# strace命令使用分析

**qq: 2391387647**

**date: 2020-6-28**

**Author: 邹卓恩**

目录

[strace命令使用分析 1](#_Toc44257285)

[第一章 strace的介绍与用法 2](#_Toc44257286)

[1.1 strace概述 2](#_Toc44257287)

[1.2 strace用法介绍 3](#_Toc44257288)

[1.2.1 strace -p pid 3](#_Toc44257289)

[1.2.2 strace -i 4](#_Toc44257290)

[1.2.3 strace -t 5](#_Toc44257291)

[1.2.4 strace -T 6](#_Toc44257292)

[1.2.5 strace -yy 6](#_Toc44257293)

[1.2.6 strace -c 7](#_Toc44257294)

[1.2.7 strace -f 8](#_Toc44257295)

[1.2.8 strace -e 8](#_Toc44257296)

[1.3 案例：结合strace分析线程间死锁 11](#_Toc44257297)

## strace的介绍与用法

1.1 strace概述

在操作系统中，进程分为用户态进程和内核态进程，应用程序运行在用户态，内核态负责对资源包括网络，磁盘，内存等管理，用户态进程要访问这些资源时，需要通过系统调用陷入内核态，由内核来统一管理和分配。

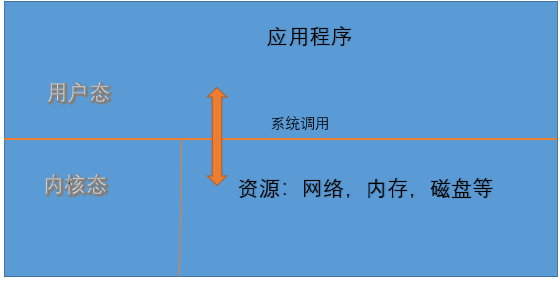


图1

通过系统调用陷入内核态，或者应用程序运行过程中，产生的signal，都可以被strace捕获到,每个系统调用的名称，参数，返回值，调用时间通过标准错误输出或者-O选项写入到文件中，通过分析这些信息，可以在应用程序发生故障时，不用重新编译程序，就可以捕获当前进程内部正在做什么。

1.2 strace用法介绍

下面对strace各个参数的使用方法进行介绍。

1.2.1 strace -p pid

跟踪正在执行的进程，正在执行的系统调用，先来熟悉几个查找进程的命令，pidof 查看当前进行运行程序的进程pid，pstree，跟踪当前进程pid查看，进程树关系。

例子：

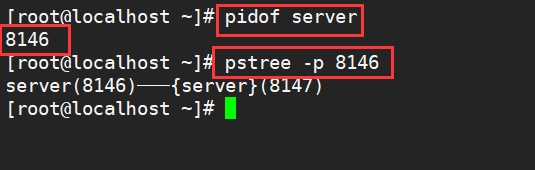


图2

由图可知，server进程的pid为8146，其中8146创建了线程8147,通过pidof和pstree可以查看进程pid和pstree查看进程树关系。

现在使用strace -p 8147跟踪这个进程当前在进行哪些系统调用。

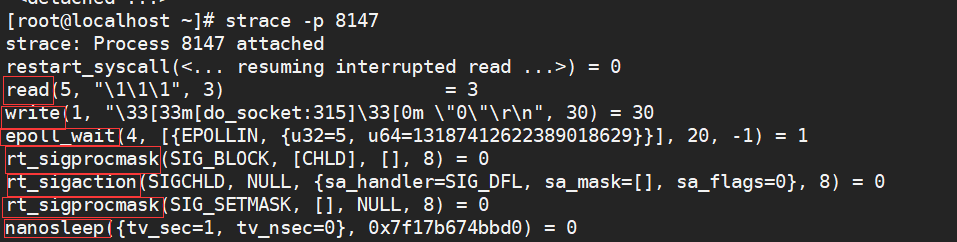


图3

从图3可知，进程8147进在执行的系统调用是read,write,epoll\_wait等,还有这些系统调用的参数，和返回值，从而大概知道进程8147正在执行的情况。

1.2.2 strace -i

现在在原来-p的基础上，加上-i参数，可以查看系统调用的入口指针。

通过（1.2.1）步骤，重新查看server的pid，并用strace -p pid -i跟踪系统调用并查看系统调用入口指针。

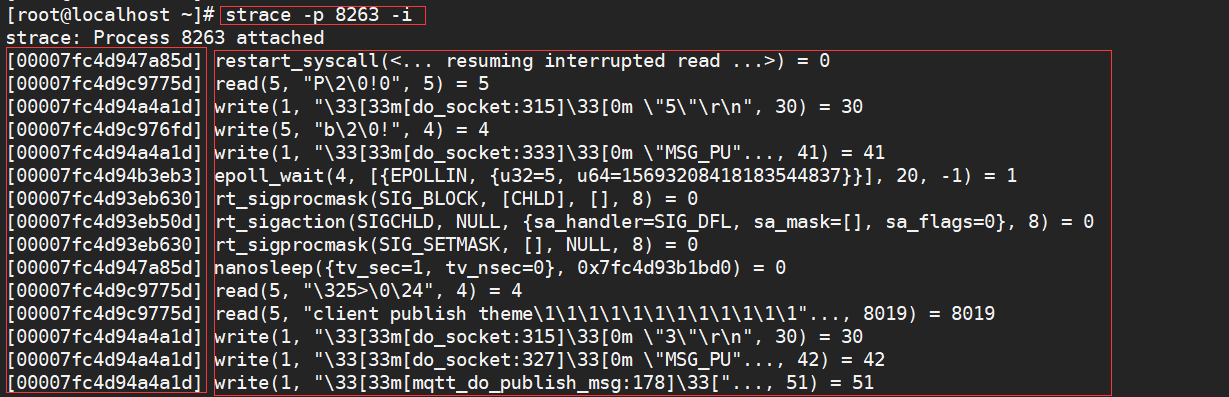


图4

其中最左侧部分就是系统调用入口指针。

1.2.3 strace -t

在原来-p 的基础上加上-t，跟踪系统调用，同时查看系统调用的时间戳。结合步（1.2.1）使用strace -p pid -t 跟踪系统调用，同时查看。

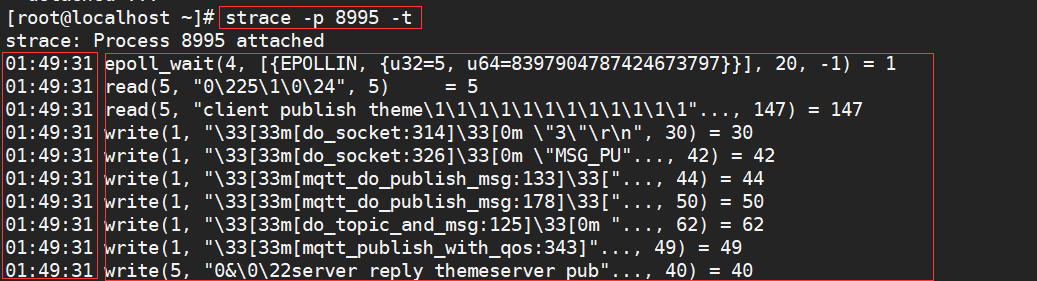


图5

其中最左侧的是系统调用的时间戳。

其中strace -t 是精确度是时分秒，-tt是精确为s。

用法：strace -p pid -tt

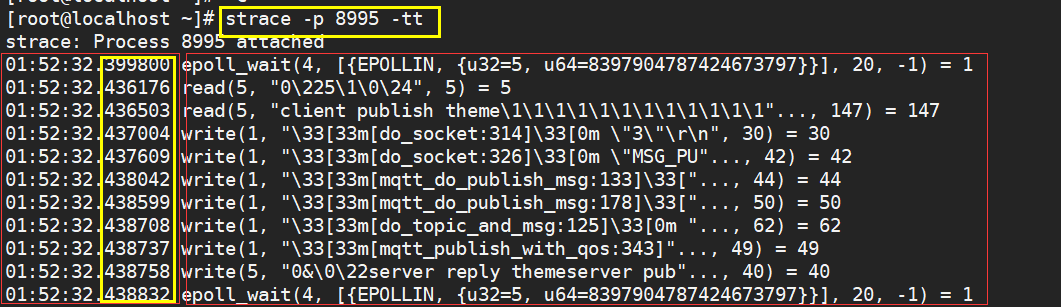


图6

1.2.4 strace -T

用于查看系统调用中，每个系统调用花费的时间。

在1.2.1的基础上，加上-T选项，strace -p pid -T。

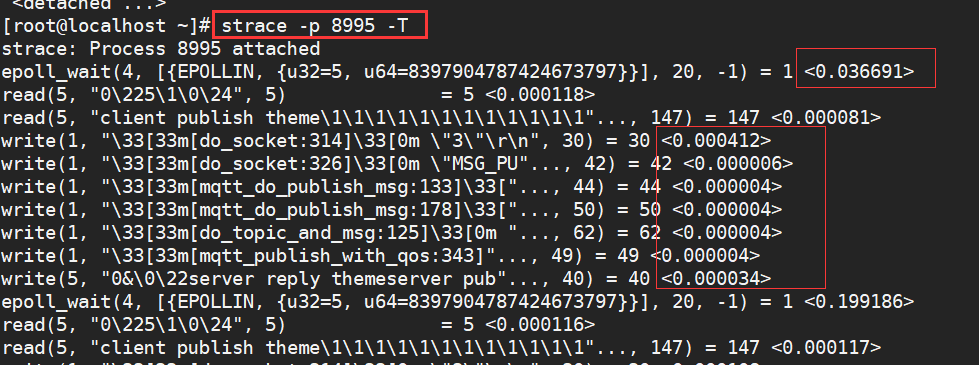


图7

如图7所示，其中最右侧部分是每个系统调用花费的时间。

1.2.5 strace -yy

用于查看，进行tcp/udp read/write时，查看socket的五元组详细信息（比如ip，port，tcp）。

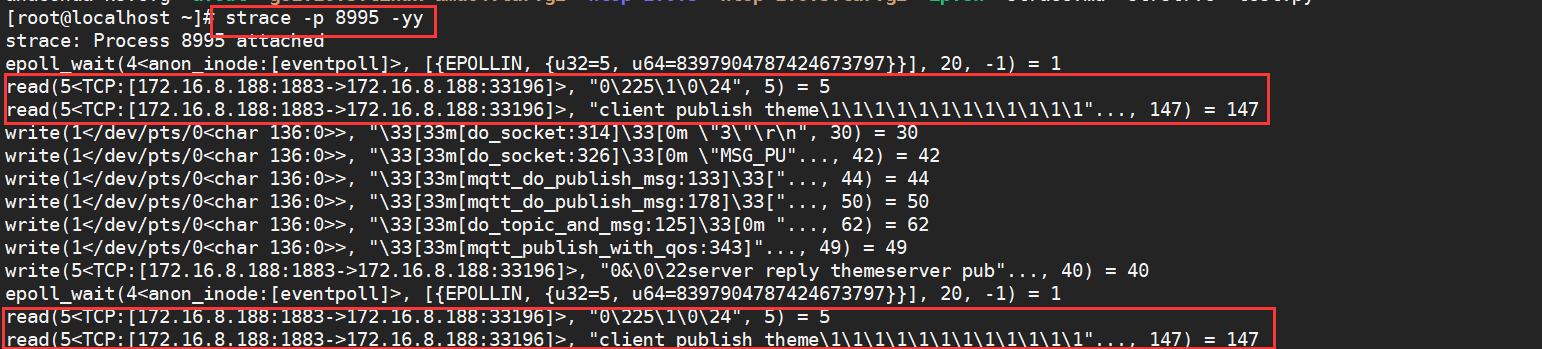


图8

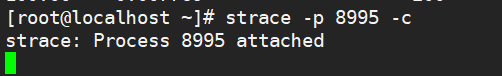
如上图8所示，加上-yy参数之后，可以看到socket文件描述符5对应的五元组信息为：

TCP:[172.16.8.188:1883->172.16.8.188:33196]。在一些服务端与客户端的交互中可以用来分析排查问题。

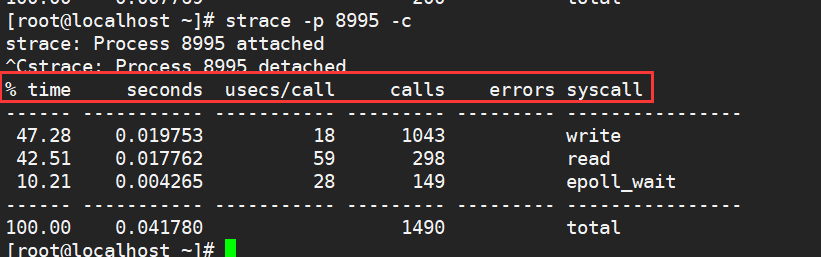
1.2.6 strace -c

用于统计每个系统调用，时间，调用次数，错误次数等。

用法：strace -p pid -c



运行一段时间后，ctrl+c终止

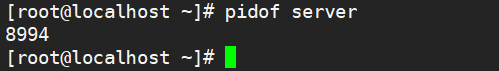


有颜色部分，从左到右，分别是运行时间面分比，运行时间，每次调用花费的时间（单位usecs）,调用总次数，调用出错次数，对应的API。

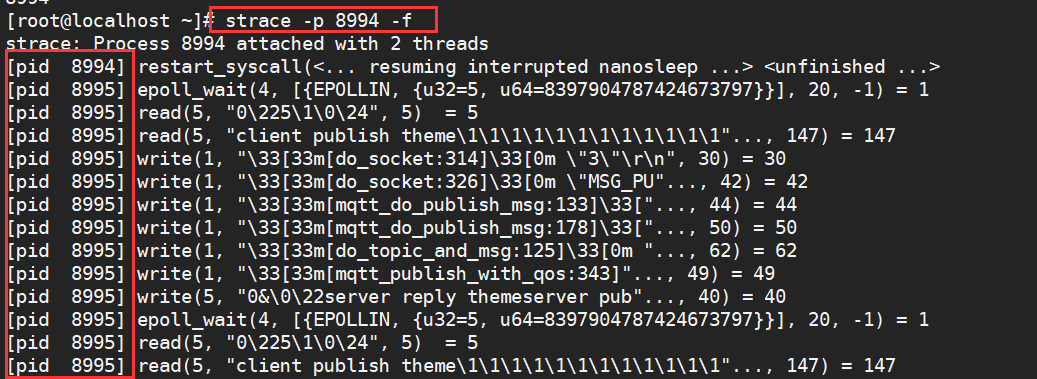
1.2.7 strace -f

跟踪系统调用的同时，同时打印pid；如果是父进程，则同时跟踪打印子进程的系统调用。

查看server程序的父进程pid。



strace -p pid -f



加上-参数之后，其中最左侧可以显示整个应用程序父进程（包括父进程）下的所有子进程的系统调用。

1.2.8 strace -e

前面介绍strace用法，一次性打印所有系统调用，现在加上-e参数之后，可以过滤掉一些系统调用，查看自己想要查看的系统调用，比如只跟踪网络，ipc通信或者只跟踪某个API比如read相关的系统调用。

1. **-e trace=desc**

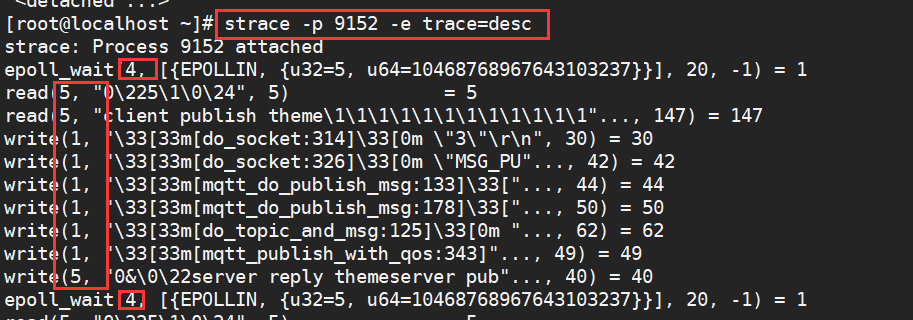
跟踪和文件描述符相关API，比如write/read/select/epoll等

代码：

code\_1是server和client通信的例程，

常试用-e trace=desc，查看看server内部进行的系统调用。

分别编译和运行server和client。



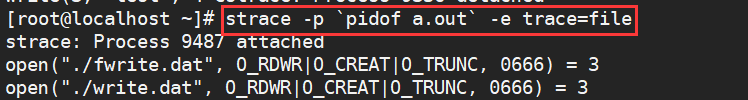
通过图，可以查看到5，1，4，相关文件描述符相关的系统调用API。

1. **-e trace=file**

跟踪和文件访问相关的调用(参数中有文件名)。

运行实例：

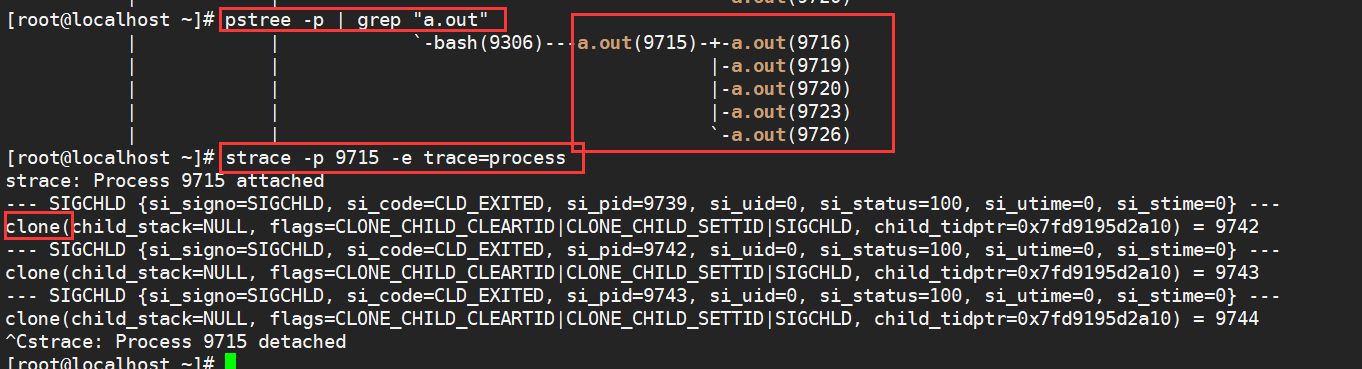
在后台用strace跟踪



1. **-e trace=process和进程管理相关的调用，比如fork/exec/exit\_group**

实例代码：

运行进程，并用pstree -p查看进程树关系，找到父进程，并用strace跟踪。

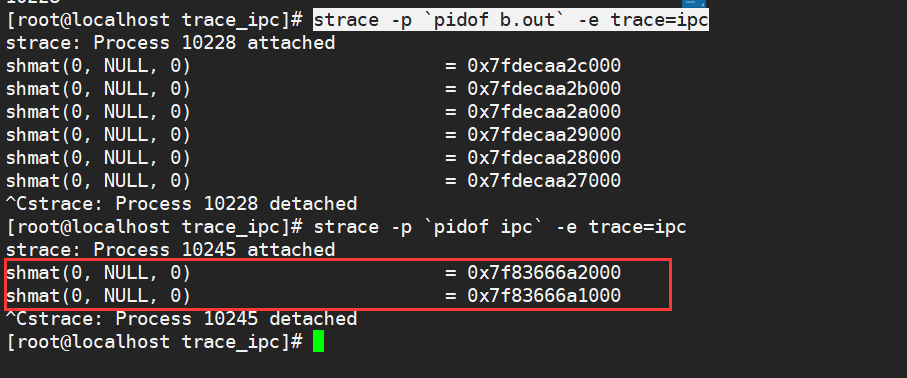


1. **-e trace=ipc 进程间通信相关，比如shmget等**

实例代码：

运行此实例代码，并另外打开一个窗口，进行跟踪

strace -p `pidof b.out` -e trace=ipc



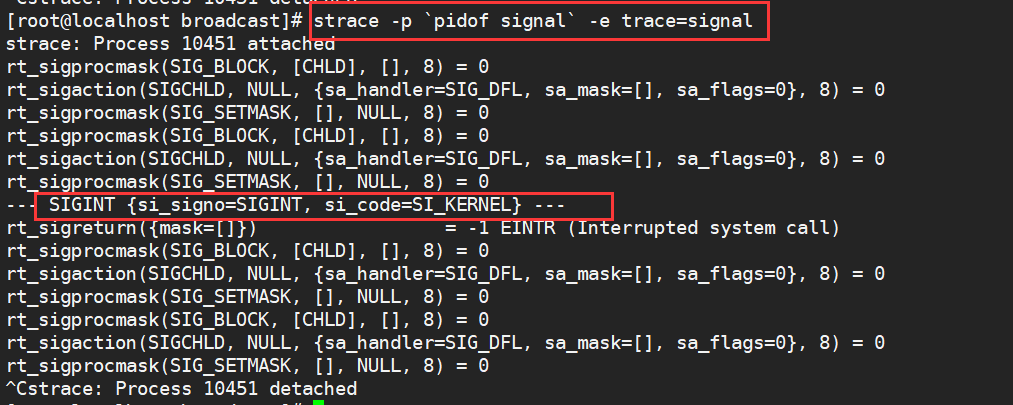
从图中可以看到，shmat相关的ipc调用。

1. **-e trace=signal 信号发送和处理相关，比如kill/sigaction**

实例代码：

运行相关的实现代码进程，并执行以下命令

strace -p `pidof signal` -e trace=signal

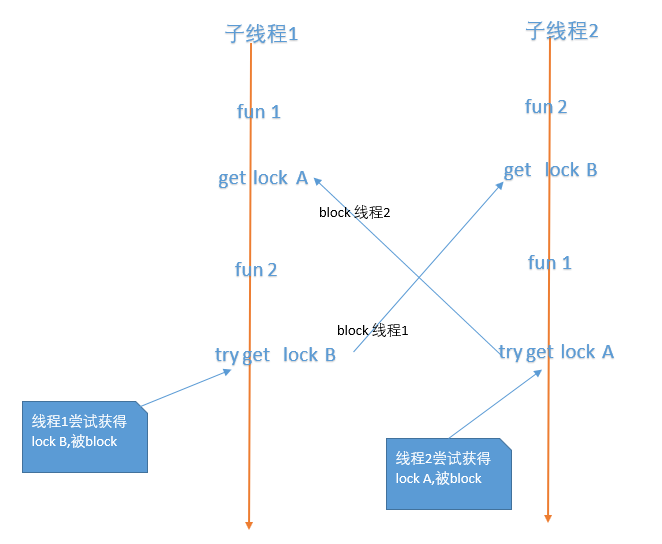


1.3 案例：结合strace分析线程间死锁问题

两个或者多个进程/线程竞争共享资源导致互相等待的现象等待的线程无法获得资源进而往下执行。

常见死锁的情形:

线程1获得锁A，等待锁B，线程2获得锁B，等待锁A。



产生死锁的四个条件

1. 互斥条件，一个资源只能被一个进程/线程使用
2. 请求与保护条件，一个进程/线程请求资源而阻塞，对此保持不放
3. 不剥夺条件，一个进程/线程获得共享资源，如果没有使用完，不能被剥夺
4. 循环等待条件，多个进程/线程形成头尾先接的等待

避免死锁的方法

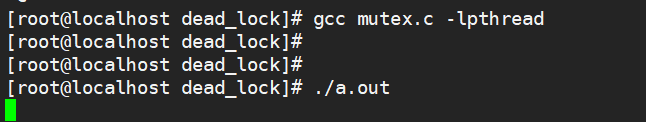
1. 银行家算法
2. 有序资源分配法
3. 锁的范围小
4. 锁的临界区内代码执行时间短

结合strace分析多线程之间产生死锁

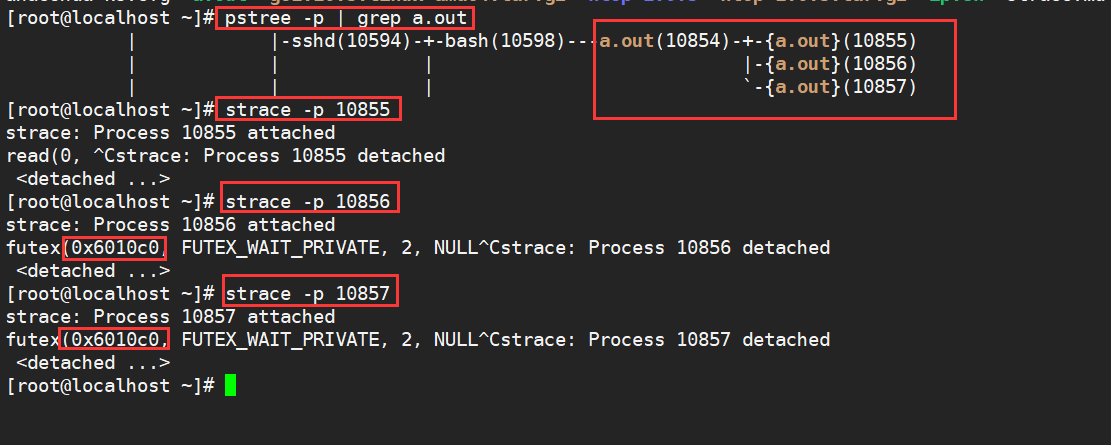
实例代码：



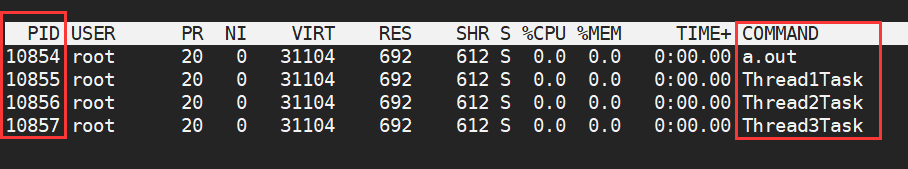
编程运行实例代码：



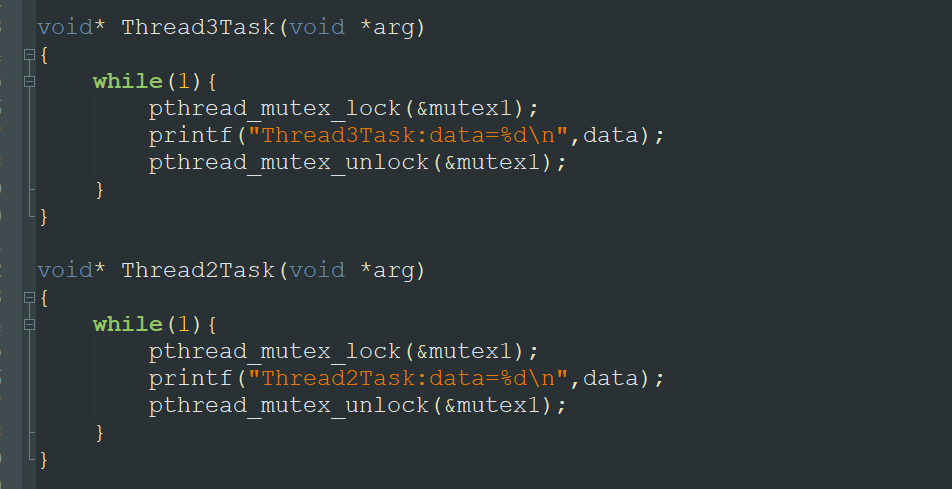
使用pstree -p | grep a.out，查看程序的进程树关系和pid，并用strace跟踪子线程pid，可以看出10856线程和10857线程被阻塞在同一个锁0x6010c0，初步定为这两个线程竞争同一个锁导致死锁。



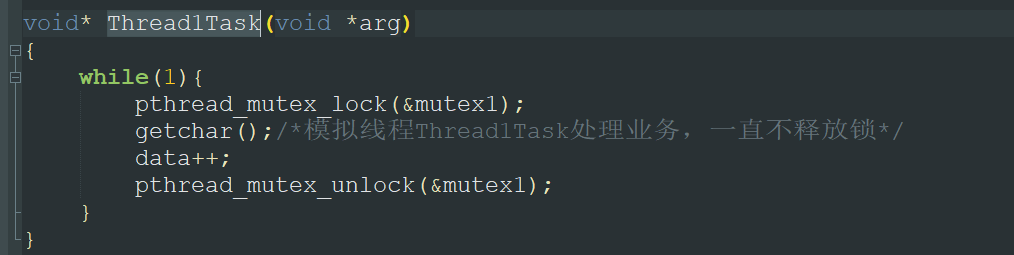
使用top -p ppid(父进程pid) -H,查看这些死锁的pid对应的线程是哪些线程，这个需要提前在代码中设置线程名：如：pthread\_setname\_np(t2, "Thread2Task");



从上图可以看出，线程对应的线程函数，进一步断定，Thread2Task和Thread3Task发生了死锁。



从代码可以看出，Thread2Task和Thread3Task竞争同一个锁。



Thread1Task线程一直在处理业务，没有释放锁，导致死锁的发生。

其它分析死锁的工具：

gdb，pstack，通过这些工具，可能查看所有线程的调用栈，从而判断线程具体在做些什么，如果是发生了死锁，则最后一次执行，是在一个锁上，但是，在一些高度裁剪的嵌入式设备上，这些工具不一定有。