

线程的查看以及利用gdb调试多线程

原创

so_u

2018-05-20 15:56:47

52727

★ 收藏 202

版权

分类专栏: linux

[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:

```
1  #include <stdio.h>
2  #include <unistd.h>
3  #include <pthread.h>
4  #include <stdlib.h>
5  #include <string.h>
6
7  void* pthread_run1(void* arg)
8  {
9      (void)arg;
10
11     while(1)
12     {
13         printf("I am thread1,ID: %d\n",pthread_self());
14         sleep(1);
15     }
16 }
17
18 void* pthread_run2(void* arg)
19 {
20     (void)arg;
21
22     while(1)
23     {
24         printf("I am thread2,ID: %d\n",pthread_self());
25         sleep(1);
26     }
27 }
28
29
30 int main()
31 {
32
33     pthread_t tid1;
34     pthread_t tid2;
35
36     pthread_create(&tid1,NULL,pthread_run1,NULL);
37     pthread_create(&tid2,NULL,pthread_run2,NULL);
38
39     printf("I am main thread\n");
40
41     pthread_join(tid1,NULL);
42     pthread_join(tid2,NULL);
43     return 0;
44 }
```

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:



点赞56



评论6



分享



收藏202



打赏



举报

关注

一键三连

- 1 // 查看当前运行的进程
- 2 ps aux|grep a.out
- 3 // 查看当前运行的轻量级进程
- 4 ps -al|grep a.out
- 5 // 查看主线程和新线程的关系
- 6 pstree -p 主线程id

```
[so@localhost 线程]$ ps aux|grep 'test1'
so 1938 0.0 0.0 22596 528 pts/0 S1+ 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)
```

https://blog.csdn.net/zhangye3017

2. 线程栈结构的查看

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

```
[so@localhost 线程]$ pstack 4400
Thread 3 (Thread 0xb77bab70 (LWP 4401)):
#0 0xb04dd424 in __kernel_vsyscall ()
#1 0xb007f396 in nanosleep () from /lib/libc.so.6
#2 0xb007f37c in sleep () from /lib/libc.so.6
#3 0xb004855c in pthread_run1 ()
#4 0xb008f1b39 in start_thread () from /lib/libpthread.so.0
#5 0xb00834d6 in clone () from /lib/libc.so.6
Thread 2 (Thread 0xb6db9b70 (LWP 4402)):
#0 0xb04dd424 in __kernel_vsyscall ()
#1 0xb007f396 in nanosleep () from /lib/libc.so.6
#2 0xb007f37c in sleep () from /lib/libc.so.6
#3 0xb0048586 in pthread_run2 ()
#4 0xb008f1b39 in start_thread () from /lib/libpthread.so.0
#5 0xb00834d6 in clone () from /lib/libc.so.6
Thread 1 (Thread 0xb77b6b70 (LWP 4400)):
#0 0xb04dd424 in __kernel_vsyscall ()
#1 0xb008f21fd in pthread_join () from /lib/libpthread.so.0
#2 0xb00485f9 in main ()
[so@localhost 线程]$ pstack 4401
Thread 1 (process 4401):
#0 0xb04dd424 in __kernel_vsyscall ()
#1 0xb007f396 in nanosleep () from /lib/libc.so.6
#2 0xb007f37c in sleep () from /lib/libc.so.6
#3 0xb004855c in pthread_run1 ()
#4 0xb008f1b39 in start_thread () from /lib/libpthread.so.0
#5 0xb00834d6 in clone () from /lib/libc.so.6
[so@localhost 线程]$ pstack 4402
Thread 1 (process 4402):
#0 0xb04dd424 in __kernel_vsyscall ()
#1 0xb007f396 in nanosleep () from /lib/libc.so.6
```

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)
```

进程间关系

```
[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]
```

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 //1.查看进程: info inferiors
- 2 //2.查看线程: info threads
- 3 //3.查看线程栈结构: bt
- 4 //4.切换线程: thread n (n代表第几个线程)

```
(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 0x0080485f9 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 0x008048586 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 0x00804855c 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线程就是主线程

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 //1. 设置断点: break 行号/函数名
- 2 //2. 查看断点: info b

```
(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 将线程切换到2号线程
[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 0x804853a <pthread_run1+6>
(gdb) 
```

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

```
(gdb) thread apply 2 n 让线程2继续执行自己的代码
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 0x804853a <pthread_run1+6>
(gdb) r 重新启动程序，再次运行到断点处
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, 0x804853a in pthread_run1
(gdb) 
```

3. 只运行当前线程

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

```
(gdb) set scheduler-locking on 设置只执行当前线程函数
(gdb) n
Single stepping until exit from function pthread_run1,
which has no line number information.
I am thread1, ID: -1208022160
(gdb) 
```

4. 所有线程并发执行

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

```
(gdb) set scheduler-locking off  <=> 所有线程并发执行
(gdb) cont  <=> 让其继续运行
Continuing.
I am thread1, ID: -1208022160
I am main thread
I am thread2, ID: -1218512016

Breakpoint 1, 0x0004853a 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, 0x0004853a in pthread_run1 ()
(gdb) bt
#0  0x0004853a in pthread_run1 ()
#1  0x000f1b39 in start_thread () from /lib/libpthread.so.0
#2  0x00034d6e in clone () from /lib/libc.so.6
```

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
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

多线程——多线程debug调试（非常非常详细的... qq_29235677的博客 2万+
在日常开发中我们经常会遇到多线程Debug调试，一般我们都是利用Spring Boot对...

gdb 查看所有线程的栈 maikerdai的专栏 3376
thr app all bt

优质评论可以帮助作者获得更高权重

评论

pthread_life: 非常感谢，看完很有收获，gdb多线程可能这是最好的教程

4 月前 回复

码农 姚泡泡: 谢谢总结!!!

5 月前 回复

我是小x: pthread_self() 不太适合用%d 打印哦，这样容易打印出负数，建议%lu

6 月前 回复

- 圆形的猫： 这个讲得nice啊，在做操作系统的实验，多线程bug找不到，收藏了 10 月前 回复 ...

1
- SO_U 博主 回复： 哈哈，谢谢啦 9 月前 回复 ...

相关推荐

- Linux gdb调试多线程_雪碧柠七的博客

3-31

照例我们先编译代码,一定要记得加-g选项,与多进程不同的是,多线程编译时要加-lpthr...
- 笔记:gdb调试多线程_小心眼儿猫的博客

3-12

在多线程编程时,当我们需要调试时,有时需要控制某些线程停在断点,有些线程继续执...
- Linux下用GDB调试多线程程序_leap的博客_gdb调试多线程...

3-12

我们都知道GDB是Linux下面一款强大程序调试的工具,以前我们都是用GDB来调试单...
- gdb调试多进程和多线程命令

高司机的专栏 5万+

1. 默认设置下,在调试多进程程序时GDB只会调试主进程。但是GDB (>V7.0) 支持...
- gdb调试多线程多进程程序

xy913741894的博客 502

gdb的简介和功能: gdb是GNU开发的一个在Unix, Linux上使用的C/C++和汇编语言...
- GDB多线程如何调试

一只青木呀 212

gdb多进程调试 下面以thread.c为例 #include<stdio.h> #include<pthread.h> void* thr...
- 如何使用GDB调试多线程_krysen的博客

3-1

如何使用GDB调试多线程 先写一段多线程程序。 makefile 加上-g参数生成可调试信...
- gdb 多线程调试

无与伦比BLOG 5596

转载地址: http://blog.csdn.net/kangroger/article/details/47986197 gdb与多线程 在...
- gdb调试多线程

微风 1万+

下文参考以下链接: http://www.cnblogs.com/xuxm2007/archive/2011/04/01/200216...
- gdb查看线程堆栈信息

weixin_34344677的博客 2221

查看堆栈: gdb -quiet -batch -ex='thread apply all bt' -p pid查看运行位置: gdb -quiet...
- gdb调试多进程与多线程

snow_5288的博客 1万+

一, gdb的基础知识1>介绍: gdb是Linux环境下的代码调试工具。2>使用: 需要在...
- 多线程调试(gdb命令行和使用集成开发qtcreator查...

沧海一帆的专栏 4134

我在qtcreator中用调用的gdb找不到如何显示线程号对应的线程名字,因此需要...
- linux gdb-多线程调试

飞翔de刺猬 1万+

linux下应用程序的调试工具主要就是gdb,可能你已经习惯了IDE形式的调试工具。...
- gdb 调试多线程

sleep技术讨论区 7252

设置core环境uname -a 查看机器参数ulimit -a 查看默认参数ulimit -c 1024 设置core...
- GDB 多线程调试:只停止断点的线程,其他线...

weixin_30590285的博客 1008

多线程调试之痛 调试器 (如VS2008和老版GDB) 往往只支持all-stop模式, 调试多线...
- GDB 多线程调试

飞翔de刺猬 950

gdb 多线程调试 http://hi.baidu.com/hcq11/blog/item/9f5bfc6e696209d680cb4a25.ht...
- GDB多线程多进程调试

yiranyaoqiu的专栏 747

gdb多线程调试gdb提供的多线程调试工具 新线程创建自动提醒 thread thread-id实现...
- 简单的gdb调试多线程

cherrydreamsover的博客 1539

利用gdb调试多线程有以下命令: 1.info threads: 显示可以调试的所有线程。gdb会...
- 线程的查看以及利用gdb调试多线程

kongxian2007的专栏 2319

gdb调试常用命令, attach, br, n, bt 在Oceanbase调试中,必须用到gdb,将常用...
- Linux下多线程调试以及查看信息

u014426028的博客 187

pstack pstack用来跟踪进程栈 这个命令在排查进程问题时非常有用 比如我们发现...



so_u

码龄4年 暂无认证

121

14万+

3万+

17万+



原创

周排名

总排名

访问

等级

2623

160

168

34

561

积分

粉丝

获赞

评论

收藏

私信

关注

搜博文文章



热门文章

线程的查看以及利用gdb调试多线程 52709

C++异常捕获和处理 19770

几种算法----n的阶乘 16695

linux：线程的创建、线程等待、线程终止、线程分离 7788

求平均数的几种方法 6780

分类专栏

 c语言 30篇

 数据结构 22篇

 剑指offer面试题 23篇

 练习 4篇

 linux 24篇

 c++ 20篇

最新评论

C 清空输入缓冲区，以及flush（stdin） ...

huang_t: 这个函数的问题我可以用其他方法解决 只是很好奇怎么去真正地清空键: ...

C 清空输入缓冲区，以及flush（stdin） ...

huang_t: 在网上查到的清空缓存区都不能我解决缓冲区啊 求大佬帮我看这是: ...

模拟实现shell----输出重定向

顾影生翔: #include <fcntl.h>

Google Test源码浅析(三) ----- RUN_A...

jixiekuangzhe: 博主，想问下run_all_test，能够在main中创建3个线程进行调用，但 ...

天若有情天亦老°: 请问O_WRONLY和O_RDONLY报错没有定义怎么搞

最新文章

软件测试知识点汇总

Google Test源码浅析(三) -----
RUN_ALL_TESTS

Google Test源码浅析(二) ----- TEST宏

2018年 90篇 2017年 39篇