作者：代兴　邮箱：503268771@qq.com　Vx 公众号：东方架构师
https://github.com/drink-cat/Book_Program-Principles

阻塞锁的含义

阻塞锁，核心包括 1 个整数变量、1 个持有锁的线程 ID、1 个重入计数、1 个线程等待队列。
加锁、解锁，使用 CAS 指令修改整数变量的值。如果加锁失败，lock 函数把线程放入线程等待队列，trylock 函数直接返回加锁失败状态。

加锁函数的示意代码

```
#include <stdbool.h>
#include <syscall.h>

struct my_mutex
{
    int lock;         // 整数变量。CAS 操作
    int owner_id;     // 加锁线程 ID
    int count;        // 重入计数
    void *wait_list; // 线程等待队列
};

// 把当前线程放入等待队列
void func_put_wait_list(struct my_mutex *mutex)
{
    // 过程略
}

// 加锁
void func_mutex_lock(struct my_mutex *mutex)
{
    // 当前的线程 ID
    int curr_tid = syscall(SYS_gettid);
    // 判断线程 ID
    if (mutex->owner_id == curr_tid)
    {
        // 已经加锁了，就计数加 1
        mutex->count++;
        return;
    }
    // 循环
    while (true)
    {
        // 尝试 CAS
        bool ret = __sync_bool_compare_and_swap(&(mutex->lock), 0, 1);
        // 成功，更新属性，退出
        if (ret)
```

```
        {
            mutex->owner_id = curr_tid; // 更新线程 ID
            mutex->count = 1;           // 计数加 1
            return;
        }
        // 加锁失败，把线程放入等待队列
        func_put_wait_list(mutex);
    }
}
```

从 glibc 源码查看阻塞锁的定义

文件 glibc-2.31/sysdeps/nptl/bits/pthreadtypes.h。
阻塞锁 pthread_mutex_t，属性包括 struct __pthread_mutex_s。

```
typedef union
{
  struct __pthread_mutex_s __data;
  char __size[__SIZEOF_PTHREAD_MUTEX_T];
  long int __align;
} pthread_mutex_t;
```

文件 glibc-2.31/sysdeps/nptl/bits/struct_mutex.h。
阻塞锁的详细定义。

```
struct __pthread_mutex_s
{
  int __lock __LOCK_ALIGNMENT;
  unsigned int __count;
  int __owner;
#if __WORDSIZE == 64
  unsigned int __nusers;
#endif
  int __kind;
#if __WORDSIZE != 64
  unsigned int __nusers;
#endif
#if __WORDSIZE == 64
  int __spins;
  __pthread_list_t __list;
# define __PTHREAD_MUTEX_HAVE_PREV      1
#else
  __extension__ union
  {
    int __spins;
    __pthread_slist_t __list;
  };
# define __PTHREAD_MUTEX_HAVE_PREV      0
```

```
#endif
};
```

文件 glibc-2.31/sysdeps/nptl/lowlevellock.h。
加锁函数，使用 CAS 指令把变量值从 0 改成 1，如果成功就返回，如果失败就让线程等待指定条件。

```
/* If FUTEX is 0 (not acquired), set to 1 (acquired with no waiters) and
   return.  Otherwise, ensure that it is >1 (acquired, possibly with waiters)
   and then block until we acquire the lock, at which point FUTEX will still be
   >1.  The lock is always acquired on return.  */
#define __lll_lock(futex, private)                                      \
  ((void)                                                               \
   ({                                                                   \
     int *__futex = (futex);                                            \
     if (__glibc_unlikely                                               \
         (atomic_compare_and_exchange_bool_acq (__futex, 1, 0)))        \
       {                                                                \
         if (__builtin_constant_p (private) && (private) == LLL_PRIVATE) \
           __lll_lock_wait_private (__futex);                           \
         else                                                           \
           __lll_lock_wait (__futex, private);                          \
       }                                                                \
   }))
#define lll_lock(futex, private)    \
  __lll_lock (&(futex), private)
```

文件 glibc-2.31/nptl/lowlevellock.c。
使用 linux 的 futex 功能，挂起线程，直到满足指定的条件。

```
void
__lll_lock_wait (int *futex, int private)
{
  if (atomic_load_relaxed (futex) == 2)
    goto futex;

  while (atomic_exchange_acquire (futex, 2) != 0)
    {
    futex:
      LIBC_PROBE (lll_lock_wait, 1, futex);
      lll_futex_wait (futex, 2, private); /* Wait if *futex == 2.  */
    }
}
```

文件 glibc-2.31/nptl/pthread_mutex_lock.c。
加锁函数，代码很多，这里截取部分代码。
逻辑为：判断是重入锁。如果已经加锁成功的线程 ID 等于当前线程 ID，就计数加 1，然后返回；否则使用 LLL_MUTEX_LOCK (mutex)加锁，加锁失败则线程挂起，加锁成功则计数加 1 并返回。

```
int
__pthread_mutex_lock (pthread_mutex_t *mutex)

  else if (__builtin_expect (PTHREAD_MUTEX_TYPE (mutex)
```

```
                == PTHREAD_MUTEX_RECURSIVE_NP, 1))
    {
      /* Recursive mutex.  */
      pid_t id = THREAD_GETMEM (THREAD_SELF, tid);

      /* Check whether we already hold the mutex.  */
      if (mutex->__data.__owner == id)
    {
      /* Just bump the counter.  */
      if (__glibc_unlikely (mutex->__data.__count + 1 == 0))
        /* Overflow of the counter.  */
        return EAGAIN;

      ++mutex->__data.__count;

      return 0;
    }

      /* We have to get the mutex.  */
      LLL_MUTEX_LOCK (mutex);

      assert (mutex->__data.__owner == 0);
      mutex->__data.__count = 1;
    }
```

用 C 和汇编分析阻塞锁的使用

编写代码： mutex_lock.c

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

// 给初始值，使得在汇编文件看到定义
pthread_mutex_t mutexlock = {
    .__data.__lock = 1,
    .__data.__count = 2,
    .__data.__owner = 3,
    .__data.__nusers = 4,
    .__data.__kind = 5};
pthread_mutexattr_t mutexattr;

// 打印锁的信息
void print_lock(char *title)
{
```

```
    printf(" %s  __lock=%d __count=%d __owner=%d \n", title,
           mutexlock.__data.__lock, mutexlock.__data.__count, mutexlock.__data.__owner);
}

// 线程执行加锁
void *thread_func_lock(void *param)
{
    pid_t os_tid = syscall(SYS_gettid); // 线程 ID
    pthread_mutex_lock(&mutexlock);     // 加锁
    printf(" son os_tid = %d  got lock \n", os_tid);
    sleep(1);
    print_lock("son lock");
    pthread_mutex_unlock(&mutexlock); // 解锁
    return NULL;
}

// 创建线程
void create_thread()
{
    pthread_t t1;
    pthread_create(&t1, NULL, thread_func_lock, NULL);
    pthread_t t2;
    pthread_create(&t2, NULL, thread_func_lock, NULL);
}

int main()
{
    // OS 的线程 ID
    pid_t os_tid = syscall(SYS_gettid);
    printf(" main os_tid = %d \n", os_tid);

    // 大小
    int size = sizeof(mutexlock);
    printf(" pthread_mutex_t size = %d \n", size);

    // 初始化
    pthread_mutexattr_init(&mutexattr);
    // 设置可重入。
    pthread_mutexattr_settype(&mutexattr, PTHREAD_MUTEX_RECURSIVE_NP);
    pthread_mutex_init(&mutexlock, &mutexattr);
    print_lock("main init");

    // 主线程加锁
    pthread_mutex_lock(&mutexlock);
    print_lock("main lock1");

    create_thread(); // 新建其他线程
    sleep(1);        // 等其他线程加锁，阻塞
```

```
    // 主线程加锁
    pthread_mutex_lock(&mutexlock);
    print_lock("main lock2");

    // 主线程解锁
    pthread_mutex_unlock(&mutexlock);
    print_lock("main unlock1");

    // 主线程解锁
    pthread_mutex_unlock(&mutexlock);
    print_lock("main unlock2");

    sleep(5);

    // 销毁。
    pthread_mutex_destroy(&mutexlock);
    pthread_mutexattr_destroy(&mutexattr);
    return 0;
}
```

编译代码：

```
gcc mutex_lock.c -lpthread -o mutex_lock
gcc mutex_lock.c -lpthread -S -o mutex_lock.s
```

运行代码：

```
[root@local lock]# ./mutex_lock
 main os_tid = 40435
 pthread_mutex_t size = 40
 main init  __lock=0 __count=0 __owner=0
 main lock1  __lock=1 __count=1 __owner=40435
 main lock2  __lock=2 __count=2 __owner=40435
 main unlock1  __lock=2 __count=1 __owner=40435
 main unlock2  __lock=0 __count=0 __owner=0
 son os_tid = 40436  got lock
 son lock  __lock=2 __count=1 __owner=40436
 son os_tid = 40437  got lock
 son lock  __lock=2 __count=1 __owner=40437
```

分析结果：

查看 mutex_lock.s，找到符号 mutexlock。pthread_mutex_t 占用 40 字节。

```
mutexlock:
    .long   1
    .long   2
    .long   3
    .long   4
    .long   5
    .zero   20
```

主线程加锁 1 次后，__owner 等于主线程 ID，__count 计数等于 1。

主线程加锁 2 次后，__count 计数等于 2。
子线程加锁后，__owner 等于子线程 ID。