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

C 语言的 volatile 关键字说明

C 语言的 volatile 关键字，属于编译期屏障。
使用 volatile 修饰的变量，禁止编译期重排序，读写操作编译为读写内存指令。

用 C 和汇编分析没有使用 volatile 的场景

编写代码： without_volatile.c

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

// 普通整数变量
int num_aa = 1;
int num_bb = 2;
int num_cc = 3;

// 查看顺序
void func_order()
{
    // 修改 bb
    num_bb = num_aa;
    // 修改 cc
    num_cc = 5;
}

// 循环的标记
int loop_mark = 1;
// 循环的次数
long long loop_count = 0;

// 查看循环
void func_loop()
{
    // 判断标记。如果标记为 0，则退出循环。
    while (loop_mark)
    {
        loop_count++;
        // 累加到很大的数字，就退出。
        if (loop_count > 9000000000000)
```

```
        {
            break;
        }
    }
}

// 线程的函数
void *thread_func(void *param)
{
    // 休眠一会
    sleep(1);
    // 关闭循环
    loop_mark = 0;
    printf(" thread change loop_mark to %d \n", loop_mark);
}

int main()
{
    // 查看顺序
    func_order();
    printf(" num  bb = %d  cc = %d \n", num_bb, num_cc);

    // 新建一个线程，修改循环标记
    pthread_t pid;
    pthread_create(&pid, NULL, thread_func, NULL);

    // 查看循环
    func_loop();
    printf(" loop  count = %lld \n", loop_count);
    return 0;
}
```

编译代码：

```
gcc without_volatile.c -O2 -lpthread -o without_volatile
gcc without_volatile.c -O2 -lpthread -S -o without_volatile.s
```

运行代码：

```
[root@local volatile]# ./without_volatile
 num  bb = 1  cc = 5
 thread change loop_mark to 0
```

分析结果：
编译时使用参数-O2 开启编译优化。
C 语言代码，先写 num_bb，再写 num_cc。 `num_bb = num_aa` 在前， `num_cc = 5` 在后。
汇编代码，先写 num_cc，再写 num_bb。 `movl  $5, num_cc(%rip)` 在前， `movl  %eax, num_bb(%rip)` 在后。
源码顺序和汇编代码顺序不一致，发生指令乱序。

C 语言代码， `while (loop_mark)` ，判断 loop_mark，如果为 0 就退出循环。
汇编代码， `movl loop_mark(%rip), %edx` 把 loop_mark 写到 edx， `testl  %edx, %edx` 判断 edx 是否等于 0。

线程把 loop_mark 改为 0 之后，循环没有退出，说明功能不符合预期。原因为，汇编代码中，循环体没有重新从内存读 loop_mark，无法感知 loop_mark 变化。

```
.L10:
    movabsq $9000000000000, %rdx
    .p2align 4,,10
    .p2align 3
.L8:
    addq    $1, %rax
    cmpq    %rdx, %rax
    jle .L8
```

用 C 和汇编分析使用 volatile 的场景

编写代码： with_volatile.c

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

// 普通整数变量
int num_aa = 1;
volatile int num_bb = 2;
volatile int num_cc = 3;

// 查看顺序
void func_order()
{
    // 修改 bb
    num_bb = num_aa;
    // 修改 cc
    num_cc = 5;
}

// 循环的标记
volatile int loop_mark = 1;
// 循环的次数
long long loop_count = 0;

// 查看循环
void func_loop()
{
    // 判断标记。如果标记为 0，则退出循环。
    while (loop_mark)
    {
        loop_count++;
        // 累加到很大的数字，就退出。
```

```
        if (loop_count > 9000000000000)
        {
            break;
        }
    }
}

// 线程的函数
void *thread_func(void *param)
{
    // 休眠一会
    sleep(1);
    // 关闭循环
    loop_mark = 0;
    printf(" thread change loop_mark to %d \n", loop_mark);
}

int main()
{
    // 查看顺序
    func_order();
    printf(" num  bb = %d  cc = %d \n", num_bb, num_cc);

    // 新建一个线程，修改循环标记
    pthread_t pid;
    pthread_create(&pid, NULL, thread_func, NULL);

    // 查看循环
    func_loop();
    printf(" loop  count = %lld \n", loop_count);
    sleep(1);
    return 0;
}
```

编译代码：

```
gcc with_volatile.c -O2 -lpthread -o with_volatile
gcc with_volatile.c -O2 -lpthread -S -o with_volatile.s
```

运行代码：

```
[root@local volatile]# ./with_volatile
 num  bb = 1  cc = 5
 thread change loop_mark to 0
 loop  count = 1719026076
```

分析结果：
编译时使用参数-O2 开启编译优化。
C 语言代码，先写 num_bb，再写 num_cc。 `num_bb = num_aa` 在前， `num_cc = 5` 在后。
汇编代码，先写 num_bb，再写 num_cc。 `movl  %eax, num_bb(%rip)` 在前， `movl  $5, num_cc(%rip)` 在后。
源码顺序和汇编代码顺序一致，没有指令乱序。

C 语言代码， `while (loop_mark)` ，判断 loop_mark，如果为 0 就退出循环。
汇编代码， `movl loop_mark(%rip), %edx` 把 loop_mark 写到 edx， `testl %edx, %edx` 判断 edx 是否等于 0。
线程把 loop_mark 改为 0 之后，循环退出，说明功能符合预期。
原因为，汇编代码中，循环体重新从内存读 loop_mark，可以感知 loop_mark 变化。

```
.L7:
    addq    $1, %rax
    cmpq    %rcx, %rax
    jg  .L9
.L5:
    movl    loop_mark(%rip), %edx
    testl   %edx, %edx
    jne .L7
```