[nginx源码分析--使用GDB调试](http://blog.csdn.net/yusiguyuan/article/details/26981329)

标签： [nginx](http://www.csdn.net/tag/nginx)

2014-05-25 21:40 1622人阅读 [评论](http://blog.csdn.net/yusiguyuan/article/details/26981329#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/yusiguyuan/article/details/26981329#report)

本文章已收录于：

[[](http://lib.csdn.net/base/19?source=blogtop) **嵌入式开发知识库**](http://lib.csdn.net/base/19?source=blogtop)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

nginx源码分析（53） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)[+]](http://blog.csdn.net/yusiguyuan/article/details/26981329)

在学习优秀的源代码时是 少不了源码的跟踪与调试，它不仅是我们解决程序bug的有效途径 也是我们理解、学习优秀源码的有效途径。

本文主要介绍一些源码调试的方法，并结合Nginx 源码进行示例。

**1， 利用GDB调试**

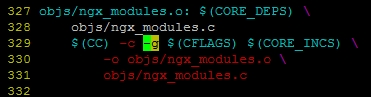
**a**，首先你应该熟悉 GDB 调试的一些基本命令（不熟悉的移步 至 [用GDB调试程序](http://blog.csdn.net/haoel/article/details/2879) ,熟悉step，run，break，list，info，continue等命令)。

**b**,  下载nginx 源码，这里使用nginx-1.0.14，解压文件。其中auto文件夹里包含了configure 运行时的各种命令集合，src是源码。为了利用

        GDB调试Nginx，需要在生成Nginx程序时把 -g 编译选项打开。我们需要修改  auto/cc/conf 文件 把 ngx\_compile\_opt="-c"  加上  -g 选项

         变为 ngx\_compile\_opt="-c -g"， 下一步执行configure命令： sudo ./configure  ，然后运行命令：vim  objs/Makefile  确认一下-g参数是

         否加上了。如下图。



         -g  编译选项已打开，然后执行命令: sudo make .(如果之前已经执行过make，那么第二次make时 需要 确保能够重新编译，此时可以通过

            刷新所有源文件时间戳，间接达到重新编译出一个新的Nginx可执行程序，命令为: find . -name "\*.c" | xargs touch ), 好了，Nginx编译成功。

**c**, 启动 nginx ,在objs目录下执行命令：sudo  ./nginx ,成功运行nginx后执行命令： ps -ef | grep nginx， 查看nginx的master及worker进程的PID，

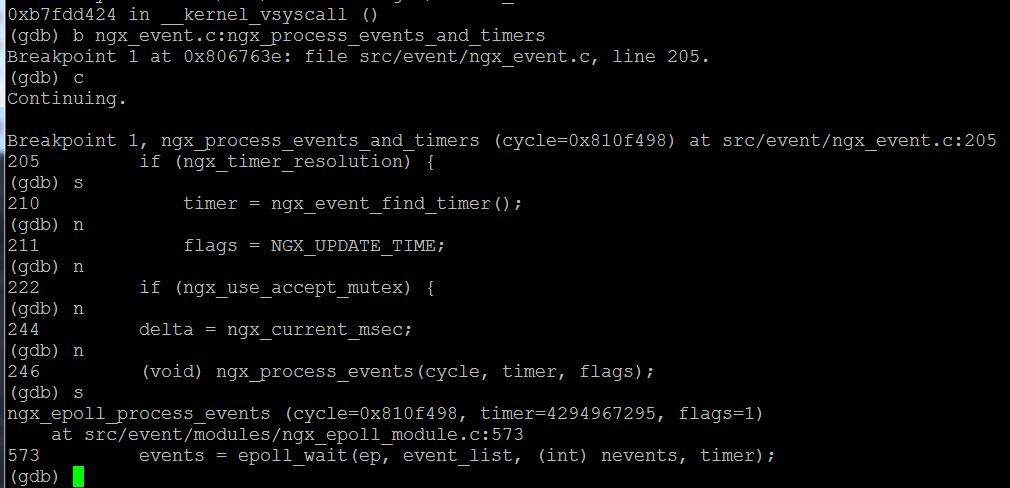
        如果对nginx 工作进程 2177进行gdb调试，那么可以利用gdb的 -p 选项。 命令： gdb  -p 2177.

      当有多个工作进程时 调试起来比较麻烦。我们可以修改配置项，修改文件 nginx.conf 。加入master\_process  off;  单进程模式 将监控进程和

      工作 进程逻辑全部合在一个 进程里。    此时只有一个进程，可以方便的利用gdb进行调试。

     我们知道工作进程会停留在epoll\_wait处等待相应的事件发生，而这个函数调用被封装在ngx\_process\_events\_and\_timers 中。于是我们在

     这个函数设置一个断点： b ngx\_event.c:ngx\_process\_events\_and\_timers ，结合gdb 命令c ,s,n  如图所示：



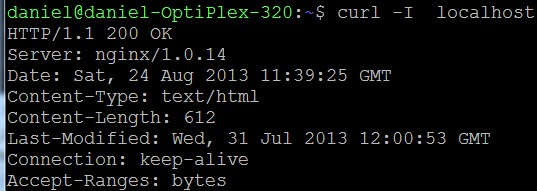
   采用命令c，使得nginx一直运行，直到遇到第一个断点,处理事件的方法是ngx\_process\_events，于是我们用命令 s 跟踪进去这个函数 .当执行到

   epoll\_wait函数的时候，发现进程停留在这里，不能在向下执行。这就证明了 worker子进程阻塞在epoll\_wait函数调用处。此时我们在另一个终端

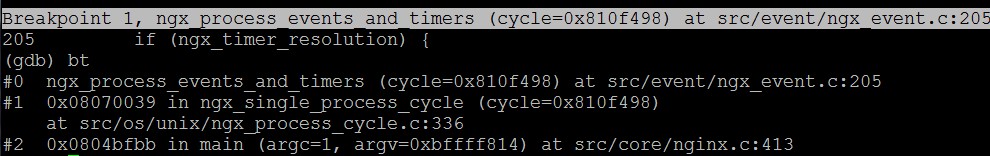
   执行下列命令，以向nginx发送消息： curl -I localhost . 可以看到请求已经发送，正在等待回应。

   此时继续执行命令 c  即可在另一终端得到 回应。此时可以通过bt 命令查看单进程模式下函数调用的过程。如图。

   利用curl 命令。



 gdb  bt



**2，利用strace、  pstack 调试 nginx**

**a**, strace 常用来跟踪进程执行时的系统调用和所接收的信号。 在[**Linux**](http://lib.csdn.net/base/linux)世界，进程不能直接访问硬件设备，当进程需要访问硬件设备(比如读取

      磁盘文件，接收网络数据等等)时，必须由用户态模式切换至内核态模式，通 过系统调用访问硬件设备。strace可以跟踪到一个进程产生的系

     统调用,包括参数，返回值，执行消耗的时间。 此外命令 ltrace 用来查看 动态库函数调用。

**b**, 那具体是哪一个函数调用呢？在strace输出结果中并不能找到答案，因其输出显示都是系统调用，要显示程序中函数调用栈信息，就轮到pstack

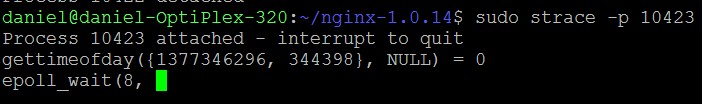
    上场了。pstack是一个脚本工具，其核心实现就是使用了gdb以及thread apply all bt命令.

  可以通过命令 man 来 查看 Man手册中 strace、ltrace、pstack的具体具体用法。

 我们在此 修改 nginx的配置文件 nginx.conf  使其为 仅具有 一个 master 进程和一个 worker进程.  把1中修改的 master\_process  off; 注解掉 加上 #

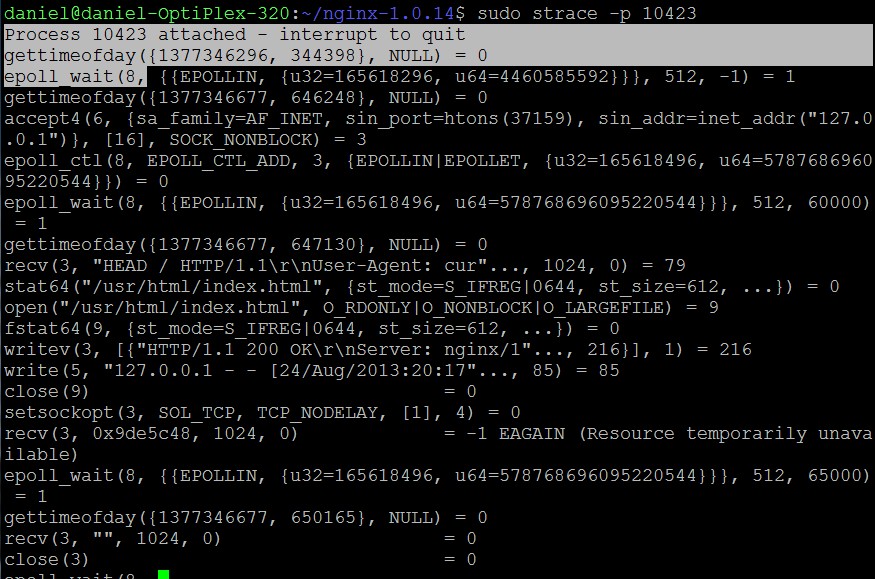
 并在配置文件上加上 worker\_processes  1;

 关闭之前的nginx 重新启动 nginx。 利用命令： ps -ef | grep nginx  查看当前存在的nginx进程，然后用strace 命令的参数 -p 选项跟踪 Nginx 工作进程，如图。



  可以看到工作进程阻塞在 epoll\_wait 系统调用上，因为此时没有客户端请求，nginx就阻塞于此，在另一个终端执行curl 命令： curl -I localhost ，

  再来看strace输出 结果如图。



strace输出的每一行记录一次系统调用，等号左边是系统调用名以及调用参数，等号右边是该系统调用的返回值。

通过上面的系统调用我们可以做出如下分析：

 ⑴.   epoll\_wait返回值为1，表示有1个描述符存在可读/写事件，这里当然是可读事件。  
 ⑵.   accept4接受该请求，返回的数字3表示socket的文件描述符。  
 ⑶.   epoll\_ctl把accept4建立的socket套接字（注意参数3）加入到事件监听机制里。  
 ⑷.   recv从发生可读事件的socket文件描述符内读取数据，读取的数据存在第二个参数内，读取了79个字节。  
 ⑸.   stat64判断客户端请求的html文件是否存在，返回值为0表示存在。  
 ⑹.   open/fstat64打开并获取文件状态信息。open文件返回的文件描述符为9，后面几个系统调用都用到这个值。  
 ⑺.   writev把响应头通过文件描述符3代表的socket套接字发给客户端。  
 ⑻.   sendfile64把文件描述符9代表的响应体通过文件描述符3代表的socket套接字发给客户端。  
 ⑼.   再往文件描述符4代表的日志文件内write一条日志信息。  
 ⑽.   recv看客户端是否还发了其它待处理的请求/信息。  
 ⑾.   最后关闭文件描述符3代表的socket套接字。  
  由于strace能够提供nginx执行过程中的这些内部信息，所以在出现一些奇怪现象，比如nginx启动失败、响应的文件数据和预期不一致、莫名其妙

  的Segment Fault段错误、存在性能瓶颈（利用-T选项跟踪各个函数的消耗时间），利用strace也许能提供一些相关帮助。最后，要退出strace跟踪，按ctrl+c即可。

详细介绍strace 参数：

-c 统计每一系统调用的所执行的时间,次数和出错的次数等.  
-d 输出strace关于标准错误的调试信息.   
-f 跟踪由fork调用所产生的子进程.   
-ff 如果提供-o filename,则所有进程的跟踪结果输出到相应的filename.pid中,pid是各进程的进程号.   
-F 尝试跟踪vfork调用.在-f时,vfork不被跟踪.   
-h 输出简要的帮助信息.   
-i 输出系统调用的入口指针.   
-q 禁止输出关于脱离的消息.   
-r 打印出相对时间关于,,每一个系统调用.   
-t 在输出中的每一行前加上时间信息.   
-tt 在输出中的每一行前加上时间信息,微秒级.   
-ttt 微秒级输出,以秒了表示时间.   
-T 显示每一调用所耗的时间.   
-v 输出所有的系统调用.一些调用关于环境变量,状态,输入输出等调用由于使用频繁,默认不输出.   
-V 输出strace的版本信息.   
-x 以十六进制形式输出非标准字符串   
-xx 所有字符串以十六进制形式输出.   
-a column   
设置返回值的输出位置.默认 为40.   
-e expr   
指定一个表达式,用来控制如何跟踪.格式如下:   
[qualifier=][!]value1[,value2]...   
qualifier只能是 trace,abbrev,verbose,raw,signal,read,write其中之一.value是用来限定的符号或数字.默认的 qualifier是 trace.感叹号是否定符号.例如:  
-eopen等价于 -e trace=open,表示只跟踪open调用.而-etrace!=open表示跟踪除了open以外的其他调用.有两个特殊的符号 all 和 none.  
注意有些shell使用!来执行历史记录里的命令,所以要使用\\.   
-e trace=set   
只跟踪指定的系统 调用.例如:-e trace=open,close,rean,write表示只跟踪这四个系统调用.默认的为set=all.   
-e trace=file   
只跟踪有关文件操作的系统调用.   
-e trace=process   
只跟踪有关进程控制的系统调用.   
-e trace=network   
跟踪与网络有关的所有系统调用.   
-e strace=signal   
跟踪所有与系统信号有关的 系统调用   
-e trace=ipc   
跟踪所有与进程通讯有关的系统调用   
-e abbrev=set   
设定 strace输出的系统调用的结果集.-v 等与 abbrev=none.默认为abbrev=all.   
-e raw=set   
将指 定的系统调用的参数以十六进制显示.   
-e signal=set   
指定跟踪的系统信号.默认为all.如 signal=!SIGIO(或者signal=!io),表示不跟踪SIGIO信号.   
-e read=set   
输出从指定文件中读出 的数据.例如:   
-e read=3,5   
-e write=set   
输出写入到指定文件中的数据.   
-o filename   
将strace的输出写入文件filename   
-p pid   
跟踪指定的进程pid.   
-s strsize   
指定输出的字符串的最大长度.默认为32.文件名一直全部输出.   
-u username   
以username 的UID和GID执行被跟踪的命令

strace 通用的完整用法：

strace -o output.txt -T -tt -e trace=all -p 10423

上面的含义是 跟踪28979进程的所有系统调用（-e trace=all），并统计系统调用的花费时间，以及开始时间（并以可视化的时分秒格式显示），最后将记录结果存在

output.txt文件里面。

限制strace只跟踪特定的系统调用:

如果你已经知道你要找什么，你可以让strace只跟踪一些类型的系统调用。例如，在nginx执行程序时，你需要监视的系统调用epoll\_wait。

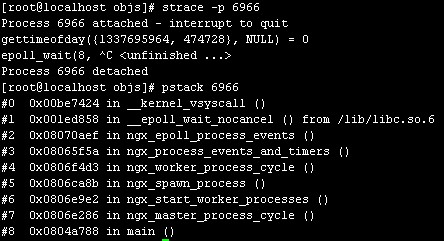
让strace只记录epoll\_wait的调用用这个命令：

strace -f -o epoll-strace.txt -e epoll\_wait -p 10423

命令strace跟踪的是系统调用，对于nginx本身的函数调用关系无法给出更为明朗的信息，如果我们发现nginx当前运行不正常，想知道nginx当前内部到底在执行什么函数，

那么命令pstack就是一个非常方便实用的工具。pstack的使用也非常简单，后面跟进程id即可，比如在无客户端请求的情况下，nginx阻塞在epoll\_wait系统调用处，此时

利用pstack查看到的nginx函数调用堆栈关系如下：



从main()函数到epoll\_wait()函数的调用关系一目了然，和在gdb内看到的堆栈信息一样。我们可以利用此进行分析优化等。

除了 1,2 还有方法 加桩调试等方法在此不再叙述。以后有机会可以介绍下特殊应用逻辑的调试。