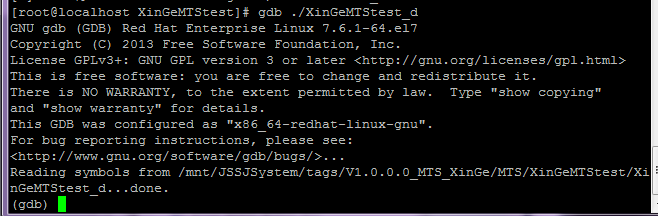
**GDB使用实测**

1. **以信鸽转码服务为例**
   1. 进入gdb环境。 gdb ./XinGeMTStest\_d



* 1. 设置断点。
     1. 对单个cpp的可执行文件。

B <行号N> ：在第N行设立断点。

* + 1. 对多cpp的可执行文件。

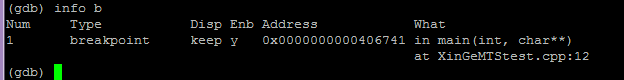
B xxx.cpp:N : 在xxx.cpp文件的第N行设置断点。



B namespace::class::method 使用 命令空间::类名::函数名



* 1. Info b 查看断点。(显示所有断点信息)



* 1. 调试

在设置断点以后，一定要先执行run命令。

R:run

N:next （类似F10）

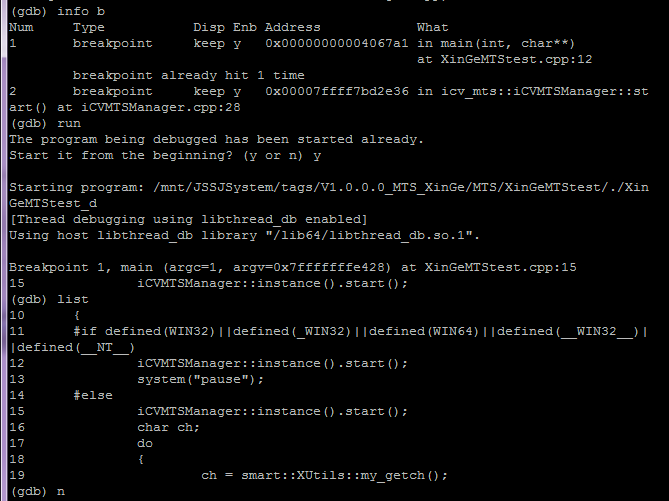
S:step (类似F11)

C:continue (类似F5)

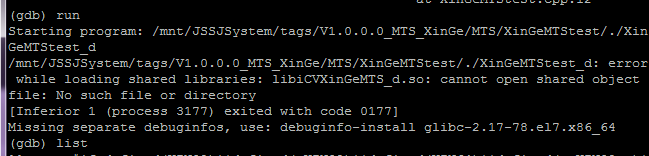
L:List (显示前后共10行)

Delete [num] 去掉断点

如果按回车，默认指前一个命令。



* 1. 在执行run时发现，报错，指libiCVXinGeMTS\_d.so 找不到。这涉及在调试某个可执行文件的同时，他需要加载其他路径的动态库。



* + 1. 将需要的库路径加入LD\_LIBRARY\_PATH 中。
    2. 确保该路径下一定有这个动态库。
    3. 比如我需要的库对应路径如下：

/mnt/JSSJSystem/tags/V1.0.0.0\_MTS\_XinGe/MTS/iCVXinGeMTS

libiCVXinGeMTS\_d.so

/mnt/JSSJSystem/tags/V1.0.0.0\_MTS\_XinGe/Depend/Linux/framework/ace/lib

libACE.s.6.4.3

/mnt/JSSJSystem/tags/V1.0.0.0\_MTS\_XinGe/Depend/Linux/framework/lib64

libiCVFramework\_d.so

libiCVParserFactory\_d.so

libiCVLog\_d.so

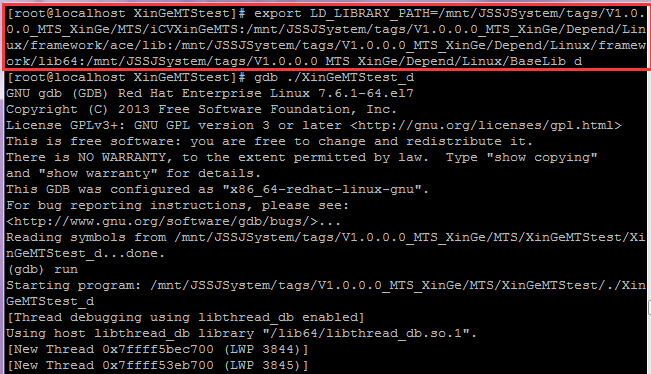
/mnt/JSSJSystem/tags/V1.0.0.0\_MTS\_XinGe/Depend/Linux/BaseLib\_d

libjsoncpp\_d.so

libsmart\_d.so

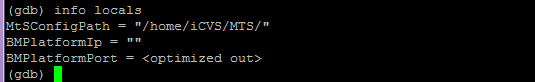
libpugixml\_d.so

那就需要在进入gdb之前，加载一下动态路径。



* 1. 打印某个变量

1.6.1 查看当前程序栈的局部变量



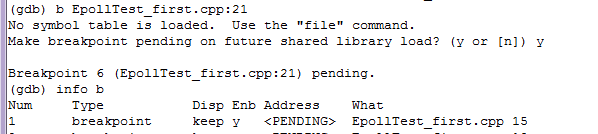
1.6.1 打印某个变量。



* 1. 指定运行时参数
     1. set args 可指定运行时参数（如：set args 10 20 30 40 50）。
     2. show args 命令可以查看设置好的运行参数。

2 错误搜集

2.1 打断点。



解决: 该问题是由于，使用g++ 编译.cpp文件的时候没加 -g参数。重新编译一遍即可。



3 core文件分析。

更改linux设置，使程序崩溃的时候产生core文件，然后gdb调试。

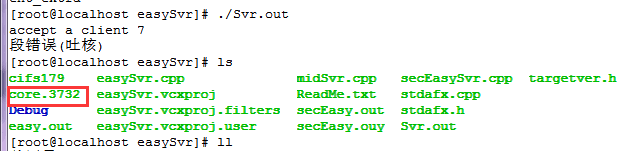
3.1 设置第83行，内存越界。



在运行之前，设置 ulimit -c 。Ulimit -c unlimited 设置可以产生coredump且不受大小限制。

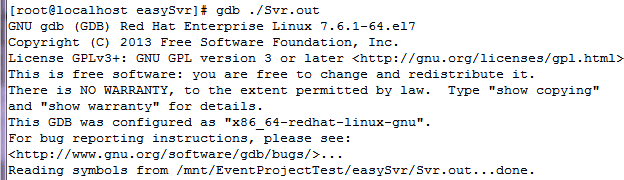


在产生崩溃以后，会生成core.

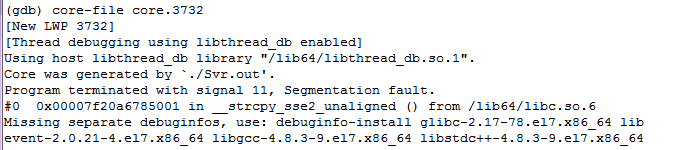


3.2 调试core。

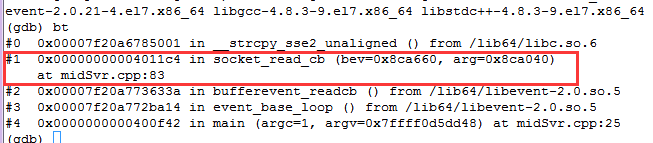
Gdb ./Svr.out 进入gdb状态



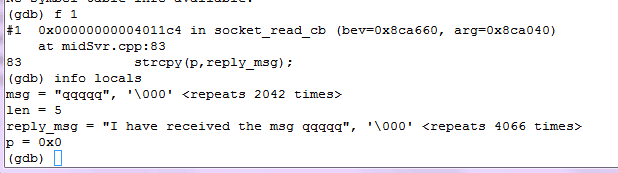
Core-file core.3732 加载core里面的堆栈信息



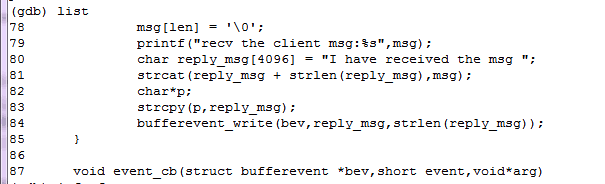
3.3 Bt： 加载堆栈信息



3.4 打印堆栈参数



3.5 list 列出上前后5行



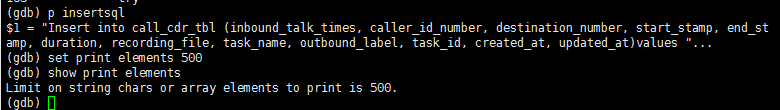
1. 多进程调试  
    使用gdb多进程调试时，如果想在进程间进行切换，那么在fork调用前，
2. 设置： set detach-on-fork off
3. 然后可以使用info inferiors查看进程信息，最前面有一个进程编号。
4. 使用inferior num 来切换到相应的进程。
5. 多线程调试
6. info threads 显示所有的线程（gdb给每个线成一个编号，一般是从1开始）
7. thread num 跳转到某个线程。
8. set scheduler-locking off|on|step 设置在调试某个线程的时候，其他线程是否执行。

Off : 不锁定人写线程，默认值

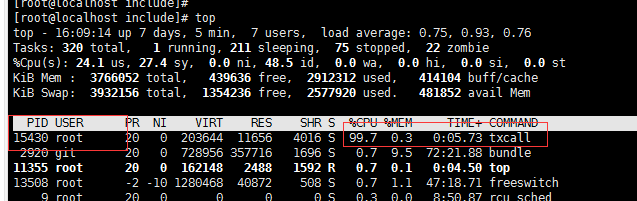
On: 锁定其他线程，只有当前线程执行，

Step: 在step（单步时），只有被调试线程运行。

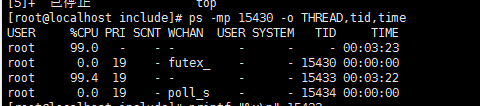
1. show scheduler-locking 查看当前锁定线程的模式。
2. 设置打印字数



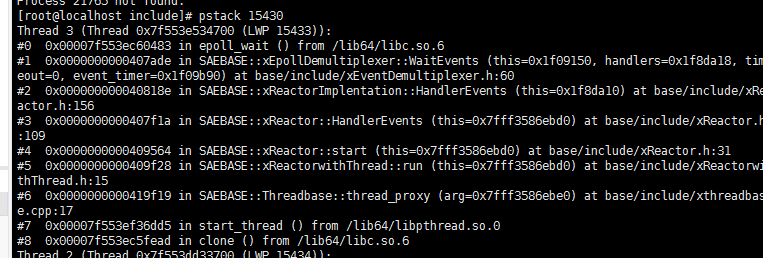
1. 查看cpu使用。
2. Top



（2）查看对应线程ps -mp 15430 -o THREAD,tid,time



（3）pstack 15400 插选线程的堆栈



最后找出是epoll\_wait 的timeout参数为0的原因。



查看程序使用的哪里的动态库

