

LOAD 0x87d000 0x4885e000 0x00000000 0x01000 0x01000 0x1000

LOAD 0x87e000 0x4885f000 0x00000000 0x7ff000 0x7ff000 RW 0x1000

LOAD 0x107d000 0xbff75000 0x00000000 0x21000 0x21000 RW 0x1000

[root@localhost:bin]#

地址0x1000522c所在的地址范围是

LOAD 0x019000 0x10000000 0x00000000 0x00000 0x56000 R E 0x1000

其中0x56000表示数据块大小 ，换算为KB (0x56000 / 1024) = 344KB

1. 根据344KB查找对应的文件

到生成core文件的设备上，当前挂死的是ipmc进程，重启ipmc进程，pid为2769.挂死时ipmc的pid是1495，重启之后的是2769。

通过命令 cat /proc/2769/smaps找到344KB对应的文件为 /home/bin/ipmc

[root@localhost:bin]#cat /proc/2769/smaps

10000000-10056000 r-xp 00000000 00:14 119 /home/bin/ipmc

Size: 344 kB

Rss: 164 kB

Pss: 164 kB

Shared\_Clean: 0 kB

Shared\_Dirty: 0 kB

Private\_Clean: 164 kB

Private\_Dirty: 0 kB

Referenced: 164 kB

Swap: 0 kB

KernelPageSize: 4 kB

MMUPageSize: 4 kB

或者通过命令

[root@localhost:bin]#cat /proc/2769/maps

00100000-00103000 r-xp 00000000 00:00 0 [vdso]

0fbe9000-0fd25000 r-xp 00000000 01:00 142 /lib/libc.so.6

0fd25000-0fd64000 ---p 0013c000 01:00 142 /lib/libc.so.6

0fd64000-0fd66000 r--p 0013b000 01:00 142 /lib/libc.so.6

0fd66000-0fd69000 rwxp 0013d000 01:00 142 /lib/libc.so.6

0fd69000-0fd6c000 rwxp 00000000 00:00 0

0fd7c000-0fd90000 r-xp 00000000 01:00 139 /lib/libpthread.so.0

0fd90000-0fdd0000 ---p 00014000 01:00 139 /lib/libpthread.so.0

0fdd0000-0fdd1000 r--p 00014000 01:00 139 /lib/libpthread.so.0

0fdd1000-0fdd2000 rwxp 00015000 01:00 139 /lib/libpthread.so.0

0fdd2000-0fdd4000 rwxp 00000000 00:00 0

0fde4000-0fe84000 r-xp 00000000 01:00 140 /lib/libm.so.6

0fe84000-0fec3000 ---p 000a0000 01:00 140 /lib/libm.so.6

0fec3000-0fec4000 r--p 0009f000 01:00 140 /lib/libm.so.6

0fec4000-0fec7000 rwxp 000a0000 01:00 140 /lib/libm.so.6

0fed7000-0fede000 r-xp 00000000 00:14 177 /home/bin/librt.so.1

0fede000-0ff1d000 ---p 00007000 00:14 177 /home/bin/librt.so.1

0ff1d000-0ff1e000 r--p 00006000 00:14 177 /home/bin/librt.so.1

0ff1e000-0ff1f000 rwxp 00007000 00:14 177 /home/bin/librt.so.1

0ff2f000-0ff34000 r-xp 00000000 01:00 166 /lib/libcrypt.so.1

0ff34000-0ff73000 ---p 00005000 01:00 166 /lib/libcrypt.so.1

0ff73000-0ff74000 r--p 00004000 01:00 166 /lib/libcrypt.so.1

0ff74000-0ff75000 rwxp 00005000 01:00 166 /lib/libcrypt.so.1

0ff75000-0ff9c000 rwxp 00000000 00:00 0

0ffac000-0ffb0000 r-xp 00000000 00:14 131 /home/bin/libcap.so.1

0ffb0000-0ffef000 ---p 00004000 00:14 131 /home/bin/libcap.so.1

0ffef000-0fff0000 rwxp 00003000 00:14 131 /home/bin/libcap.so.1

10000000-10056000 r-xp 00000000 00:14 119 /home/bin/ipmc

10095000-1009c000 rwxp 00055000 00:14 119 /home/bin/ipmc

1009c000-100d4000 rwxp 00000000 00:00 0 [heap]

48000000-4801d000 r-xp 00000000 01:00 172 /lib/ld.so.1

4801d000-48021000 rw-p 00000000 00:00 0

4805c000-4805d000 r--p 0001c000 01:00 172 /lib/ld.so.1

4805d000-4805e000 rwxp 0001d000 01:00 172 /lib/ld.so.1

4805e000-4805f000 ---p 00000000 00:00 0

4805f000-4885e000 rw-p 00000000 00:00 0

4885e000-4885f000 ---p 00000000 00:00 0

4885f000-4905e000 rw-p 00000000 00:00 0

bfa61000-bfa82000 rw-p 00000000 00:00 0 [stack]

[root@localhost:bin]#

可以找到对应的地址空间10000000-10056000是文件/home/bin/ipmc

1. readelf –S /home/bin/ipmc查找/home/bin/ipmc文件的.text段，已编译程序的机器代码

[root@localhost:bin]#./readelf -S /home/bin/ipmc

There are 42 section headers, starting at offset 0x11087c:

Section Headers:

[Nr] Name Type Addr Off Size ES Flg Lk Inf Al

[ 0] NULL 00000000 000000 000000 00 0 0 0

[ 1] .interp PROGBITS 10000134 000134 00000d 00 A 0 0 1

[ 2] .note.ABI-tag NOTE 10000144 000144 000020 00 A 0 0 4

[ 3] .gnu.hash GNU\_HASH 10000164 000164 000a40 04 A 4 0 4

[ 4] .dynsym DYNSYM 10000ba4 000ba4 0014b0 10 A 5 1 4

[ 5] .dynstr STRTAB 10002054 002054 0010de 00 A 0 0 1

[ 6] .gnu.version VERSYM 10003132 003132 000296 02 A 4 0 2

[ 7] .gnu.version\_r VERNEED 100033c8 0033c8 000130 00 A 5 5 4

[ 8] .rela.dyn RELA 100034f8 0034f8 000414 0c A 4 0 4

[ 9] .rela.plt RELA 1000390c 00390c 0005d0 0c A 4 26 4

[10] .init PROGBITS 10003edc 003edc 000044 00 AX 0 0 4

[11] .text PROGBITS 10003f20 003f20 040ef4 00 AX 0 0 4

[12] .fini PROGBITS 10044e14 044e14 000020 00 AX 0 0 4

[13] .rodata PROGBITS 10044e38 044e38 010df8 00 A 0 0 8

[14] .eh\_frame\_hdr PROGBITS 10055c30 055c30 000014 00 A 0 0 4

[15] .eh\_frame PROGBITS 10055c44 055c44 00002c 00 A 0 0 4

[16] .ctors PROGBITS 10095c70 055c70 000008 00 WA 0 0 4

[17] .dtors PROGBITS 10095c78 055c78 000008 00 WA 0 0 4

[18] .jcr PROGBITS 10095c80 055c80 000004 00 WA 0 0 4

[19] .data.rel.ro PROGBITS 10095c84 055c84 000200 00 WA 0 0 4

[20] .got2 PROGBITS 10095e84 055e84 00000c 00 WA 0 0 1

[21] .dynamic DYNAMIC 10095e90 055e90 0000f0 08 WA 5 0 4

[22] .data PROGBITS 10095f80 055f80 003fa8 00 WA 0 0 4

[23] .got PROGBITS 10099f28 059f28 0015cc 04 WAX 0 0 4

[24] .sdata PROGBITS 1009b4f4 05b4f4 000060 00 WA 0 0 4

[25] .sbss NOBITS 1009b554 05b554 000064 00 WA 0 0 4

[26] .plt NOBITS 1009b5b8 05b554 000618 00 WAX 0 0 4

[27] .bss NOBITS 1009bbd0 05b554 017178 00 WA 0 0 8

[28] .comment PROGBITS 00000000 05b554 000402 00 0 0 1

[29] .debug\_aranges PROGBITS 00000000 05b958 000640 00 0 0 8

[30] .debug\_pubnames PROGBITS 00000000 05bf98 005dd4 00 0 0 1

[31] .debug\_info PROGBITS 00000000 061d6c 05398b 00 0 0 1

[32] .debug\_abbrev PROGBITS 00000000 0b56f7 007552 00 0 0 1

[33] .debug\_line PROGBITS 00000000 0bcc49 00ba43 00 0 0 1

[34] .debug\_frame PROGBITS 00000000 0c868c 00a638 00 0 0 4

[35] .debug\_str PROGBITS 00000000 0d2cc4 00ee78 01 MS 0 0 1

[36] .debug\_loc PROGBITS 00000000 0e1b3c 02d32d 00 0 0 1

[37] .debug\_ranges PROGBITS 00000000 10ee69 001868 00 0 0 1

[38] .PPC.EMB.apuinfo NOTE 00000000 1106d1 000020 00 0 0 1

[39] .shstrtab STRTAB 00000000 1106f1 000189 00 0 0 1

[40] .symtab SYMTAB 00000000 110f0c 008b40 10 41 981 4

[41] .strtab STRTAB 00000000 119a4c 008eb2 00 0 0 1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)

O (extra OS processing required) o (OS specific), p (processor specific)

[root@localhost:bin]#

找到/home/bin/ipmc 文件中程序代码的地址为0x10003f20.

1. 进入产生core文件的设备gdb，利用add-symbol-file /home/bin/ipmc 0x10003f20命令，bt，找到函数调用栈。

[root@localhost:bin]#./gdb

dlopen failed on 'libthread\_db.so.1' - libthread\_db.so.1: cannot open shared object file: No such file or directory

GDB will not be able to debug pthreads.

GNU gdb Red Hat Linux (6.7-1rh)

Copyright (C) 2007 Free Software Foundation, Inc.

License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software: you are free to change and redistribute it.

There is NO WARRANTY, to the extent permitted by law. Type "show copying"

and "show warranty" for details.

This GDB was configured as "ppc-linux".

(gdb) add-symbol-file /home/bin/ipmc 0x10003f20

add symbol table from file "/home/bin/ipmc" at

.text\_addr = 0x10003f20

(y or n) y

Reading symbols from /home/bin/ipmc...done.

(gdb) core-file core.ipmc.1495.11.1347890990

Core was generated by `./ipmc -d'.

Program terminated with signal 11, Segmentation fault.

#0 0x1000522c in pim\_if\_jp\_interval\_get (ifindex=<value optimized out>) at ipmc\_if.c:417

417 ipmc\_if.c: No such file or directory.

in ipmc\_if.c

(gdb) bt

#0 0x1000522c in pim\_if\_jp\_interval\_get (ifindex=<value optimized out>) at ipmc\_if.c:417

#1 0x100052c4 in pim\_if\_jp\_holdtime\_get (ifindex=3261079552) at ipmc\_if.c:404

#2 0x10015cc0 in pim\_jp\_up\_wc\_jp\_send (ifindex=3261079552, upstream=<value optimized out>, grp\_addr=3892379905, rp\_addr=218959107,

type=1 '\001') at pim/pim\_join\_prune.c:1037

#3 0x10017660 in pim\_jp\_up\_wc\_j (wc\_mrt=0x100cf668, old\_wc\_mrt=0x0, event=<value optimized out>) at pim/pim\_join\_prune\_state\_machine.c:967

#4 0x10017a30 in pim\_jp\_up\_wc\_jt\_timer\_expire (thread=<value optimized out>) at pim/pim\_join\_prune\_state\_machine.c:869

#5 0x10035450 in thread\_call (thread=0xbff94d3c) at thread.c:1054

#6 0x100044d4 in main (argc=2, argv=<value optimized out>) at ipmc\_main.c:283

#7 0x0fc07260 in ?? ()

(gdb)

1. 到这里，就可以找到挂死的函数及函数栈。

当前挂死在所在文件及挂死行 ipmc\_if.c:417。

core.ipmc.1495.11.1347890990， signal 11表示访问空指针。

找到该行代码：

return(pif->pim\_conf->jp\_itv?pif->pim\_conf->jp\_itv : pim\_instance\_global->pim\_conf.jp\_itv);

(gdb) print pif

$4 = (struct ipmc\_if \*) 0x100cb3e0

(gdb) print pif->pim\_conf

$5 = (struct pim\_if\_config \*) 0x0 此处空指针

(gdb) print pif->pim\_conf->jp\_itv

Cannot access memory at address 0x8

1. 如果bt信息中不显示函数所在行，比如刚才的

#0 0x1000522c in pim\_if\_jp\_interval\_get (ifindex=<value optimized out>) at ipmc\_if.c:417

不显示ipmc\_if.c:417，就需要利用反汇编来继续往下查找挂死行。

1. 然而，反汇编我就不会了！