**从汇编角度理解volatile**

摘要：本文分析了可能需要volatile的几个地方，并通过对各种情况下的汇编代码的分析来加深理解。

一般对于volatile的解释是这样的：将变量定义为volatile可以防止编译器对变量进行优化，每次均从内存中访问变量，而不是寄存器。既然让编译器优化可以提高访问速度，那为什么又要不用它以及什么时候不用它？其实主要是配合的问题，一个变量可能在寄存器中，也可能在内存中，如果程序只有一个执行流且没有跳转，那么变量不管放到内存还是寄存器，理论上在编译阶段编译器是能够知道变量的具体位置的，使用时就不会出现问题。事实上编译器是通过一个简单的规则来保证的：局部变量一般会放到寄存器中，全局变量在函数开始时必然是操作内存，之后可能是放入寄存器，到函数结束时必然将全局变量的值再写入内存(如果有写的话)，这样每次函数调用时全局变量的最新值都在内存中，且都从内存中访问，保证了数据的一致性。但这种一致性在多执行流或有跳转的情况下就有问题了。目前程序的多执行流主要有：多线程和设备寄存器，跳转主要有：setjmp/longjmp,goto,信号处理，下面我们挨个情况来分析。

**线程间共享变量加锁**

mutex.c:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int g\_i;

void \* pfunc(void \*args) {

while(1) {

pthread\_mutex\_lock(&mutex);

g\_i = 3;

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

int main(void) {

pthread\_t tid;

pthread\_mutex\_init(&mutex,NULL);

pthread\_create(&tid,NULL,pfunc,NULL);

int i,j,k;

for(i=0;i<100000;i++){

pthread\_mutex\_lock(&mutex);

printf("g\_id in main is %d\n",g\_i);

pthread\_mutex\_unlock(&mutex);

}

return 0;

}

正常情况下会看到g\_id最后变为3，当访问全局变量加锁时，不论是否加volatile，是否优化结果都正确，原因在于编译器对全局变量进行优化还有这样的规则：函数内代码在执行过程中，只要有访问到下级函数(即被调用函数)，全局变量必须访问内存，因为编译器并不知道下级函数是否会修改全局变量的值，所以必须直接从内存中读取全局变量或把全局变量中的值写到内存，加锁时pthread\_mutex\_lock/ pthread\_mutex\_unlock充当了下级函数，所以此时虽然在多个执行流中同时访问全局变量，由于互斥锁保证了访问时序，而每次访问时都操作内存，保证了全局变量的一致性，所以加不加volatile没关系。细节详见下面的汇编分析。

**结论是：线程间共享变量加锁访问加不加volatile都一样。**

**线程间共享变量不加锁**

承接上面的讨论，如果操作全局变量前后没有函数调用，只是对全局变量做了一些操作。因为第一次函数内操作全局变量是会在内存中进行，并且函数内不调用其它函数，编译器一般会将对全局变量的多次操作优化为一次操作，且对内存进行操作，按说这种情况下也没有问题。但有一种情况比较特殊，即在某函数中存在死循环，而对全局变量的操作在死循环中，由于编译器仅在编译过程起作用，不会考虑运行时多执行流的交互影响，会想当然的认为在死循环中的操作是无意义的，会直接优化掉对全局变量的操作，可能引发问题。下面是示例代码，细节分析请看汇编代码。

Nomutex.c

void \* pfunc(void \*args) {

int i =3;

while(1) {

g\_i = 3;//仅去掉了加锁部分

}

return NULL;

}

这里需要说明一下，为什么有时候线程间共享变量可以不加锁。有一种情况，即全局变量仅在一个或多个线程内修改，在多个线程中读取，并且对变量的写操作是可以在一条cpu指令中执行完成的(cpu指令执行时是原子的，不可打断的)，典型情况是对整数赋值，这时在读取线程中即便不用锁，读到的内容也要么是旧的，要么是新的，不会有中间状态或脏数据的情况出现。对于经典的i++这样的操作是必须加锁的，虽然加1(incl)这条指令是原子的，但通常会先将变量从内存中取出放到寄存器，在寄存器中加1，然后再把结果写回内存，这是三条指令，这三条指令无法保证原子性。当然也可以用原子操作，如\_\_sync\_fetch\_and\_add，其实原子操作也是调用了lock指令的，但却比加锁快了1倍左右，估计是其它方面的优化。详见下面的汇编分析。

**结论是：线程间共享变量不加锁访问尽量加volatile，当然一般代码没有那么简单，问题不大。**

**Setjmp/longjmp:**

见setjmp.c例子：

#include<setjmp.h>

#include <stdio.h>

static jmp\_buf buf;

int main()

{

int b = 0;

printf("first b is %d\n",b);

if(setjmp(buf)!=0) {

printf("after jmp b is %d\n", b);

exit(0);

}

b=5;

printf("after set b is %d\n",b);

longjmp(buf , 1);

printf("normal exit\n");

return 0;

}

gcc -g setjmp.c &&./a.out

first b is 0

after set b is 5

after jmp b is 5

gcc -g setjmp.c -O2 && ./a.out

first b is 0

after set b is 5

after jmp b is 0

大家看两种编译结果afterjmp后b的值是不一样的，对变量b加了volatile后发现又正常了，说明编译器在优化局部变量时没有考虑setjmp/longjmp，因为setjmp/longjmp是可以跳转到别的函数中的，所以编译器认为无能为力，干脆即便跳到本函数内也不管，还是按照以前的优化策略，这样就导致问题了.但是goto却不会出问题，因为goto仅能在函数内部跳转，编译器会通盘考虑，详见下面的汇编分析。

**结论是：尽量不要用setjmp/longjmp,如果用的话不要在函数内跳转，全局变量的问题参见其他部分。**

**设备寄存器**

假设某种平台采用内存映射I/O，即设备寄存器(一般包含串口发送寄存器和串口接收寄存器)位于固定的内存地址，可以像操作普通内存地址一样操作设备的发送寄存器和接收寄存器。设备寄存器往往具有以下特性：

* 设备寄存器中的数据不需要cpu改写就可以自己发生变化，每次读上来的值都可能不一样。
* 连续多次向设备寄存器中写数据并不是在做无用功，而是有特殊意义的。

而对于编译器来说，如果有如下的代码

#include <stdio.h>

unsigned char recv;//串口接收寄存器

unsigned char send;//串口发送寄存器

unsigned char buf[3];

int main(void)

{

buf[0] = recv;

buf[1] = recv;

buf[2] = recv;

send = ~buf[0];

send = ~buf[1];

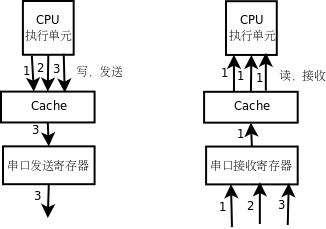
send = ~buf[2];

return 0;

}

如果是普通的内存单元，只要程序不去改写它，它就不会变，可以先把内存单元里的值读到寄存器缓存起来，以后每次用到这个值就直接从寄存器读取，这样效率更高。另一方面，如果对一个普通的内存单元连续做三次写操作，只有最后一次的值会保存到内存单元中，所以前两次写操作是多余的，可以优化掉。但如果这个地址是设备寄存器，这样优化就错了，这时必须用volatile限定符修饰变量，告诉编译器，即使在编译时指定了优化选项，每次读这个变量仍然要老老实实从内存读取，每次写这个变量也仍然要老老实实写回内存，不能省略任何步骤。

有了volatile限定符，是可以防止编译器优化对设备寄存器的访问，但是对于有Cache的平台，仅仅这样还不够，还是无法防止Cache优化对设备寄存器的访问。在访问普通的内存单元时，Cache对程序员是透明的，比如执行了movzbl 0x804a019,%eax这样一条指令，我们并不知道eax的值是真的从内存地址0x804a019读到的，还是从Cache中读到的，如果Cache已经缓存了这个地址的数据就从Cache读，如果Cache没有缓存就从内存读，这些步骤都是硬件自动做的，而不是用指令控制Cache去做的，程序员写的指令中只有寄存器、内存地址，而没有Cache，程序员甚至不需要知道Cache的存在。同样道理，如果执行了mov %al,0x804a01a这样一条指令，我们并不知道寄存器的值是真的写回内存了，还是只写到了Cache中，以后再由Cache写回内存，即使只写到了Cache中而暂时没有写回内存，下次读0x804a01a这个地址时仍然可以从Cache中读到上次写的数据。然而，在读写设备寄存器时Cache的存在就不容忽视了，如果串口发送和接收寄存器的内存地址被Cache缓存了会有什么问题呢？如下图所示。



如果串口发送寄存器的地址被Cahce缓存，CPU执行单元对串口发送寄存器做写操作都写到Cache中去了，串口发送寄存器并没有及时得到数据，也就不能及时发送，CPU执行单元先后发出的1、2、3三个字节都会写到Cache中的同一个单元，最后Cache中只保存了第3个字节，如果这时Cache把数据写回到串口发送寄存器，只能把第3个字节发送出去，前两个字节就丢失了。与此类似，如果串口接收寄存器的地址被Cache缓存，CPU执行单元在读第1个字节时，Cache会从串口接收寄存器读上来缓存，然而串口接收寄存器后面收到的2、3两个字节Cache并不知道，因为Cache把串口接收寄存器当作普通内存单元，并且相信内存单元中的数据是不会自己变的，以后每次读串口接收寄存器时，Cache都会把缓存的第1个字节提供给CPU执行单元。

通常，有Cache的平台都有办法对某一段地址范围禁用Cache，一般是在页表中设置的，可以设定哪些页面允许Cache缓存，哪些页面不允许Cache缓存。

**结论是：对于设备寄存器变量，必须设置volatile。**

**信号处理**

Signal.c:

#define BIGNUM (1000)

int total=0;

void handle(int signo)

{

printf("%u\n",total);

if (signo == SIGALRM)

{

printf("Total = %d\n",total);

exit(0);

}

total++;

}

int main ()

{

unsigned x=0;

int i,j;

struct sched\_param param;

struct itimerval val;

val.it\_interval.tv\_sec = 0;

val.it\_interval.tv\_usec = 10000; // 10 ms

val.it\_value.tv\_sec = 0;

val.it\_value.tv\_usec = 10000; // 10 ms

setitimer(ITIMER\_VIRTUAL, &val, NULL);

alarm(10);

signal(SIGALRM,handle);

signal(SIGVTALRM,handle);

while (total < 100 )

{

//Printf(“total is %d\n”,total);

for (i=0; i<BIGNUM; i++)

{

for (j=0; j<BIGNUM; j++)

x = x+j ;

}

}

printf("total = %d\n",total);

}

编译器优化后会认为在main函数中total值没有变化，原因是while语句中没有函数调用，将print注释打开后就不一样了，具体请看汇编分析。

**结论是：当信号处理函数中操作某个全局变量时此全局变量最好加上volatile。**

**总结：对全局变量最好都加volatile，事实上加与不加性能上没有太大影响，不必担心这方面。如果做不到全部全局变量加volalile的话，请看下面：**

1. **对于设备寄存器变量，必须设置volatile**
2. **尽量不要用setjmp/longjmp,更不要用它在函数内跳转。**
3. **当访问全局变量时有死循环，且没有下级函数时要尤其注意。**

[1 mutex\_O0 13](#_Toc318411810)

[2 mutex\_O2 14](#_Toc318411811)

[3 nomutex\_O2 16](#_Toc318411812)

[4 Setjmp\_O0 17](#_Toc318411813)

[5 Setjmp\_O2 18](#_Toc318411814)

[6 Setjmp\_O2\_volatile 19](#_Toc318411815)

[7 goto\_O0 20](#_Toc318411816)

[8 goto\_O2 20](#_Toc318411817)

[9 reg\_O0 21](#_Toc318411818)

[10 reg\_O2 22](#_Toc318411819)

[11 signal\_O0 23](#_Toc318411820)

[12 signal\_O2 25](#_Toc318411821)

# mutex\_O0

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int g\_i;

void \* pfunc(void \*args) {

400638: 55 push %rbp //保存栈帧指针

400639: 48 89 e5 mov %rsp,%rbp //更新栈顶指针为栈帧指针

40063c: 48 83 ec 10 sub $0x10,%rsp//向下移动栈顶指针

400640: 48 89 7d f8 mov %rdi,0xfffffffffffffff8(%rbp)//保存函数输入参数

while(1) {

pthread\_mutex\_lock(&mutex);

400644: bf 20 0b 50 00 mov $0x500b20,%edi

400649: e8 f2 fe ff ff callq 400540 <pthread\_mutex\_lock@plt>//函数调用

g\_i = 3;

40064e: c7 05 f0 04 10 00 03 movl $0x3,1049840(%rip) # 500b48 <g\_i>

400655: 00 00 00

pthread\_mutex\_unlock(&mutex);

400658: bf 20 0b 50 00 mov $0x500b20,%edi//准备参数

40065d: e8 0e ff ff ff callq 400570 <pthread\_mutex\_unlock@plt>//函数调用

400662: eb e0 jmp 400644 <pfunc+0xc>//调用到循环处

0000000000400664 <main>:

}

return NULL;

}

int main(void) {

400664: 55 push %rbp//保存栈帧指针

400665: 48 89 e5 mov %rsp,%rbp//更新栈顶指针为栈帧指针

400668: 48 83 ec 20 sub $0x20,%rsp//向下移动栈顶指针

pthread\_t tid;

pthread\_mutex\_init(&mutex,NULL);

40066c: be 00 00 00 00 mov $0x0,%esi//准备参数

400671: bf 20 0b 50 00 mov $0x500b20,%edi//准备参数

400676: e8 e5 fe ff ff callq 400560 <pthread\_mutex\_init@plt>

pthread\_create(&tid,NULL,pfunc,NULL);

40067b: 48 8d 7d f8 lea 0xfffffffffffffff8(%rbp),%rdi//将tid的地址让入rdi寄存器

40067f: b9 00 00 00 00 mov $0x0,%ecx

400684: ba 38 06 40 00 mov $0x400638,%edx//pfunc

400689: be 00 00 00 00 mov $0x0,%esi

40068e: e8 9d fe ff ff callq 400530 <pthread\_create@plt>

int i,j,k;

for(i=0;i<100000;i++){

400693: c7 45 f4 00 00 00 00 movl $0x0,0xfffffffffffffff4(%rbp)//i变量地址

40069a: 81 7d f4 9f 86 01 00 cmpl $0x1869f,0xfffffffffffffff4(%rbp)

4006a1: 7f 31 jg 4006d4 <main+0x70>

pthread\_mutex\_lock(&mutex);

4006a3: bf 20 0b 50 00 mov $0x500b20,%edi

4006a8: e8 93 fe ff ff callq 400540 <pthread\_mutex\_lock@plt>

printf("g\_id in main is %d\n",g\_i);

4006ad: 8b 35 95 04 10 00 mov 1049749(%rip),%esi # 500b48 <g\_i>

4006b3: bf cc 07 40 00 mov $0x4007cc,%edi

4006b8: b8 00 00 00 00 mov $0x0,%eax

4006bd: e8 8e fe ff ff callq 400550 <printf@plt>

pthread\_mutex\_unlock(&mutex);

4006c2: bf 20 0b 50 00 mov $0x500b20,%edi

4006c7: e8 a4 fe ff ff callq 400570 <pthread\_mutex\_unlock@plt>

4006cc: 48 8d 45 f4 lea 0xfffffffffffffff4(%rbp),%rax

4006d0: ff 00 incl (%rax)

4006d2: eb c6 jmp 40069a <main+0x36>

//test();

}

return 0;

# mutex\_O2

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int g\_i;

void \* pfunc(void \*args) {

400640: 48 83 ec 08 sub $0x8,%rsp

while(1) {

pthread\_mutex\_lock(&mutex);

400644: bf 20 0b 50 00 mov $0x500b20,%edi

400649: e8 f2 fe ff ff callq 400540 <pthread\_mutex\_lock@plt>

g\_i = 3;

pthread\_mutex\_unlock(&mutex);

40064e: bf 20 0b 50 00 mov $0x500b20,%edi

400653: c7 05 eb 04 10 00 03 movl $0x3,1049835(%rip) # 500b48 <g\_i>

40065a: 00 00 00

40065d: e8 0e ff ff ff callq 400570 <pthread\_mutex\_unlock@plt>

400662: eb e0 jmp 400644 <pfunc+0x4>

400664: 66 data16

400665: 66 data16

400666: 66 data16

400667: 90 nop

400668: 66 data16

400669: 66 data16

40066a: 66 data16

40066b: 90 nop

40066c: 66 data16

40066d: 66 data16

40066e: 66 data16

40066f: 90 nop

0000000000400670 <main>:

}

return NULL;

int main(void) {

400670: 53 push %rbx

pthread\_t tid;

pthread\_mutex\_init(&mutex,NULL);

400671: 31 f6 xor %esi,%esi

400673: bf 20 0b 50 00 mov $0x500b20,%edi

400678: bb 9f 86 01 00 mov $0x1869f,%ebx

40067d: 48 83 ec 10 sub $0x10,%rsp

400681: e8 da fe ff ff callq 400560 <pthread\_mutex\_init@plt>

pthread\_create(&tid,NULL,pfunc,NULL);

400686: 48 8d 7c 24 08 lea 0x8(%rsp),%rdi

40068b: 31 c9 xor %ecx,%ecx

40068d: ba 40 06 40 00 mov $0x400640,%edx

400692: 31 f6 xor %esi,%esi

400694: e8 97 fe ff ff callq 400530 <pthread\_create@plt>

400699: 66 data16

40069a: 66 data16

40069b: 66 data16

40069c: 90 nop

40069d: 66 data16

40069e: 66 data16

40069f: 90 nop

int i,j,k;

for(i=0;i<100000;i++){

pthread\_mutex\_lock(&mutex);

4006a0: bf 20 0b 50 00 mov $0x500b20,%edi

4006a5: e8 96 fe ff ff callq 400540 <pthread\_mutex\_lock@plt>

printf("g\_id in main is %d\n",g\_i);

4006aa: 8b 35 98 04 10 00 mov 1049752(%rip),%esi # 500b48 <g\_i>

4006b0: bf cc 07 40 00 mov $0x4007cc,%edi

4006b5: 31 c0 xor %eax,%eax

4006b7: e8 94 fe ff ff callq 400550 <printf@plt>

pthread\_mutex\_unlock(&mutex);

4006bc: bf 20 0b 50 00 mov $0x500b20,%edi

4006c1: e8 aa fe ff ff callq 400570 <pthread\_mutex\_unlock@plt>

4006c6: ff cb dec %ebx

4006c8: 79 d6 jns 4006a0 <main+0x30>

//test();

}

return 0;

}

# nomutex\_O2

int g\_i;

void \* pfunc(void \*args) {

int i =3;

while(1) {

g\_i = 3;

400640: eb fe jmp 400640 <pfunc>

400642: 66 data16

400643: 66 data16

400644: 66 data16

400645: 90 nop

400646: 66 data16

400647: 66 data16

400648: 66 data16

400649: 90 nop

40064a: 66 data16

40064b: 66 data16

40064c: 90 nop

40064d: 66 data16

40064e: 66 data16

40064f: 90 nop

0000000000400650 <main>:

}

return NULL;

}

# Setjmp\_O0

int main()

{

400598: 55 push %rbp

400599: 48 89 e5 mov %rsp,%rbp

40059c: 48 83 ec 10 sub $0x10,%rsp

int b = 0;

4005a0: c7 45 fc 00 00 00 00 movl $0x0,0xfffffffffffffffc(%rbp)

printf("first b is %d\n",b);

4005a7: 8b 75 fc mov 0xfffffffffffffffc(%rbp),%esi

4005aa: bf fc 06 40 00 mov $0x4006fc,%edi

4005af: b8 00 00 00 00 mov $0x0,%eax

4005b4: e8 ff fe ff ff callq 4004b8 <printf@plt>

if(setjmp(buf)!=0) {

4005b9: bf 40 0a 50 00 mov $0x500a40,%edi

4005be: e8 d5 fe ff ff callq 400498 <\_setjmp@plt>

4005c3: 85 c0 test %eax,%eax

4005c5: 74 1c je 4005e3 <main+0x4b>

printf("after jmp b is %d\n", b);

4005c7: 8b 75 fc mov 0xfffffffffffffffc(%rbp),%esi

4005ca: bf 0b 07 40 00 mov $0x40070b,%edi

4005cf: b8 00 00 00 00 mov $0x0,%eax

4005d4: e8 df fe ff ff callq 4004b8 <printf@plt>

exit(0);

4005d9: bf 00 00 00 00 mov $0x0,%edi

4005de: e8 e5 fe ff ff callq 4004c8 <exit@plt>

}

b=5;

4005e3: c7 45 fc 05 00 00 00 movl $0x5,0xfffffffffffffffc(%rbp)

printf("after set b is %d\n",b);

4005ea: 8b 75 fc mov 0xfffffffffffffffc(%rbp),%esi

4005ed: bf 1e 07 40 00 mov $0x40071e,%edi

4005f2: b8 00 00 00 00 mov $0x0,%eax

4005f7: e8 bc fe ff ff callq 4004b8 <printf@plt>

longjmp(buf , 1);

4005fc: be 01 00 00 00 mov $0x1,%esi

400601: bf 40 0a 50 00 mov $0x500a40,%edi

400606: e8 9d fe ff ff callq 4004a8 <longjmp@plt>

40060b: 90 nop

40060c: 90 nop

40060d: 90 nop

40060e: 90 nop

40060f: 90 nop

# Setjmp\_O2

int main()

{

4005a0: 48 83 ec 08 sub $0x8,%rsp

int b = 0;

printf("first b is %d\n",b);

4005a4: 31 f6 xor %esi,%esi

4005a6: bf ec 06 40 00 mov $0x4006ec,%edi

4005ab: 31 c0 xor %eax,%eax

4005ad: e8 06 ff ff ff callq 4004b8 <printf@plt>

if(setjmp(buf)!=0) {

4005b2: bf 20 0a 50 00 mov $0x500a20,%edi

4005b7: e8 dc fe ff ff callq 400498 <\_setjmp@plt>

4005bc: 85 c0 test %eax,%eax

4005be: 74 15 je 4005d5 <main+0x35>

printf("after jmp b is %d\n", b);

4005c0: bf fb 06 40 00 mov $0x4006fb,%edi

4005c5: 31 f6 xor %esi,%esi

4005c7: 31 c0 xor %eax,%eax

4005c9: e8 ea fe ff ff callq 4004b8 <printf@plt>

exit(0);

4005ce: 31 ff xor %edi,%edi

4005d0: e8 f3 fe ff ff callq 4004c8 <exit@plt>

}

b=5;

printf("after set b is %d\n",b);

4005d5: be 05 00 00 00 mov $0x5,%esi

4005da: bf 0e 07 40 00 mov $0x40070e,%edi

4005df: 31 c0 xor %eax,%eax

4005e1: e8 d2 fe ff ff callq 4004b8 <printf@plt>

longjmp(buf , 1);

4005e6: be 01 00 00 00 mov $0x1,%esi

4005eb: bf 20 0a 50 00 mov $0x500a20,%edi

4005f0: e8 b3 fe ff ff callq 4004a8 <longjmp@plt>

# Setjmp\_O2\_volatile

int main()

{

4005a0: 48 83 ec 08 sub $0x8,%rsp

volatile int b = 0;

printf("first b is %d\n",b);

4005a4: bf fc 06 40 00 mov $0x4006fc,%edi

4005a9: 31 c0 xor %eax,%eax

4005ab: c7 44 24 04 00 00 00 movl $0x0,0x4(%rsp)

4005b2: 00

4005b3: 8b 74 24 04 mov 0x4(%rsp),%esi

4005b7: e8 fc fe ff ff callq 4004b8 <printf@plt>

if(setjmp(buf)!=0) {

4005bc: bf 40 0a 50 00 mov $0x500a40,%edi

4005c1: e8 d2 fe ff ff callq 400498 <\_setjmp@plt>

4005c6: 85 c0 test %eax,%eax

4005c8: 74 17 je 4005e1 <main+0x41>

printf("after jmp b is %d\n", b);

4005ca: 8b 74 24 04 mov 0x4(%rsp),%esi

4005ce: bf 0b 07 40 00 mov $0x40070b,%edi

4005d3: 31 c0 xor %eax,%eax

4005d5: e8 de fe ff ff callq 4004b8 <printf@plt>

exit(0);

4005da: 31 ff xor %edi,%edi

4005dc: e8 e7 fe ff ff callq 4004c8 <exit@plt>

}

b=5;

4005e1: c7 44 24 04 05 00 00 movl $0x5,0x4(%rsp)

4005e8: 00

printf("after set b is %d\n",b);

4005e9: 8b 74 24 04 mov 0x4(%rsp),%esi

4005ed: bf 1e 07 40 00 mov $0x40071e,%edi

4005f2: 31 c0 xor %eax,%eax

4005f4: e8 bf fe ff ff callq 4004b8 <printf@plt>

longjmp(buf , 1);

4005f9: be 01 00 00 00 mov $0x1,%esi

4005fe: bf 40 0a 50 00 mov $0x500a40,%edi

400603: e8 a0 fe ff ff callq 4004a8 <longjmp@plt>

# goto\_O0

int main()

{

4004f8: 55 push %rbp

4004f9: 48 89 e5 mov %rsp,%rbp

4004fc: 48 83 ec 10 sub $0x10,%rsp

int b = 0;

400500: c7 45 fc 00 00 00 00 movl $0x0,0xfffffffffffffffc(%rbp)

printf("first b is %d\n",b);

400507: 8b 75 fc mov 0xfffffffffffffffc(%rbp),%esi

40050a: bf 4c 06 40 00 mov $0x40064c,%edi

40050f: b8 00 00 00 00 mov $0x0,%eax

400514: e8 07 ff ff ff callq 400420 <printf@plt>

if(b!=0) {

400519: 83 7d fc 00 cmpl $0x0,0xfffffffffffffffc(%rbp)

40051d: 74 1c je 40053b <main+0x43>

here:

printf("after jmp b is %d\n", b);

40051f: 8b 75 fc mov 0xfffffffffffffffc(%rbp),%esi

400522: bf 5b 06 40 00 mov $0x40065b,%edi

400527: b8 00 00 00 00 mov $0x0,%eax

40052c: e8 ef fe ff ff callq 400420 <printf@plt>

exit(0);

400531: bf 00 00 00 00 mov $0x0,%edi

400536: e8 f5 fe ff ff callq 400430 <exit@plt>

}

b=5;

40053b: c7 45 fc 05 00 00 00 movl $0x5,0xfffffffffffffffc(%rbp)

printf("after set b is %d\n",b);

400542: 8b 75 fc mov 0xfffffffffffffffc(%rbp),%esi

400545: bf 6e 06 40 00 mov $0x40066e,%edi

40054a: b8 00 00 00 00 mov $0x0,%eax

40054f: e8 cc fe ff ff callq 400420 <printf@plt>

goto here;

400554: eb c9 jmp 40051f <main+0x27>

# goto\_O2

int main()

{

400500: 48 83 ec 08 sub $0x8,%rsp

int b = 0;

printf("first b is %d\n",b);

400504: 31 f6 xor %esi,%esi

400506: bf 2c 06 40 00 mov $0x40062c,%edi

40050b: 31 c0 xor %eax,%eax

40050d: e8 0e ff ff ff callq 400420 <printf@plt>

if(b!=0) {

here:

printf("after jmp b is %d\n", b);

exit(0);

}

b=5;

printf("after set b is %d\n",b);

400512: be 05 00 00 00 mov $0x5,%esi

400517: bf 3b 06 40 00 mov $0x40063b,%edi

40051c: 31 c0 xor %eax,%eax

40051e: e8 fd fe ff ff callq 400420 <printf@plt>

400523: bf 4e 06 40 00 mov $0x40064e,%edi

400528: be 05 00 00 00 mov $0x5,%esi

40052d: 31 c0 xor %eax,%eax

40052f: e8 ec fe ff ff callq 400420 <printf@plt>

400534: 31 ff xor %edi,%edi

400536: e8 f5 fe ff ff callq 400430 <exit@plt>

# reg\_O0

int main(void)

{

4004a8: 55 push %rbp

4004a9: 48 89 e5 mov %rsp,%rbp

buf[0] = recv;

4004ac: 0f b6 05 01 04 10 00 movzbl 1049601(%rip),%eax # 5008b4 <recv>

4004b3: 88 05 fd 03 10 00 mov %al,1049597(%rip) # 5008b6 <buf>

buf[1] = recv;

4004b9: 0f b6 05 f4 03 10 00 movzbl 1049588(%rip),%eax # 5008b4 <recv>

4004c0: 88 05 f1 03 10 00 mov %al,1049585(%rip) # 5008b7 <buf+0x1>

buf[2] = recv;

4004c6: 0f b6 05 e7 03 10 00 movzbl 1049575(%rip),%eax # 5008b4 <recv>

4004cd: 88 05 e5 03 10 00 mov %al,1049573(%rip) # 5008b8 <buf+0x2>

send = ~buf[0];

4004d3: 0f b6 05 dc 03 10 00 movzbl 1049564(%rip),%eax # 5008b6 <buf>

4004da: f7 d0 not %eax

4004dc: 88 05 d3 03 10 00 mov %al,1049555(%rip) # 5008b5 <send>

send = ~buf[1];

4004e2: 0f b6 05 ce 03 10 00 movzbl 1049550(%rip),%eax # 5008b7 <buf+0x1>

4004e9: f7 d0 not %eax

4004eb: 88 05 c4 03 10 00 mov %al,1049540(%rip) # 5008b5 <send>

send = ~buf[2];

4004f1: 0f b6 05 c0 03 10 00 movzbl 1049536(%rip),%eax # 5008b8 <buf+0x2>

4004f8: f7 d0 not %eax

4004fa: 88 05 b5 03 10 00 mov %al,1049525(%rip) # 5008b5 <send>

return 0;

400500: b8 00 00 00 00 mov $0x0,%eax

# reg\_O2

int main(void)

{

buf[0] = recv;

4004b0: 0f b6 05 c5 03 10 00 movzbl 1049541(%rip),%eax # 50087c <recv>

4004b7: 88 05 c1 03 10 00 mov %al,1049537(%rip) # 50087e <buf>

buf[1] = recv;

4004bd: 88 05 bc 03 10 00 mov %al,1049532(%rip) # 50087f <buf+0x1>

buf[2] = recv;

4004c3: 88 05 b7 03 10 00 mov %al,1049527(%rip) # 500880 <buf+0x2>

send = ~buf[0];

4004c9: f7 d0 not %eax

send = ~buf[1];

send = ~buf[2];

4004cb: 88 05 ac 03 10 00 mov %al,1049516(%rip) # 50087d <send>

return 0;

}

4004d1: 31 c0 xor %eax,%eax

4004d3: c3 retq

# signal\_O0

int main ()

{

400635: 55 push %rbp

400636: 48 89 e5 mov %rsp,%rbp

400639: 48 83 ec 30 sub $0x30,%rsp

unsigned x=0;

40063d: c7 45 fc 00 00 00 00 movl $0x0,0xfffffffffffffffc(%rbp)

int i,j;

struct sched\_param param;

struct itimerval val;

val.it\_interval.tv\_sec = 0;

400644: 48 c7 45 d0 00 00 00 movq $0x0,0xffffffffffffffd0(%rbp)

40064b: 00

val.it\_interval.tv\_usec = 10000; // 10 ms

40064c: 48 c7 45 d8 10 27 00 movq $0x2710,0xffffffffffffffd8(%rbp)

400653: 00

val.it\_value.tv\_sec = 0;

400654: 48 c7 45 e0 00 00 00 movq $0x0,0xffffffffffffffe0(%rbp)

40065b: 00

val.it\_value.tv\_usec = 10000; // 10 ms

40065c: 48 c7 45 e8 10 27 00 movq $0x2710,0xffffffffffffffe8(%rbp)

400663: 00

setitimer(ITIMER\_VIRTUAL, &val, NULL);

400664: 48 8d 75 d0 lea 0xffffffffffffffd0(%rbp),%rsi

400668: ba 00 00 00 00 mov $0x0,%edx

40066d: bf 01 00 00 00 mov $0x1,%edi

400672: e8 81 fe ff ff callq 4004f8 <setitimer@plt>

alarm(10);

400677: bf 0a 00 00 00 mov $0xa,%edi

40067c: b8 00 00 00 00 mov $0x0,%eax

400681: e8 92 fe ff ff callq 400518 <alarm@plt>

signal(SIGALRM,handle);

400686: be e8 05 40 00 mov $0x4005e8,%esi

40068b: bf 0e 00 00 00 mov $0xe,%edi

400690: e8 53 fe ff ff callq 4004e8 <signal@plt>

signal(SIGVTALRM,handle);

400695: be e8 05 40 00 mov $0x4005e8,%esi

40069a: bf 1a 00 00 00 mov $0x1a,%edi

40069f: e8 44 fe ff ff callq 4004e8 <signal@plt>

while (total < 100 )

4006a4: 83 3d 69 04 10 00 63 cmpl $0x63,1049705(%rip) # 500b14 <total>

4006ab: 7f 39 jg 4006e6 <main+0xb1>

{

for (i=0; i<BIGNUM; i++)

4006ad: c7 45 f8 00 00 00 00 movl $0x0,0xfffffffffffffff8(%rbp)

4006b4: 81 7d f8 e7 03 00 00 cmpl $0x3e7,0xfffffffffffffff8(%rbp)

4006bb: 7f e7 jg 4006a4 <main+0x6f>

{

for (j=0; j<BIGNUM; j++)

4006bd: c7 45 f4 00 00 00 00 movl $0x0,0xfffffffffffffff4(%rbp)

4006c4: 81 7d f4 e7 03 00 00 cmpl $0x3e7,0xfffffffffffffff4(%rbp)

4006cb: 7f 11 jg 4006de <main+0xa9>

x = x+j ;

4006cd: 8b 45 f4 mov 0xfffffffffffffff4(%rbp),%eax

4006d0: 48 8d 55 fc lea 0xfffffffffffffffc(%rbp),%rdx

4006d4: 01 02 add %eax,(%rdx)

4006d6: 48 8d 45 f4 lea 0xfffffffffffffff4(%rbp),%rax

4006da: ff 00 incl (%rax)

4006dc: eb e6 jmp 4006c4 <main+0x8f>

4006de: 48 8d 45 f8 lea 0xfffffffffffffff8(%rbp),%rax

4006e2: ff 00 incl (%rax)

4006e4: eb ce jmp 4006b4 <main+0x7f>

}

}

printf("total = %d\n",total);

4006e6: 8b 35 28 04 10 00 mov 1049640(%rip),%esi # 500b14 <total>

# signal\_O2

int total=0;

void handle(int signo)

{

printf("%u\n",total);

4005f0: 8b 35 ee 04 10 00 mov 1049838(%rip),%esi # 500ae4 <total>

4005f6: 53 push %rbx

4005f7: 31 c0 xor %eax,%eax

4005f9: 89 fb mov %edi,%ebx

4005fb: bf bc 07 40 00 mov $0x4007bc,%edi

400600: e8 d3 fe ff ff callq 4004d8 <printf@plt>

if (signo == SIGALRM)

400605: 83 fb 0e cmp $0xe,%ebx

400608: 74 08 je 400612 <handle+0x22>

{

printf("Total = %d\n",total);

exit(0);

}

total++;

40060a: ff 05 d4 04 10 00 incl 1049812(%rip) # 500ae4 <total>

400610: 5b pop %rbx

400611: c3 retq

400612: 8b 35 cc 04 10 00 mov 1049804(%rip),%esi # 500ae4 <total>

400618: bf c0 07 40 00 mov $0x4007c0,%edi

40061d: 31 c0 xor %eax,%eax

40061f: e8 b4 fe ff ff callq 4004d8 <printf@plt>

400624: 31 ff xor %edi,%edi

400626: e8 dd fe ff ff callq 400508 <exit@plt>

40062b: 66 data16

40062c: 66 data16

40062d: 90 nop

40062e: 66 data16

40062f: 90 nop

0000000000400630 <main>:

}

int main ()

{

400630: 48 83 ec 28 sub $0x28,%rsp

unsigned x=0;

int i,j;

struct sched\_param param;

struct itimerval val;

val.it\_interval.tv\_sec = 0;

val.it\_interval.tv\_usec = 10000; // 10 ms

val.it\_value.tv\_sec = 0;

val.it\_value.tv\_usec = 10000; // 10 ms

setitimer(ITIMER\_VIRTUAL, &val, NULL);

400634: 31 d2 xor %edx,%edx

400636: bf 01 00 00 00 mov $0x1,%edi

40063b: 48 89 e6 mov %rsp,%rsi

40063e: 48 c7 04 24 00 00 00 movq $0x0,(%rsp)

400645: 00

400646: 48 c7 44 24 08 10 27 movq $0x2710,0x8(%rsp)

40064d: 00 00

40064f: 48 c7 44 24 10 00 00 movq $0x0,0x10(%rsp)

400656: 00 00

400658: 48 c7 44 24 18 10 27 movq $0x2710,0x18(%rsp)

40065f: 00 00

400661: e8 92 fe ff ff callq 4004f8 <setitimer@plt>

alarm(10);

400666: bf 0a 00 00 00 mov $0xa,%edi

40066b: 31 c0 xor %eax,%eax

40066d: e8 a6 fe ff ff callq 400518 <alarm@plt>

signal(SIGALRM,handle);

400672: be f0 05 40 00 mov $0x4005f0,%esi

400677: bf 0e 00 00 00 mov $0xe,%edi

40067c: e8 67 fe ff ff callq 4004e8 <signal@plt>

signal(SIGVTALRM,handle);

400681: be f0 05 40 00 mov $0x4005f0,%esi

400686: bf 1a 00 00 00 mov $0x1a,%edi

40068b: e8 58 fe ff ff callq 4004e8 <signal@plt>

while (total < 100 )

{

for (i=0; i<BIGNUM; i++)

400690: 8b 0d 4e 04 10 00 mov 1049678(%rip),%ecx # 500ae4 <total>

400696: 83 f9 63 cmp $0x63,%ecx

400699: 7f 1a jg 4006b5 <main+0x85>

40069b: 31 d2 xor %edx,%edx

{

for (j=0; j<BIGNUM; j++)

40069d: 31 c0 xor %eax,%eax

40069f: 90 nop

4006a0: ff c0 inc %eax

4006a2: 3d e7 03 00 00 cmp $0x3e7,%eax

4006a7: 7e f7 jle 4006a0 <main+0x70>

4006a9: ff c2 inc %edx

4006ab: 81 fa e7 03 00 00 cmp $0x3e7,%edx

4006b1: 7e ea jle 40069d <main+0x6d>

4006b3: eb e1 jmp 400696 <main+0x66>

x = x+j ;

}

}

printf("total = %d\n",total);

4006b5: 8b 35 29 04 10 00 mov 1049641(%rip),%esi # 500ae4 <total>

4006bb: bf cc 07 40 00 mov $0x4007cc,%edi

4006c0: 31 c0 xor %eax,%eax

4006c2: e8 11 fe ff ff callq 4004d8 <printf@plt>

}

4006c7: 48 83 c4 28 add $0x28,%rsp

4006cb: c3 retq

4006cc: 90 nop

4006cd: 90 nop