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

非阻塞锁的含义

非阻塞锁，核心包括 1 个整数变量。
加锁、解锁，使用 CAS 指令修改整数变量的值。如果加锁失败，lock 函数循环尝试加锁，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>

// 加锁
void func_spin_lock(int *lock)
{
    // 循环
    while (true)
    {
        // 尝试 CAS
        bool ret = __sync_bool_compare_and_swap(lock, 0, 1);
        // 成功就退出
        if (ret)
        {
            return;
        }
    }
}
```

从 glibc 源码查看非阻塞锁的定义

文件 glibc-2.31/sysdeps/nptl/bits/pthreadtypes.h。
非阻塞锁，直接用 1 个整数变量来表示。

```
/* POSIX spinlock data type.  */
typedef volatile int pthread_spinlock_t;
```

文件 glibc-2.31/sysdeps/i386/nptl/pthread_spin_init.c。
初始化函数 pthread_spin_init 复用解锁函数 pthread_spin_unlock。

```
/* Not needed.  pthread_spin_init is an alias for pthread_spin_unlock.  */
```

文件 glibc-2.31/sysdeps/x86_64/nptl/pthread_spin_trylock.S。
尝试加锁函数 pthread_spin_trylock，使用 cmpxchgl 指令，尝试修改值从 1 到 0。

```
ENTRY(pthread_spin_trylock)
    movl    $1, %eax
    xorl    %ecx, %ecx
```

```
    LOCK
    cmpxchgl %ecx, (%rdi)
    movl    $EBUSY, %eax
    cmovel  %ecx, %eax
    retq
END(pthread_spin_trylock)
```

文件 glibc-2.31/sysdeps/x86_64/nptl/pthread_spin_lock.S。
加锁函数 pthread_spin_lock，代码逻辑比较晦涩，没有使用 cmpxchgl 指令。使用循环，尝试把值减 1，然后比较结果是否为 0。

```
ENTRY(pthread_spin_lock)
1:  LOCK
    decl    0(%rdi)
    jne 2f
    xor %eax, %eax
    ret

    .align  16
2:  rep
    nop
    cmpl    $0, 0(%rdi)
    jg  1b
    jmp 2b
END(pthread_spin_lock)
```

文件 glibc-2.31/sysdeps/x86_64/nptl/pthread_spin_unlock.S。
解锁函数 pthread_spin_unlock，修改值为 1。加锁成功的线程才能解锁，只使用 movl。

```
ENTRY(pthread_spin_unlock)
    movl    $1, (%rdi)
    xorl    %eax, %eax
    retq
END(pthread_spin_unlock)

    /* The implementation of pthread_spin_init is identical.  */
    .globl  pthread_spin_init
pthread_spin_init = pthread_spin_unlock
```

文件 glibc-2.31/nptl/pthread_spin_init.c。
初始化函数 pthread_spin_init，修改值为 0。

```
int
pthread_spin_init (pthread_spinlock_t *lock, int pshared)
{
  /* Relaxed MO is fine because this is an initializing store.  */
  atomic_store_relaxed (lock, 0);
  return 0;
}
```

文件 glibc-2.31/nptl/pthread_spin_lock.c。这里展示核心代码。
加锁函数 pthread_spin_lock，使用 atomic_compare_exchange_weak_acquire，进而使用 cmpxchg 指令。在循环内，

如果修改值从 0 到 1 成功，就退出循环。

```
int
pthread_spin_lock (pthread_spinlock_t *lock)
{
  int val = 0;
  if (__glibc_likely (atomic_compare_exchange_weak_acquire (lock, &val, 1)))
    return 0;

  do
    {
      do
    {
      atomic_spin_nop ();
      val = atomic_load_relaxed (lock);
    }
      while (val != 0);
    }
  while (!atomic_compare_exchange_weak_acquire (lock, &val, 1));

  return 0;
}
```

用 C 分析非阻塞锁

编写代码：spin_lock.c

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

// 给一个初始值，使得在汇编文件看到定义
pthread_spinlock_t spinlock_aa = 3333;

int main()
{
    int spin_size = sizeof(spinlock_aa);
    printf(" spinlock size = %d \n", spin_size);

    pthread_spin_init(&spinlock_aa, PTHREAD_PROCESS_PRIVATE);
    printf(" after init . value = %d \n", spinlock_aa);

    pthread_spin_trylock(&spinlock_aa);
    printf(" after trylock . value = %d \n", spinlock_aa);

    pthread_spin_unlock(&spinlock_aa);
    printf(" after unlock . value = %d \n", spinlock_aa);
```

```
    return 0;
}
```

编译代码：

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

运行代码：

```
[root@local lock]# ./spin_lock
 spinlock size = 4
 after init . value = 1
 after trylock . value = 0
 after unlock . value = 1
```

分析结果：

查看 spin_lock.s，找到符号 spinlock_aa。pthread_spinlock_t 的类型为 4 字节整数。

```
spinlock_aa:
    .long   3333
```

初始化之后，spinlock_aa 的值为 1。

尝试加锁之后，spinlock_aa 的值为 0。

解锁之后，spinlock_aa 的值为 1。