(1条消息)线程的查看以及利用gdb调试多线程 - zhangye3017的博客 - CSDN博客

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

更多linux知识点：linux目录索引

---

1. 线程的查看

首先创建两个线程：

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <stdlib.h>
#include <string.h>

void* pthread_run1(void* arg)
{
    (void)arg;

    while(1)
    {
        printf("I am thread1,ID: %d\n",pthread_self());
        sleep(1);
    }
}

void* pthread_run2(void* arg)
{
    (void)arg;

    while(1)
    {
        printf("I am thread2,ID: %d\n",pthread_self());
        sleep(1);
    }
}


int main()
{

    pthread_t tid1;
    pthread_t tid2;

    pthread_create(&tid1,NULL,pthread_run1,NULL);
    pthread_create(&tid2,NULL,pthread_run2,NULL);

    printf("I am main thread\n");

    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
    return 0;
}
```

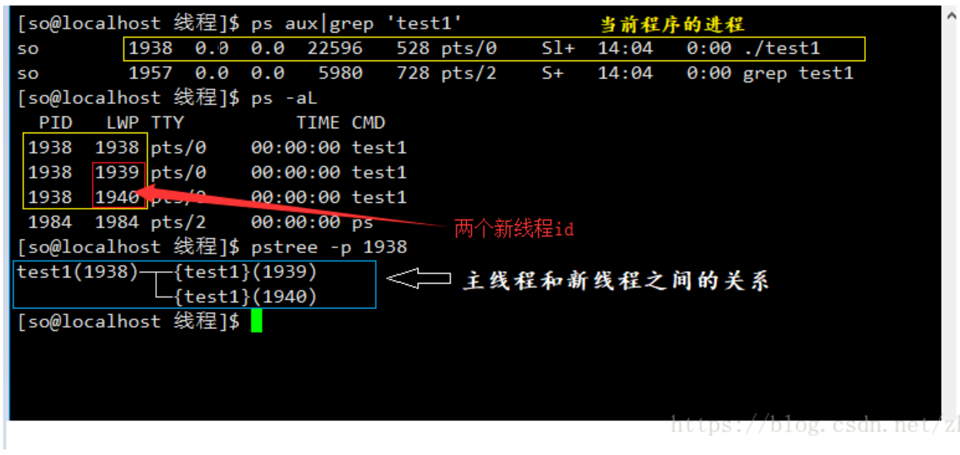
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44
- 45

分析：上面程序中创建了两个线程，程序执行起来，main函数所在程序为主线程，在这个主线程中有两个新线程运行

命令行查看：

```
ps aux|grep a.out

ps -aL|grep a.out

pstree -p 主线程id
```

- 1
- 2
- 3
- 4
- 5
- 6



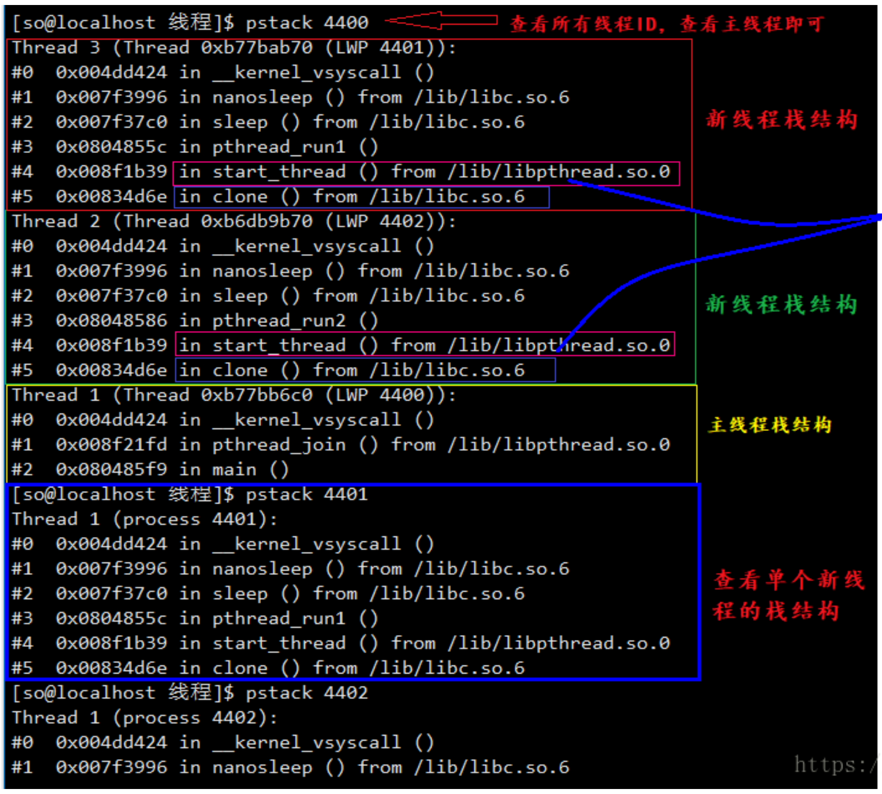
```
[so@localhost 线程]$ ps aux|grep 'test1'
so        1938  0.0  0.0  22596   528 pts/0    Sl+  14:04   0:00 ./test1
so        1957  0.0  0.0   5980   728 pts/2    S+   14:04   0:00 grep test1
[so@localhost 线程]$ ps -aL
  PID   LWP TTY          TIME CMD
 1938  1938 pts/0    00:00:00 test1
 1938  1939 pts/0    00:00:00 test1
 1938  1940 pts/0    00:00:00 test1
 1984  1984 pts/2    00:00:00 ps
[so@localhost 线程]$ pstree -p 1938
test1(1938)─┬─{test1}(1939)
            └─{test1}(1940)
[so@localhost 线程]$
```

https://blog.csdn.net/zhangye3017

2. 线程栈结构的查看

```
1. 获取线程ID
2. 通过命令查看栈结构 ps stack 线程ID
```

- 1
- 2



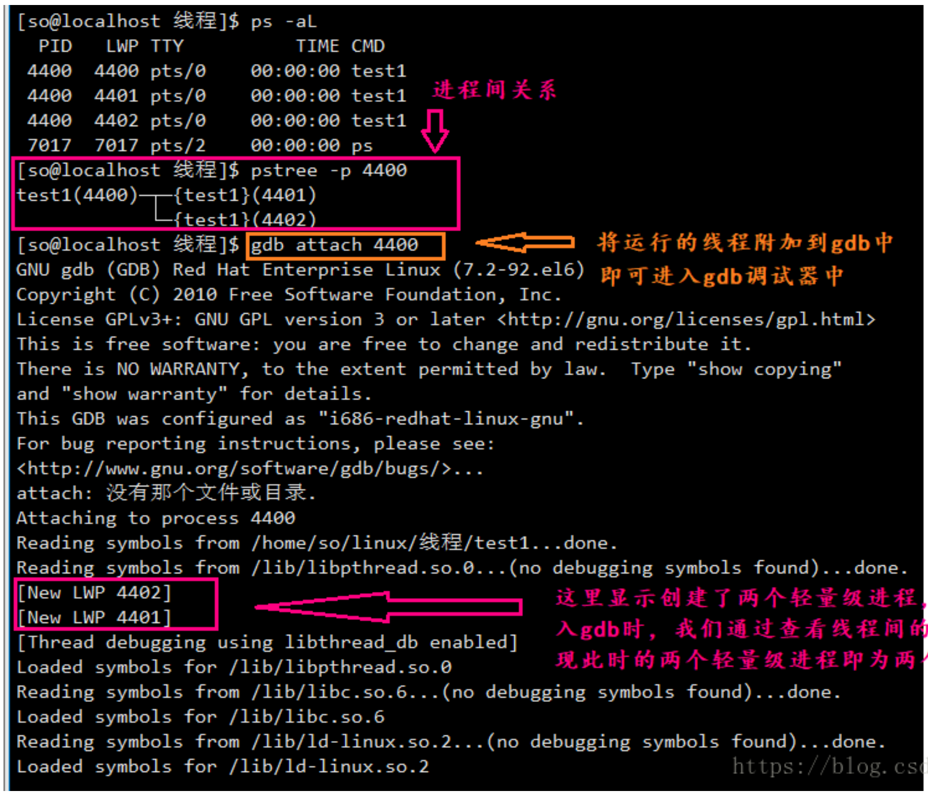
```
[so@localhost 线程]$ pstack 4400
Thread 3 (Thread 0xb77bab70 (LWP 4401)):
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x007f3996 in nanosleep () from /lib/libc.so.6
#2  0x007f37c0 in sleep () from /lib/libc.so.6
#3  0x0804855c in pthread_run1 ()
#4  0x008f1b39 in start_thread () from /lib/libpthread.so.0
#5  0x00834d6e in clone () from /lib/libc.so.6
Thread 2 (Thread 0xb6db9b70 (LWP 4402)):
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x007f3996 in nanosleep () from /lib/libc.so.6
#2  0x007f37c0 in sleep () from /lib/libc.so.6
#3  0x08048586 in pthread_run2 ()
#4  0x008f1b39 in start_thread () from /lib/libpthread.so.0
#5  0x00834d6e in clone () from /lib/libc.so.6
Thread 1 (Thread 0xb77bb6c0 (LWP 4400)):
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x008f21fd in pthread_join () from /lib/libpthread.so.0
#2  0x080485f9 in main ()
[so@localhost 线程]$ pstack 4401
Thread 1 (process 4401):
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x007f3996 in nanosleep () from /lib/libc.so.6
#2  0x007f37c0 in sleep () from /lib/libc.so.6
#3  0x0804855c in pthread_run1 ()
#4  0x008f1b39 in start_thread () from /lib/libpthread.so.0
#5  0x00834d6e in clone () from /lib/libc.so.6
[so@localhost 线程]$ pstack 4402
Thread 1 (process 4402):
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x007f3996 in nanosleep () from /lib/libc.so.6
```

查看所有线程ID，查看主线程即可
新线程栈结构
这两句更能说明两个新线是克隆了主线程的PCB，并且是主线程的一个执行分支
新线程栈结构
主线程栈结构
查看单个新线程的栈结构

https://blog.csdn.net/zhangye3017

3. 利用gdb查看线程信息

1. 将进程附加到gdb调试器当中，查看是否创建了新线程：gdb attach 主线程ID



```
[so@localhost 线程]$ ps -aL
  PID   LWP TTY          TIME CMD
 4400  4400 pts/0    00:00:00 test1
 4400  4401 pts/0    00:00:00 test1
 4400  4402 pts/0    00:00:00 test1
 7017  7017 pts/2    00:00:00 ps
[so@localhost 线程]$ pstree -p 4400
test1(4400)─┬─{test1}(4401)
            └─{test1}(4402)
[so@localhost 线程]$ gdb attach 4400
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-92.el6)
Copyright (C) 2010 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 "i686-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
attach: 没有那个文件或目录.
Attaching to process 4400
Reading symbols from /home/so/linux/线程/test1...done.
Reading symbols from /lib/libpthread.so.0...(no debugging symbols found)...done.
[New LWP 4402]
[New LWP 4401]
[Thread debugging using libthread_db enabled]
Loaded symbols for /lib/libpthread.so.0
Reading symbols from /lib/libc.so.6...(no debugging symbols found)...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
```

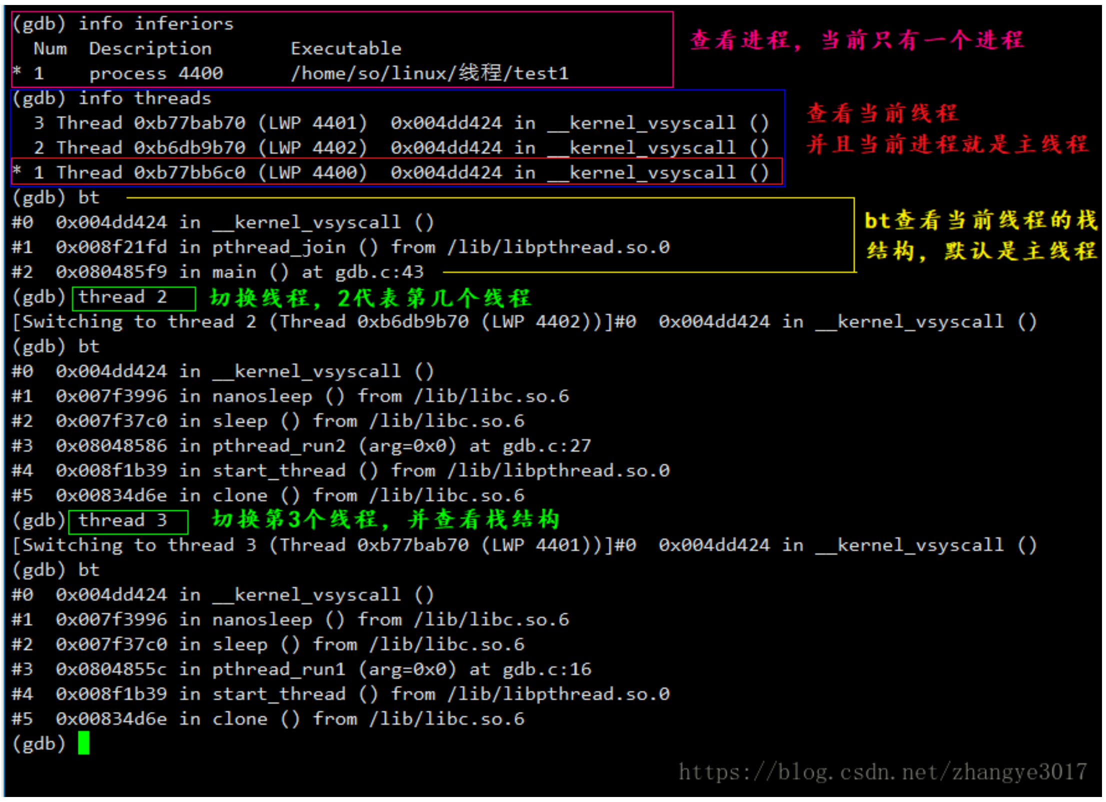
进程间关系
将运行的线程附加到gdb中即可进入gdb调试器中
这里显示创建了两个轻量级进程，在没有进入gdb时，我们通过查看线程间的关系，发现此时的两个轻量级进程即为两个新线程

https://blog.csdn.net/zhangye3017

2. 查看线程的一些信息

- 1
- 2

- 3
- 4



```
(gdb) info inferiors
  Num  Description       Executable
* 1    process 4400      /home/so/linux/线程/test1
(gdb) info threads
  3 Thread 0xb77bab70 (LWP 4401)  0x004dd424 in __kernel_vsyscall ()
  2 Thread 0xb6db9b70 (LWP 4402)  0x004dd424 in __kernel_vsyscall ()
* 1 Thread 0xb77bb6c0 (LWP 4400)  0x004dd424 in __kernel_vsyscall ()
(gdb) bt
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x008f21fd in pthread_join () from /lib/libpthread.so.0
#2  0x080485f9 in main () at gdb.c:43
(gdb) thread 2
[Switching to thread 2 (Thread 0xb6db9b70 (LWP 4402))]#0  0x004dd424 in __kernel_vsyscall ()
(gdb) bt
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x007f3996 in nanosleep () from /lib/libc.so.6
#2  0x007f37c0 in sleep () from /lib/libc.so.6
#3  0x08048586 in pthread_run2 (arg=0x0) at gdb.c:27
#4  0x008f1b39 in start_thread () from /lib/libpthread.so.0
#5  0x00834d6e in clone () from /lib/libc.so.6
(gdb) thread 3
[Switching to thread 3 (Thread 0xb77bab70 (LWP 4401))]#0  0x004dd424 in __kernel_vsyscall ()
(gdb) bt
#0  0x004dd424 in __kernel_vsyscall ()
#1  0x007f3996 in nanosleep () from /lib/libc.so.6
#2  0x007f37c0 in sleep () from /lib/libc.so.6
#3  0x0804855c in pthread_run1 (arg=0x0) at gdb.c:16
#4  0x008f1b39 in start_thread () from /lib/libpthread.so.0
#5  0x00834d6e in clone () from /lib/libc.so.6
(gdb) 
```

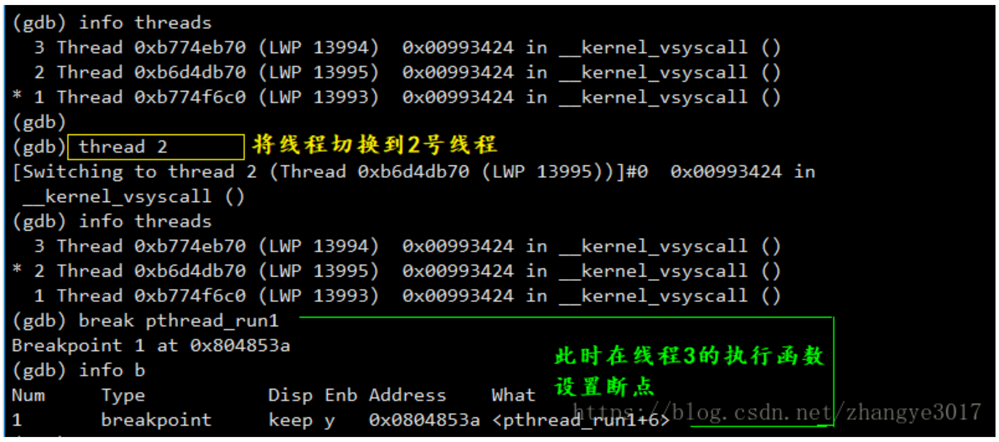
查看进程，当前只有一个进程
查看当前线程
并且当前进程就是主线程
bt查看当前线程的栈结构，默认是主线程
切换线程，2代表第几个线程
切换第3个线程，并查看栈结构
https://blog.csdn.net/zhangye3017

4. 利用gdb调试多线程

当程序没有启动，线程还没有执行，此时利用gdb调试多线程和调试普通程序一样，通过设置断点，运行，查看信息等等，在这里不在演示，最后会加上调试线程的命令

1. 设置断点

   - 1
   - 2

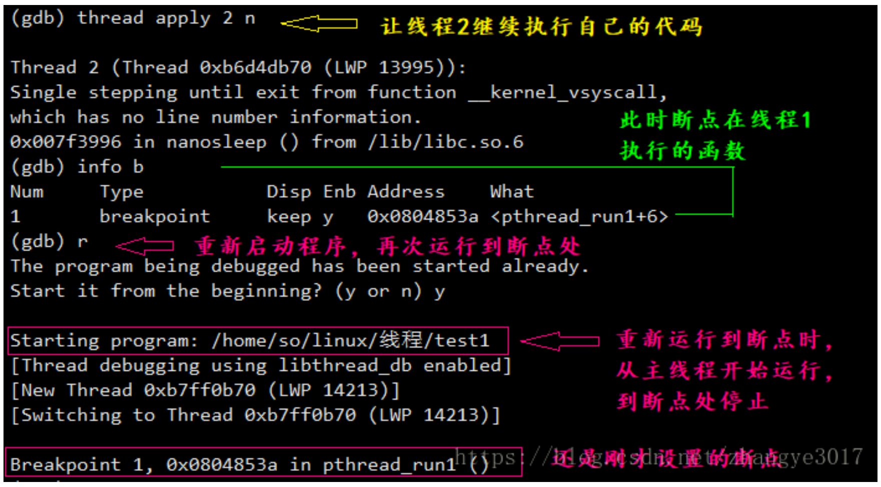


```
(gdb) info threads
  3 Thread 0xb774eb70 (LWP 13994)  0x00993424 in __kernel_vsyscall ()
  2 Thread 0xb6d4db70 (LWP 13995)  0x00993424 in __kernel_vsyscall ()
* 1 Thread 0xb774f6c0 (LWP 13993)  0x00993424 in __kernel_vsyscall ()
(gdb)
(gdb) thread 2
[Switching to thread 2 (Thread 0xb6d4db70 (LWP 13995))]#0  0x00993424 in
 __kernel_vsyscall ()
(gdb) info threads
  3 Thread 0xb774eb70 (LWP 13994)  0x00993424 in __kernel_vsyscall ()
* 2 Thread 0xb6d4db70 (LWP 13995)  0x00993424 in __kernel_vsyscall ()
  1 Thread 0xb774f6c0 (LWP 13993)  0x00993424 in __kernel_vsyscall ()
(gdb) break pthread_run1
Breakpoint 1 at 0x804853a
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x0804853a <pthread_run1+6>
```

将线程切换到2号线程
此时在线程3的执行函数设置断点
https://blog.csdn.net/zhangye3017

2. 执行线程2的函数，指行完毕继续运行到断点处

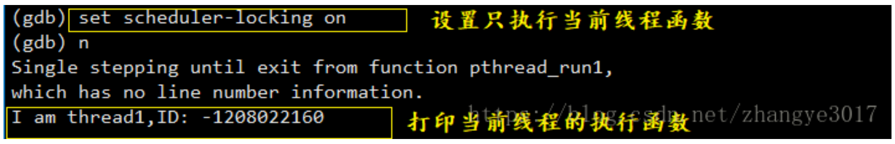
```
1. 继续使某一线程运行：thread apply 1-n（第几个线程） n
2. 重新启动程序运行到断点处：r
```
   - 1
   - 2


让线程2继续执行自己的代码
此时断点在线程1执行的函数
重新启动程序，再次运行到断点处
重新运行到断点时，从主线程开始运行，到断点处停止
这是刚才设置的断点

```
(gdb) thread apply 2 n

Thread 2 (Thread 0xb6d4db70 (LWP 13995)):
Single stepping until exit from function __kernel_vsyscall,
which has no line number information.
0x007f3996 in nanosleep () from /lib/libc.so.6
(gdb) info b
Num     Type           Disp Enb Address    What
1       breakpoint     keep y   0x0804853a <pthread_run1+6>
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y

Starting program: /home/so/linux/线程/test1
[Thread debugging using libthread_db enabled]
[New Thread 0xb7ff0b70 (LWP 14213)]
[Switching to Thread 0xb7ff0b70 (LWP 14213)]

Breakpoint 1, 0x0804853a in pthread_run1 ()
```
https://blog.csdn.net/zhangye3017

3. 只运行当前线程

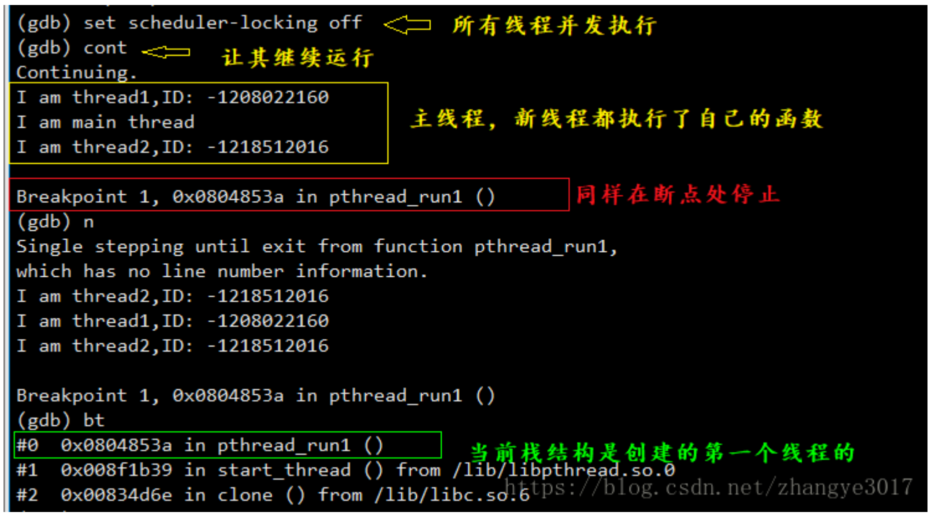
```
1. 设置：set scheduler-locking on
2. 运行：n
```
- 1
- 2



```
(gdb) set scheduler-locking on
(gdb) n
Single stepping until exit from function pthread_run1,
which has no line number information.
I am thread1,ID: -1208022160
```
https://blog.csdn.net/zhangye3017

4. 所有线程并发执行

```
1. 设置：set scheduler-locking off
2. 运行：n
```
- 1
- 2


所有线程并发执行
让其继续运行
主线程，新线程都执行了自己的函数
同样在断点处停止
当前栈结构是创建的第一个线程的

```
(gdb) set scheduler-locking off
(gdb) cont
Continuing.
I am thread1,ID: -1208022160
I am main thread
I am thread2,ID: -1218512016

Breakpoint 1, 0x0804853a in pthread_run1 ()
(gdb) n
Single stepping until exit from function pthread_run1,
which has no line number information.
I am thread2,ID: -1218512016
I am thread1,ID: -1208022160
I am thread2,ID: -1218512016

Breakpoint 1, 0x0804853a in pthread_run1 ()
(gdb) bt
#0  0x0804853a in pthread_run1 ()
#1  0x008f1b39 in start_thread () from /lib/libpthread.so.0
#2  0x00834d6e in clone () from /lib/libc.so.6
```
https://blog.csdn.net/zhangye3017

总结调试多线程的命令

命令	用法
info threads	显示当前可调试的所有线程，每个线程会有一个GDB为其分配的ID，后面操作线程的时候会用到这个ID。 前面有*的是当前调试的线程
thread ID(1,2,3…)	切换当前调试的线程为指定ID的线程
break thread_test.c:123 thread all（例：在相应函数的位置设置断点break pthread_run1）	在所有线程中相应的行上设置断点
thread apply ID1 ID2 command	让一个或者多个线程执行GDB命令command
thread apply all command	让所有被调试线程执行GDB命令command
set scheduler-locking 选项 command	设置线程是以什么方式来执行命令

命令	用法
set scheduler-locking off	不锁定任何线程，也就是所有线程都执行，这是默认值
set scheduler-locking on	只有当前被调试程序会执行
set scheduler-locking on step	在单步的时候，除了next过一个函数的情况(熟悉情况的人可能知道，这其实是一个设置断点然后continue的行为)以外，只有当前线程会执行

文章最后发布于: 2018-05-20 15:56:47