

# 互斥锁测试

在两个核进行通信的过程中常常需要对共享资源进行保护，通过互斥锁的方式可以使得在操作共享资源的同时防止另外一个核同时对该共享资源访问。下面就来展示有无互斥锁的测试对比。

无互斥锁：

Debug Control | Project Explo... | Remote Syste... | Commands | History | Scripts | Linked: cpu0

cpu0 connected  
Cortex-A9\_0 #1 stopped (SVC)  
cpu1 connected  
Cortex-A9\_1 #1 stopped (SVC)

In memcmp.c  
Unable to read source file /home/hn/WJ\_TEST/newlib/newlib-1.20.0.  
S:0x1E00437C 104,0 LDRB r4,[r0,#0]  
wait  
finish  
Execution stopped in SVC mode at S:0x1E001448  
In main.c  
S:0x1E001448 66,0 if (memcmp(ptr+i, val, 1))  
wait  
finish  
Execution stopped in SVC mode at S:0x1E001648  
S:0x1E001648 208,0 ret = verify((u8 \*)p, (u8 \*)&cnt, 16);

Status: connected  
Command: Press (Alt+/) for Content Assist  
Submit

main.c | main.c

```
189 : : "memory", "cc");
190
191 #if 1
192 /*
193  * this part is for test without mutex and semaphore
194  */
195 u32 *p = 0x20000000; // 定义共享资源地址
196 u32 cnt = 0;
197 u32 wrong_cnt = 0;
198 s32 ret;
199 while (1) {
200     cnt++;
201     // led17_blink();
202     // yeshen_mdelay_nobreak(200);
203
204     /* 定义共享资源地址+16清除计数 清除资源 */
205     memset(p, cnt & 0xff, 16);
206
207     /* 清除资源+16清除计数 清除资源 */
208     ret = verify((u8 *)p, (u8 *)&cnt, 16);
209     if (ret) {
210         wrong_cnt++;
211         uartprintf("core0 wrong cnt %d\r\n", wrong_cnt);
212     }
213 }
214 #endif
215
216
```

Locals

Name	Value	Type
i	Optimised away	s32
j	Optimised away	s32
cpu_flag	0	volatile u32*
p	0x20000000	u32*
cnt	13095432	u32
wrong_cnt	1079	u32
ret	0	s32

Disassembly

Address	Opcode	Disassembly
S:0x1E001638	E24B3018	SUB r3, r11, #0x18
S:0x1E00163C	E1A01003	MOV r1, r3
S:0x1E001640	E3A02010	MOV r2, #0x10
S:0x1E001644	E0FFFF6F	BL verify ; 0x1E001648
S:0x1E001648	STR	r0, [r11, #-0x10]
S:0x1E00164C	E51B3010	LDR r3, [r11, #-0x10]
S:0x1E001650	E3530000	CMP r3, #0
S:0x1E001654	0AFFFFEC	BEQ {pc} -0x48 ; 0x1E001648
S:0x1E001658	E51B3008	LDR r3, [r11, #-8]
S:0x1E00165C	E2833001	ADD r3, r3, #1
S:0x1E001660	E50B3008	STR r3, [r11, #-8]
S:0x1E001664	E30C01B4	MOV r0, #0xc1b4
S:0x1E001668	E3410E00	MOVT r0, #0x1e00

App Console | Target Cons... | Error Log

Starting debug server  
Waiting for debug server to start accepting connection  
debug server started successfully

The screenshot displays a multi-processor debugging environment. The 'Debug Control' panel on the left shows two CPUs connected: 'cpu0' and 'Cortex-A9\_0 #1', both in a stopped state. The 'Commands' panel in the top center shows the execution flow, including 'finish', 'wait', 'continue', 'interrupt', and 'finish' commands, with execution stopped in SVC mode at address S:0x1F001538. The 'Locals' panel on the top right shows variables: 'i' and 'j' are optimized away; 'cpu\_flag' is 1; 'p' is 0x20000000; 'cnt' is 4285694136; 'wrong\_cnt' is 1089; and 'ret' is 0. The 'Disassembly' panel on the bottom right shows assembly instructions for the current instruction address S:0x1F001538, including 'SUB', 'MOV', 'MOV', 'BL', 'STR', 'LDR', 'CMP', 'BEQ', 'LDR', 'ADD', 'STR', and 'MOVT'. The main code editor at the bottom shows the C code for a memory verification test, including a while loop that increments 'cnt' and checks if the value at 'p' matches 'cnt'.

```
156
157 #if 1
158 /*
159  * this part is for test without mutex and semaphore
160  */
161
162 u32 *p = 0x20000000; // 读写数据地址
163 u32 cnt = 0xffffffff;
164 u32 wrong_cnt = 0;
165 s32 ret;
166 while (1) {
167     cnt--;
168     // led15_blink();
169     // yesheh_mdelay_nobreak(100);
170
171     /* 读写数据地址+16 读写数据 */
172     memset(p, cnt & 0xff, 16);
173
174     /* 读写数据+16 读写数据 */
175     ret = verify((u8 *)p, (u8 *)&cnt, 16);
176     if (ret) {
177         wrong_cnt++;
178         uartprintf("core1 wrong cnt %d\n", wrong_cnt);
179     }
180 }
181 #endif
182
```

以上展示的是两个核在没有任何同步保护机制的情况下对同一块共享内存进行写入然后读取验证。wrong\_cnt记录验证错误的次数，也就是读出来的值不是之前写进去的值的次数，从图中看到两个核在进行一定次数的测试后分别发生了1079次和1089次的验证数据错误，因此没有互斥锁等同步机制的情况下访问共享资源是不安全的。

**有互斥锁：**

**阻塞版：**

Debug Console: Status: connected

Commands: S:0x1E001654 240,0 if (ret) {  
wait  
next  
Execution stopped in SVC mode at S:0x1E00167C  
S:0x1E00167C 246,0 unlock\_mutex(mutex);  
wait  
next  
Execution stopped in SVC mode at S:0x1E001684  
S:0x1E001684 247,0 }  
wait  
next  
Execution stopped in SVC mode at S:0x1E001610  
S:0x1E001610 228,0 cnt++;

Locals: i, j, cpu\_flag, p, cnt, wrong\_cnt, ret

```

217 #if 1
218 /*
219  * this part is for mutex and semaphore test and use mutex with block
220  */
221 u32 *p = 0x20000000; //接受数据寄存器地址
222 u64 cnt = 0;
223 u32 wrong_cnt = 0;
224 s32 ret;
225 void *mutex;
226 mutex = init_mutex(); //创建 互斥量
227 while (1) {
228     cnt++;
229     //led17_blink();
230     //yeshen_mdelay_nobreak(100);
231
232     /* 等待信号 */
233     lock_mutex(mutex);
234
235     /* 将数据寄存器地址*16清0并写入数据 */
236     memset(p, cnt & 0xff, 16);
237
238     /* 将数据寄存器*16清 0并写入数据 */
239     ret = verify((u8 *)p, (u8 *)&cnt, 16);
240     if (ret) {
241         wrong_cnt++;
242         uartprintf("core0 wrong cnt %d\r\n", wrong_cnt);
243     }
244
245     /* 退出 */
246     unlock_mutex(mutex);
247 }
248 #endif
249
250

```

Disassembly: S:0x1E001600 E3A03000 MOV r3, #0  
S:0x1E001604 E50B3008 STR r3, [r11, #-8]  
S:0x1E001608 E8000834 BL init\_mutex ; 0x1  
S:0x1E00160C E50B0010 STR r0, [r11, #-0x10]  
S:0x1E001610 E51B3014 LDR r3, [r11, #-0x1c]  
S:0x1E001614 E2833001 ADD r3, r3, #1  
S:0x1E001618 E51B0010 STR r3, [r11, #-0x1c]  
S:0x1E00161C E80007D6 BL lock\_mutex ; 0x1  
S:0x1E001620 E51B301C LDR r3, [r11, #-0x1c]  
S:0x1E001628 E6EF3073 UXTB r3, r3  
S:0x1E00162C E51B000C LDR r0, [r11, #-0xc]  
S:0x1E001630 E1A01003 MOV r1, r3

App Console: Starting debug server  
Waiting for debug server to start accepting connection  
Debug server started successfully

Debug Console: Status: connected

Commands: In memcmp.c  
Unable to read source file /home/hn/WJ\_TEST/newlib/newlib-1.20.0/  
S:0x1F003530 112,0 POP {r4,r5}  
wait  
finish  
Execution stopped in SVC mode at S:0x1F0013EC  
In main.c  
S:0x1F0013EC 67,0 if (memcmp(ptr+1, val, 1))  
wait  
finish  
Execution stopped in SVC mode at S:0x1F001540  
S:0x1F001540 207,0 ret = verify((u8 \*)p, (u8 \*)&cnt, 16);

Locals: i, j, cpu\_flag, p, cnt, wrong\_cnt, ret

```

185 #if 1
186 /*
187  * this part is for mutex and semaphore test and use mutex with block
188  */
189 u32 *p = 0x20000000; //接受数据寄存器地址
190 u64 cnt = 0;
191 u32 wrong_cnt = 0;
192 s32 ret;
193 void *mutex;
194 mutex = init_mutex(); //创建 互斥量
195 while (1) {
196     cnt++;
197     //led15_blink();
198     //yeshen_mdelay_nobreak(100);
199
200     /* 等待信号 */
201     lock_mutex(mutex);
202
203     /* 将数据寄存器地址*16清0并写入数据 */
204     memset(p, cnt & 0xff, 16);
205
206     /* 将数据寄存器*16清 0并写入数据 */
207     ret = verify((u8 *)p, (u8 *)&cnt, 16);
208     if (ret) {
209         wrong_cnt++;
210         uartprintf("core1 wrong cnt %d\r\n", wrong_cnt);
211     }
212
213     /* 退出 */
214     unlock_mutex(mutex);
215 }
216 #endif
217
218

```

Disassembly: S:0x1F001530 E24B301C SUB r3, r11, #0x1c  
S:0x1F001534 E1A01003 MOV r1, r3  
S:0x1F001538 E3A02010 MOV r2, #0x10  
S:0x1F00153C E8FFFF9A BL verify ; 0x1F001  
S:0x1F001540 E51B3014 LDR r3, [r11, #-0x14]  
S:0x1F001544 E3530000 CMP r3, #0  
S:0x1F001548 0A000006 BEQ {pc}+0x20 ; 0x1  
S:0x1F001550 E51B3008 LDR r3, [r11, #-8]  
S:0x1F001554 E2833001 ADD r3, r3, #1  
S:0x1F001558 E50B3008 STR r3, [r11, #-8]  
S:0x1F00155C E30B033C MOV r0, #0xb33c  
S:0x1F001560 E3410F00 MOV r0, #0xf000

App Console: A suitable Debug Server is already running

上图是两个核利用互斥锁对共享内存进行读写的测试，每个核在操作共享内存时都要通过init\_mutex初始化互斥锁，第一个调用init\_mutex的线程返回一个开的锁，第二个调用init\_mutex的线程返回锁的地址，锁是否打开由第一个线程是否已经释放锁来决定。每个核在操作共享内存时候通过lock\_mutex锁定，操作完后通过unlock\_mutex释放锁，从图中可以发现两个核的测试中的wrong\_cnt一直为0，即一直没有出错，因此本文实现的互斥锁能很好的保护双核通信中共享资源的访问。

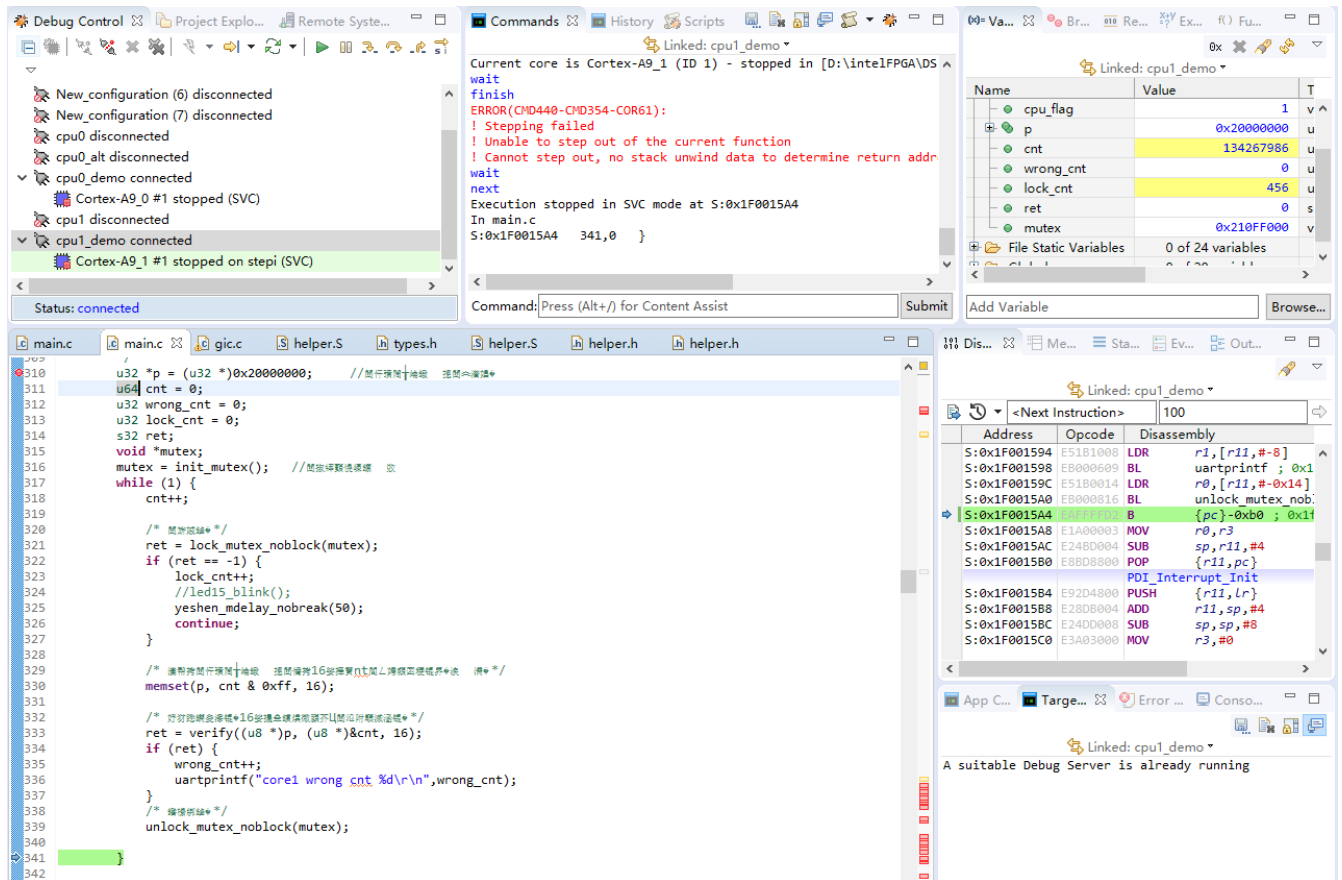
## 非阻塞版：

上述版本的互斥锁是阻塞版的，就是当本线程调用lock\_mutex获取互斥锁时发现锁已经被占用的时候会阻塞在临界区，直到锁被其他线程释放后lock\_mutex获取了互斥锁后才能返回访问共享资源。在这阻塞期间本线程什么都不能做，本文提供了另外一种非阻塞方案，可以让用户选择在锁被占用的情况下是否做其他工作。

The screenshot displays a debugger window with the following components:

- Debug Control:** Shows the status of various components. 'Cortex-A9\_0 #1' and 'Cortex-A9\_1 #1' are both 'stopped (SVC)'.
- Commands:** Shows the execution flow. The program is currently at address S:0x1E0016C8, executing a 'ret' instruction.
- Variables:** A table of variables for 'cpu0\_demo' is shown:

Name	Value	Type
cpu_flag	0	u
p	0x20000000	u
cnt	37575854	u
wrong_cnt	0	u
lock_cnt	1635	u
ret	0	s
mutex	0x2108FF00	s
- Disassembly:** A table of instructions is shown, including 'SUB', 'MOV', 'BL', 'STR', 'LDR', 'CMP', 'BEQ', 'LDR', 'ADD', 'STR', 'MOV', and 'MOVT'.
- Source Code:** The C code for 'main.c' is visible, showing the initialization of 'mutex' and the 'while' loop that tests the lock.



非阻塞版和阻塞版其中行为不同的只是获取锁的操作，初始化和释放锁的行为一致。而获取锁在不成功的时候直接返回-1,表示锁已经被占用，返回0表示获取锁成功，用户可以根据返回值作出相应的动作，不必一直阻塞直到锁在其他地方释放。从图中看到两个核的测试中wrong\_cnt也是一直为0，说明共享资源访问没有冲突，而另外用一个lock\_cnt的变量记录获取锁时被占用的次数，两个核分别有1635和456次申请获取锁时锁被占用。

## 总结

在没有同步机制保护的情况下双核同时访问共享资源会导致数据错乱冲突，在有互斥锁保护的情况下双核可以同时访问共享资源而不发生冲突，且根据应用的需求提供了阻塞版和非阻塞版两种互斥机制。