

实验三

实验三

题目3：死锁分析

实验目的

实验内容

实验设计原理

实验步骤

实验结果及分析

情况一

情况二

情况三

程序代码

题目3：死锁分析

实验目的

编程实现多个并发进程/线程：进程/线程间采用锁机制，竞争使用临界资源导致死锁——由此分析死锁发生的原因。

实验内容

编程实现死锁发生的3种情况：持续占用锁导致其他线程等待、同线程多次锁操作、双线程交叉使用互斥锁，观察在gdb调试下的进程信息分析死锁产生的具体过程。

实验设计原理

死锁问题大概可以分为3种情况：

1. 主线程锁不能及时释放导致别的线程卡住，这类不算真正的死锁。
2. 同一个线程锁多次导致死锁。
3. 两个锁交叉使用导致死锁。

这三种情况的定位方法都是一样，都是根据锁的占有者顺藤摸瓜：比如线程1在等锁，必然有别的线程在占有锁，可以查看线程1所需求的锁的占有者找到线程2，以此类推，直到把锁的关系理清楚就知道死锁原因了。

实验步骤

- 将第一种情况占有导致锁等待封装于函数test1中，其依次调用线程1和线程2：前者获得互斥锁后进入无限循环而不再释放锁，后者为正常得到并释放锁的简单线程。
 - 将第二种情况同线程多次锁操作封装于函数test3中，其调用线程5：其得到互斥锁后执行fun5再次请求锁，造成单线程中的自锁。
 - 将第三种情况双线程交叉求锁封装于函数test2中，其依次调用线程3和线程4：前者得到锁0后进行一段休眠，之后执行fun3请求互斥锁1并释放最终释放初始锁0；后者得到锁1后执行fun4请求初始锁0并释放最终释放锁1。
- 编码并编译完成后分别执行各个test并使用gdb调试进行分析。

实验结果及分析

情况一

实验五 多线程编程

云产品资源

实验报告

子用户名: u-mdhwep5u@1644771854403952

子用户密码: Zg2Wd2Qn8Zf1Uv7M

AK ID: LTAI5HYZCZUmzmVP8P7HVS

AK Secret: GCtqgQDSiK7Z2XMt6iZaBIMid...

注意: 若登录子账号, 需打开控制台进行登录。

一键复制子账号登录链接

ECS云服务器

磁盘ID: d-uf63q2kl6ntv41k0i0

ECS公网地址: 47.103.99.179

ECS登录名: root

登录密码: lr0la1Fv7C

ECS实例ID: i-uf63q2kl6ntv41x8gj

IP白名单: 0.0.0.0/0

地域: 华东 2 (上海)

2. root@iZuf63q2kl6ntv41x8gj: ~

include <semaphore.h>

pthread_mutex_t g_mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t g_mutex1 = PTHREAD_MUTEX_INITIALIZER;

void *threadProc1(void *)

{

printf("threadProc1 enter = %d\n", getpid());

pthread_mutex_lock(&g_mutex);

while(1);

pthread_mutex_unlock(&g_mutex);

printf("threadProc1 unlock pid = %d\n", getpid());

}

void *threadProc2(void *)

{

printf("threadProc2 enter = %d\n", getpid());

pthread_mutex_lock(&g_mutex);

pthread_mutex_unlock(&g_mutex);

printf("threadProc2 unlock pid = %d\n", getpid());

}

void fun3()

实验五 多线程编程

云产品资源

实验报告

子用户名: u-mdhwep5u@1644771854403952

子用户密码: Zg2Wd2Qn8Zf1Uv7M

AK ID: LTAI5HYZCZUmzmVP8P7HVS

AK Secret: GCtqgQDSiK7Z2XMt6iZaBIMid...

注意: 若登录子账号, 需打开控制台进行登录。

一键复制子账号登录链接

ECS云服务器

磁盘ID: d-uf63q2kl6ntv41k0i0

ECS公网地址: 47.103.99.179

ECS登录名: root

登录密码: lr0la1Fv7C

ECS实例ID: i-uf63q2kl6ntv41x8gj

IP白名单: 0.0.0.0/0

地域: 华东 2 (上海)

2. root@iZuf63q2kl6ntv41x8gj: ~

Is this ok [y/d/h]: y

Downloading packages:

(1/2): libstdc++-devel-4.8.5-44.el7.x86_64.rpm | 1.5 MB 00:00:00

(2/2): gcc-c++-4.8.5-44.el7.x86_64.rpm | 7.2 MB 00:00:00

Total | 41 MB/s | 8.7 MB 00:00:00

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Installing : libstdc++-devel-4.8.5-44.el7.x86_64 | 1/2

Installing : gcc-c++-4.8.5-44.el7.x86_64 | 2/2

Verifying : gcc-c++-4.8.5-44.el7.x86_64 | 1/2

Verifying : libstdc++-devel-4.8.5-44.el7.x86_64 | 2/2

Installed:

gcc-c++-x86_64 0:4.8.5-44.el7

Dependency Installed:

libstdc++-devel.x86_64 0:4.8.5-44.el7

Complete!

[root@iZuf63q2kl6ntv41x8gj: ~]# g++ -o deadlock deadlock.cpp -lpthread -g

/usr/lib/gcc/x86_64-redhat-linux/4.8.5/../../../../lib64/crt1.o: In function '_start':

(.text+0x20): undefined reference to 'main'

collect2: error: ld returned 1 exit status

[root@iZuf63q2kl6ntv41x8gj: ~]# vim deadlock.cpp

[root@iZuf63q2kl6ntv41x8gj: ~]# vim deadlock.cpp

[root@iZuf63q2kl6ntv41x8gj: ~]# ./deadlock

-bash: ./deadlock: command not found

[root@iZuf63q2kl6ntv41x8gj: ~]# g++ -o deadlock deadlock.cpp -lpthread -g

[root@iZuf63q2kl6ntv41x8gj: ~]# ls

deadlock

[root@iZuf63q2kl6ntv41x8gj: ~]# ./deadlock

test1 begin

test1 end

threadProc1 enter = 11697

threadProc2 enter = 11697

[root@iZuf63q2kl6ntv41x8gj: ~]#

执行deadlock后根据输出观察到线程1和线程2均进入阻塞状态。结合代码可以推测线程1阻塞的原因是本身代码的死循环，同时占有了g_mutex；而线程2进入的时候需要等待g_mutex可用，而结果就是线程2一直阻塞在等待状态。通过gdb调试信息如下：

实验五 多核多线程编程 详情

实验手册 云产品资源 实验报告

实验云账号，创建成功后生成 收起

子用户名: u-mdhwp5u@1644771854403952

子用户密码: Zg2Wd2Qn821Uv7M

AK ID: IA15H3CZUmZmVtP87HVSI

AK Secret: GCEqqQDSjkM7Z2XM6JzaBfMid...

注意: 若登录子账号，请打开密钥私钥门进行登录。

一键复制子账号登录链接

ECS服务器

磁盘ID: d-uf63q2kl6ntv41k0i0

ECS公网地址: 47.103.99.179

ECS登录名: root

登录密码: l0la1Fv7C

ECS实例ID: i-uf63q2kl6ntv41x8gj

IP名称: 0.0.0.0/0

地域: 华东2 (上海)

剩余体验时间 01:44:44 结束体验

> . root@iZuf63q2kl6ntv41x8gjZ ~#

Installed:

gdb.x86_64 0:7.6.1-120.el7

Complete!

[root@iZuf63q2kl6ntv41x8gjZ ~]# gdb ./deadlock

GNU gdb (GDB) Red Hat Enterprise Linux 7.6.1-120.el7

Copyright (C) 2013 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 "x86_64-redhat-linux-gnu".

For bug reporting instructions, please see:

<http://www.gnu.org/software/gdb/bugs/>.

Reading symbols from /root/.deadlock...done.

(gdb) run

Starting program: /root/.deadlock

[Thread debugging using libthread_db enabled]

Using host libthread_db library "/lib64/libthread_db.so.1".

test1 begin

(New Thread 0x7ffff6d0700 (LWP 11839))

threadProc1 enter = 11830

[New Thread 0x7ffff6cf700 (LWP 11840)]

test1 end

threadProc2 enter = 11830

^C

Program received signal SIGINT, Interrupt.

0x00007ffff70969fd in nanosleep () from /lib64/libc.so.6

Missing separate debuginfos, use: debuginfo-install glibc-2.17-325.el7_9.x86_64 libgcc-4.8.5-44.el7.x86_64 libstdc++-4.8.5-44.el7.x86_64

Id thread

Id target Id frame

3 Thread 0x7ffff6cf700 (LWP 11840) "deadlock" 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

2 Thread 0x7ffff6d0700 (LWP 11839) "deadlock" threadProc1 () at deadlock.cpg:21

* 1 Thread 0x7ffff7f0740 (LWP 11830) "deadlock" 0x00007ffff70969fd in nanosleep () from /lib64/libc.so.6

(gdb) t 3

[Switching to thread 3 (Thread 0x7ffff6cf700 (LWP 11840))]

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

(gdb)

default +

14 of 39.7 30 Rows 132 Cols on 11K1518.1

通过g_mutex信息可以观察到其占有者为11839，而堆栈显示序号为3的线程此时进入了lock_wait状态：线程11840，也就是上文中的线程2（图中序号为3）在持续等锁，然而其已经被线程11839（上文中的线程1，图中序号为2）占用，后者持续死循环而g_mutex不会被主动释放，在这段时间线程11840就在持续等待造成死锁现象。

情况二

由g_mutex信息可观察到此时互斥锁的占有者就是唯一的线程12060本身，然而在fun5代码中其索求更多的同一互斥锁，而这是无法达成的，故引起单一线程自身的死锁。

情况三

实验五 多核多线程编程

剩余体验时间: 01:23:16

结束体验

实验手册

云产品资源

实验报告

体验云账号, 创建资源后生成

收起

子用户名: u-mdhwp5u@1644771854403952

子用户密码: Zg2Wd2Qn8Zf1Uv7M

AK ID: LTAI5tHZZCZUmZmVIP8P7HVSI

AK Secret: GCEqgQDSjKM7ZZXMt6jZaBfMid...

注意: 若登录子账号, 需打开临时窗口进行登录。

一键复制子账号登录链接

ECS服务器

磁盘ID: d-uf63q2kl6ntv41k0i0

ECS公网地址: 47.103.99.179

ECS登录名: root

登录密码: Ir0la1Fv7C

ECS实例ID: i-uf63q2kl6ntv41x8gj

IP白名单: 0.0.0.0/0

地域: 华东 2 (上海)

2. root@izuf63q2kl6ntv41x8gj:~

Thread3();

Thread4();

}

void test3()

{

Thread5();

}

int main(int argc, char *argv[])

{

//test1();

test2();

//test3();

while(1)

{

sleep(1);

}

return 0;

}

~

~

:wq

default

命令终端 已连接 华东2(上海) i-uf63q2kl6ntv41x8gj 47.103.99.179 22 sanm2wvz 14 pt 39, 4 39 Rows 132 Cols en_US.UTF-8

实验五 多核多线程编程

剩余体验时间: 01:21:02

结束体验

实验手册

云产品资源

实验报告

体验云账号, 创建资源后生成

收起

子用户名: u-mdhwp5u@1644771854403952

子用户密码: Zg2Wd2Qn8Zf1Uv7M

AK ID: LTAI5tHZZCZUmZmVIP8P7HVSI

AK Secret: GCEqgQDSjKM7ZZXMt6jZaBfMid...

注意: 若登录子账号, 需打开临时窗口进行登录。

一键复制子账号登录链接

ECS服务器

磁盘ID: d-uf63q2kl6ntv41k0i0

ECS公网地址: 47.103.99.179

ECS登录名: root

登录密码: Ir0la1Fv7C

ECS实例ID: i-uf63q2kl6ntv41x8gj

IP白名单: 0.0.0.0/0

地域: 华东 2 (上海)

2. root@izuf63q2kl6ntv41x8gj:~

Program received signal SIGINT, Interrupt.

0x00007ffff70969fd in nanosleep () from /lib64/libc.so.6

Missing separate debuginfos, use: debuginfo-install glibc-2.17-325.el7_9.x86_64 libgcc-4.8.5-44.el7.x86_64 libstdc++-4.8.5-44.el7.x86_64

(gdb) info thread

Id Target Id Frame

2 Thread 0x7ffff6fd0780 (LWP 12060) "deadlock" 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

* 1 Thread 0x7ffff7ff0740 (LWP 12051) "deadlock" 0x00007ffff70969fd in nanosleep () from /lib64/libc.so.6

(gdb) t 2

[Switching to thread 2 (Thread 0x7ffff6fd0780 (LWP 12060))]

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

(gdb) bt

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

#1 0x00007ffff7bc8e9b in __lll_lock_883 () from /lib64/libpthread.so.0

#2 0x00007ffff7bcd6d8 in pthread_mutex_lock () from /lib64/libpthread.so.0

#3 0x0000000000400a8d in fun5 () at deadlock.cpp:119

#4 0x0000000000400a4d in threadProc5 () at deadlock.cpp:135

#5 0x00007ffff7bc6ea5 in start_thread () from /lib64/libpthread.so.0

#6 0x00007ffff70cfb0d in clone () from /lib64/libc.so.6

(gdb) f 2

#2 0x00007ffff7bcd6d8 in pthread_mutex_lock () from /lib64/libpthread.so.0

(gdb) p g_mutex

\$1 = {__data = {__lock = 2, __count = 0, __owner = 12060, __nusers = 1, __kind = 0, __spins = 0, __elision = 0, __list = {__prev = 0x0, __next = 0x0}}, __size = "\002\000\000\000\000\000\000\000\034\000\000\001", '\000' <repeats 26 times>, __align = 2}

(gdb) 72

[4]+ Stopped gdb deadlock

[root@izuf63q2kl6ntv41x8gj:~]# vim deadlock.cpp

[root@izuf63q2kl6ntv41x8gj:~]# g++ -o deadlock deadlock.cpp -lpthread -g

[root@izuf63q2kl6ntv41x8gj:~]# ./deadlock

threadProc4 enter = 12212

threadProc3 enter = 12212

fun4 begin

fun3 begin

^Z

[5]+ Stopped ./deadlock

[root@izuf63q2kl6ntv41x8gj:~]#

default

命令终端 已连接 华东2(上海) i-uf63q2kl6ntv41x8gj 47.103.99.179 22 sanm2wvz 14 pt 30, 35 39 Rows 132 Cols en_US.UTF-8

根据输出可观察到线程3和线程4各自的执行函数均进入了死锁, 在gdb调试信息中显示:

实验五 多核多线程编程

剩余体验时间: 01:19:44

结束体验

实验手册

云产品资源

实验报告

体验云账号, 创建资源后生成

收起

子用户名: u-mdhwp5u@1644771854403952

子用户密码: Zg2Wd2Qn8Zf1Uv7M

AK ID: LTAI5tHZZCZUmZmVIP8P7HVSI

AK Secret: GCEqgQDSjKM7ZZXMt6jZaBfMid...

注意: 若登录子账号, 需打开临时窗口进行登录。

一键复制子账号登录链接

ECS服务器

磁盘ID: d-uf63q2kl6ntv41k0i0

ECS公网地址: 47.103.99.179

ECS登录名: root

登录密码: Ir0la1Fv7C

ECS实例ID: i-uf63q2kl6ntv41x8gj

IP白名单: 0.0.0.0/0

地域: 华东 2 (上海)

2. root@izuf63q2kl6ntv41x8gj:~

[New Thread 0x7ffff6fd0780 (LWP 12271)]

threadProc3 enter = 12267

[New Thread 0x7ffff6cf700 (LWP 12272)]

threadProc4 enter = 12267

fun4 begin

fun3 begin

^C

Program received signal SIGINT, Interrupt.

0x00007ffff70969fd in nanosleep () from /lib64/libc.so.6

Missing separate debuginfos, use: debuginfo-install glibc-2.17-325.el7_9.x86_64 libgcc-4.8.5-44.el7.x86_64 libstdc++-4.8.5-44.el7.x86_64

(gdb) info thread

Id Target Id Frame

3 Thread 0x7ffff6cf700 (LWP 12272) "deadlock" 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

2 Thread 0x7ffff6fd0780 (LWP 12271) "deadlock" 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

* 1 Thread 0x7ffff7ff0740 (LWP 12267) "deadlock" 0x00007ffff70969fd in nanosleep () from /lib64/libc.so.6

(gdb) t 2

[Switching to thread 2 (Thread 0x7ffff6fd0780 (LWP 12271))]

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

(gdb) bt

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

#1 0x00007ffff7bc8e9b in __lll_lock_883 () from /lib64/libpthread.so.0

#2 0x00007ffff7bcd6d8 in pthread_mutex_lock () from /lib64/libpthread.so.0

#3 0x0000000000400981 in fun3 () at deadlock.cpp:57

#4 0x0000000000400a00 in threadProc3 () at deadlock.cpp:89

#5 0x00007ffff7bc6ea5 in start_thread () from /lib64/libpthread.so.0

#6 0x00007ffff70cfb0d in clone () from /lib64/libc.so.6

(gdb) t 3

[Switching to thread 3 (Thread 0x7ffff6cf700 (LWP 12272))]

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

(gdb) bt

#0 0x00007ffff7bcd54d in __lll_lock_wait () from /lib64/libpthread.so.0

#1 0x00007ffff7bc8e9b in __lll_lock_883 () from /lib64/libpthread.so.0

#2 0x00007ffff7bcd6d8 in pthread_mutex_lock () from /lib64/libpthread.so.0

#3 0x00000000004009af in fun4 () at deadlock.cpp:71

#4 0x0000000000400a53 in threadProc4 () at deadlock.cpp:105

#5 0x00007ffff7bc6ea5 in start_thread () from /lib64/libpthread.so.0

#6 0x00007ffff70cfb0d in clone () from /lib64/libc.so.6

(gdb)

default

命令终端 已连接 华东2(上海) i-uf63q2kl6ntv41x8gj 47.103.99.179 22 sanm2wvz 14 pt 39, 7 39 Rows 132 Cols en_US.UTF-8

线程12272及线程12271同时处于lock_wait状态, 而两个线程的堆栈信息所显示的状态基本一致。


```

        pthread_mutex_unlock(&g_mutex1);
        printf("fun3 end\n");
    }

void fun4()
{
    printf("fun4 begin\n");
    pthread_mutex_lock(&g_mutex);
    pthread_mutex_unlock(&g_mutex);
    printf("fun4 end\n");
}

void *threadProc3(void*)
{
    printf("threadProc3 enter = %d\n", getpid());
    pthread_mutex_lock(&g_mutex);
    sleep(1);
    fun3();
    pthread_mutex_unlock(&g_mutex);
    printf("threadProc3 unlock pid = %d\n", getpid());
}

void *threadProc4(void*)
{
    printf("threadProc4 enter = %d\n", getpid());
    pthread_mutex_lock(&g_mutex1);
    fun4();
    pthread_mutex_unlock(&g_mutex1);
    printf("threadProc4 unlock pid = %d\n", getpid());
}

void fun5()
{
    printf("fun5 begin\n");
    pthread_mutex_lock(&g_mutex);
    pthread_mutex_unlock(&g_mutex);
    printf("fun5 end\n");
}

void *threadProc5(void*)
{
    printf("threadProc5 enter = %d\n", getpid());
    pthread_mutex_lock(&g_mutex);
    fun5();
    pthread_mutex_unlock(&g_mutex);
    printf("threadProc5 unlock pid = %d\n", getpid());
}

void Thread1()
{
    pthread_t tid;
    pthread_create(&tid, NULL, threadProc1, NULL);
    pthread_detach(tid);
}

void Thread2()
{
    pthread_t tid;

```



```

        pthread_create(&tid, NULL, threadProc2, NULL);
        pthread_detach(tid);
    }

void Thread3()
{
    pthread_t tid;
    pthread_create(&tid, NULL, threadProc3, NULL);
    pthread_detach(tid);
}

void Thread4()
{
    pthread_t tid;
    pthread_create(&tid, NULL, threadProc4, NULL);
    pthread_detach(tid);
}

void Thread5()
{
    pthread_t tid;
    pthread_create(&tid, NULL, threadProc5, NULL);
    pthread_detach(tid);
}

//回调接口卡主导致其他线程等锁

void test1()
{
    printf("test1 begin\n");
    Thread1();
    Thread2();
    printf("test1 end\n");
}

//两个锁交叉使用导致死锁

void test2()
{
    Thread3();
    Thread4();
}

//同一锁被一个线程锁多次

void test3()
{
    Thread5();
}

int main(int argc, char *argv[])
{
    test1();
    //test2();
    //test3();
    while(1)
    {
        sleep(1);
    }
}

```

```
}  
    return 0;  
}
```