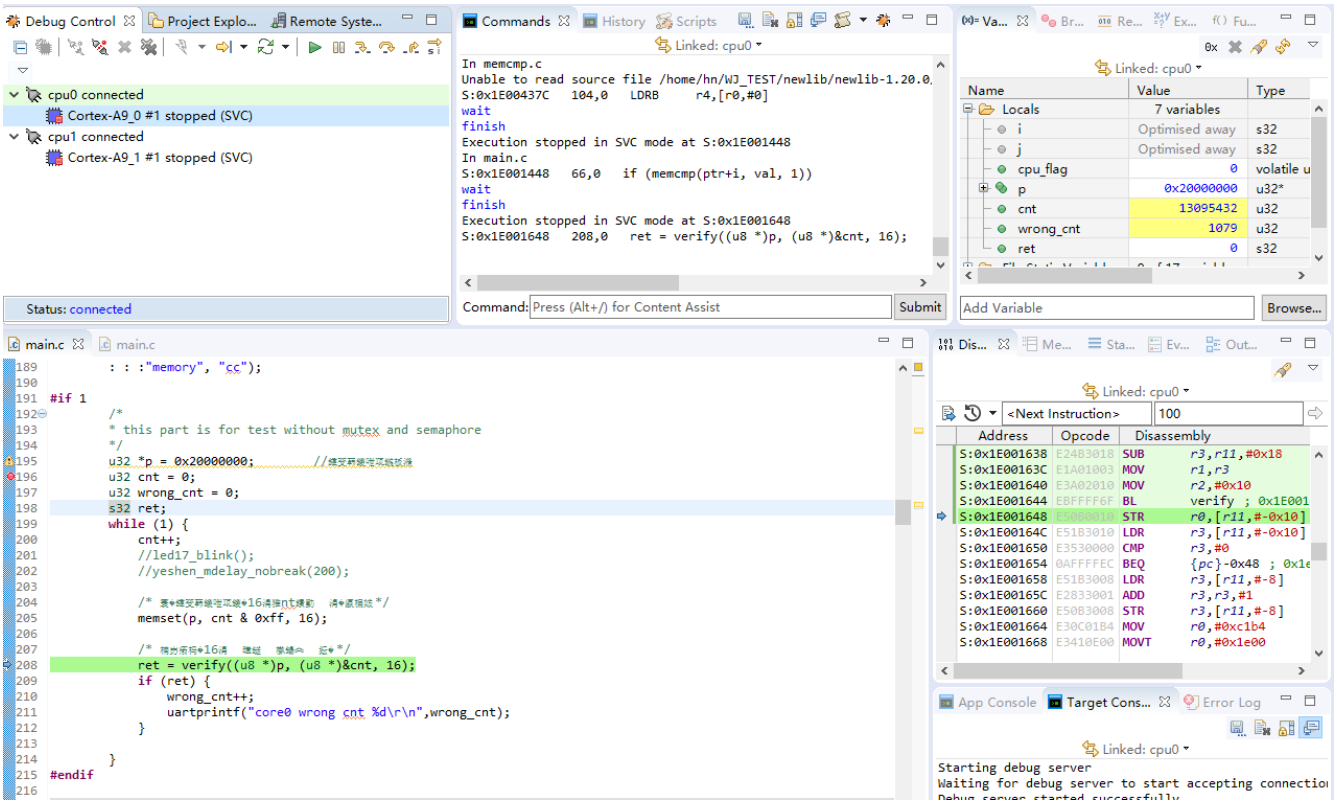
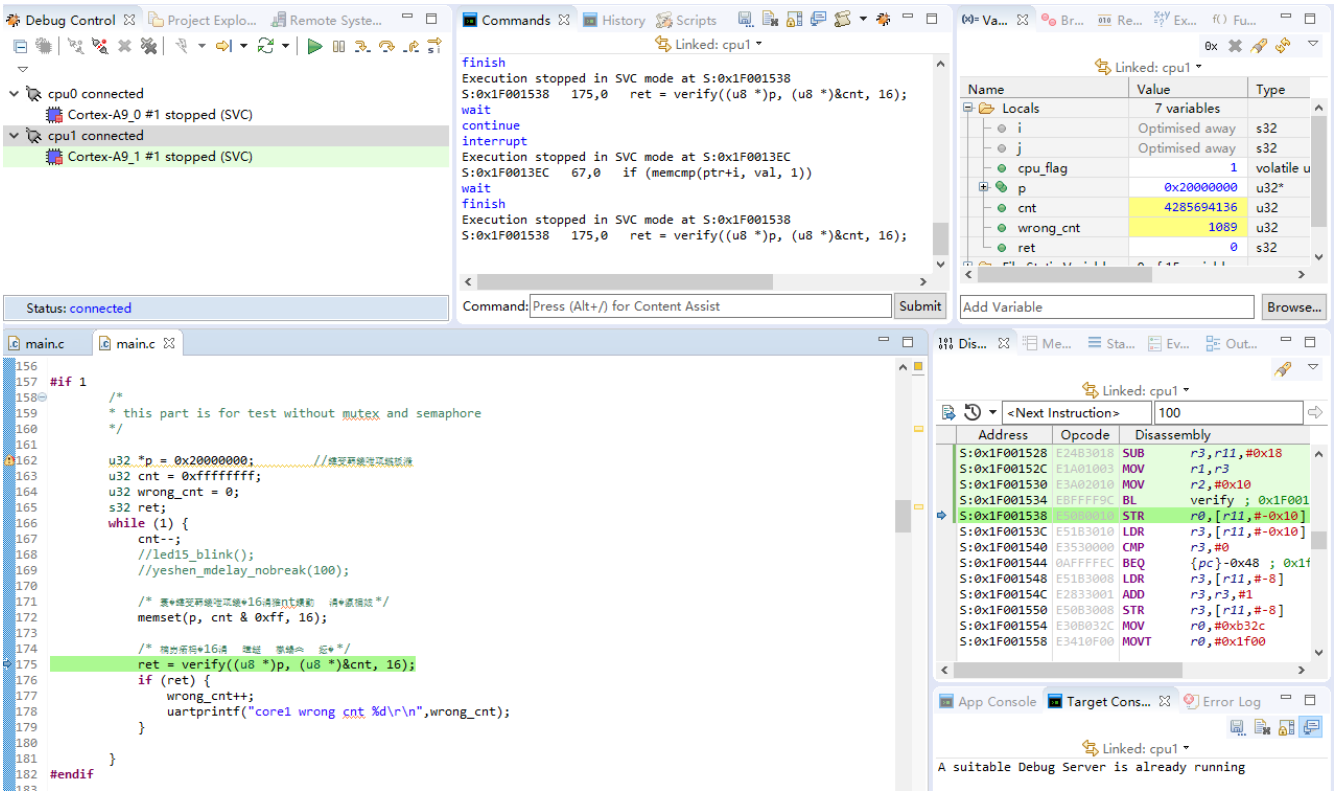
互斥锁测试

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：

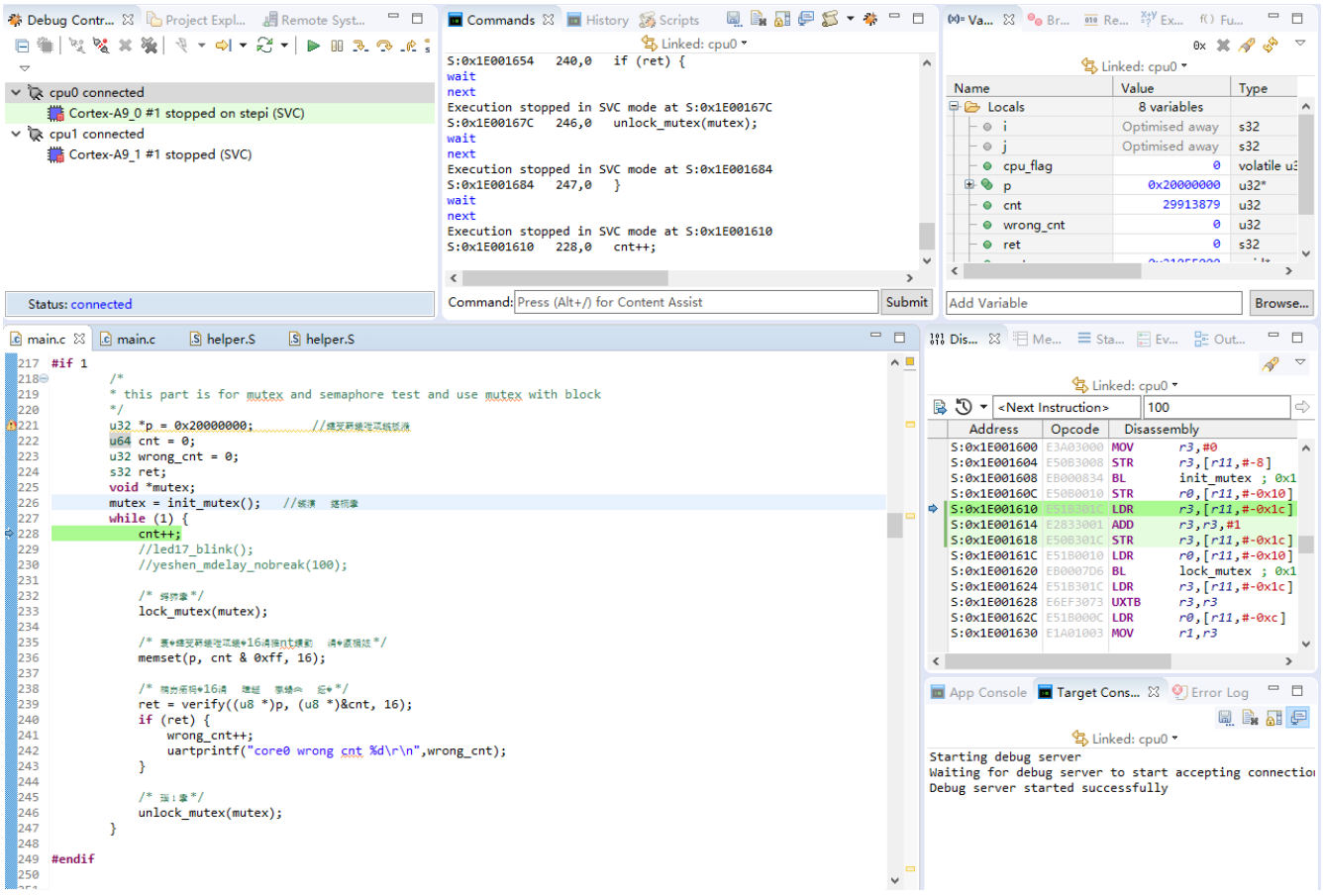


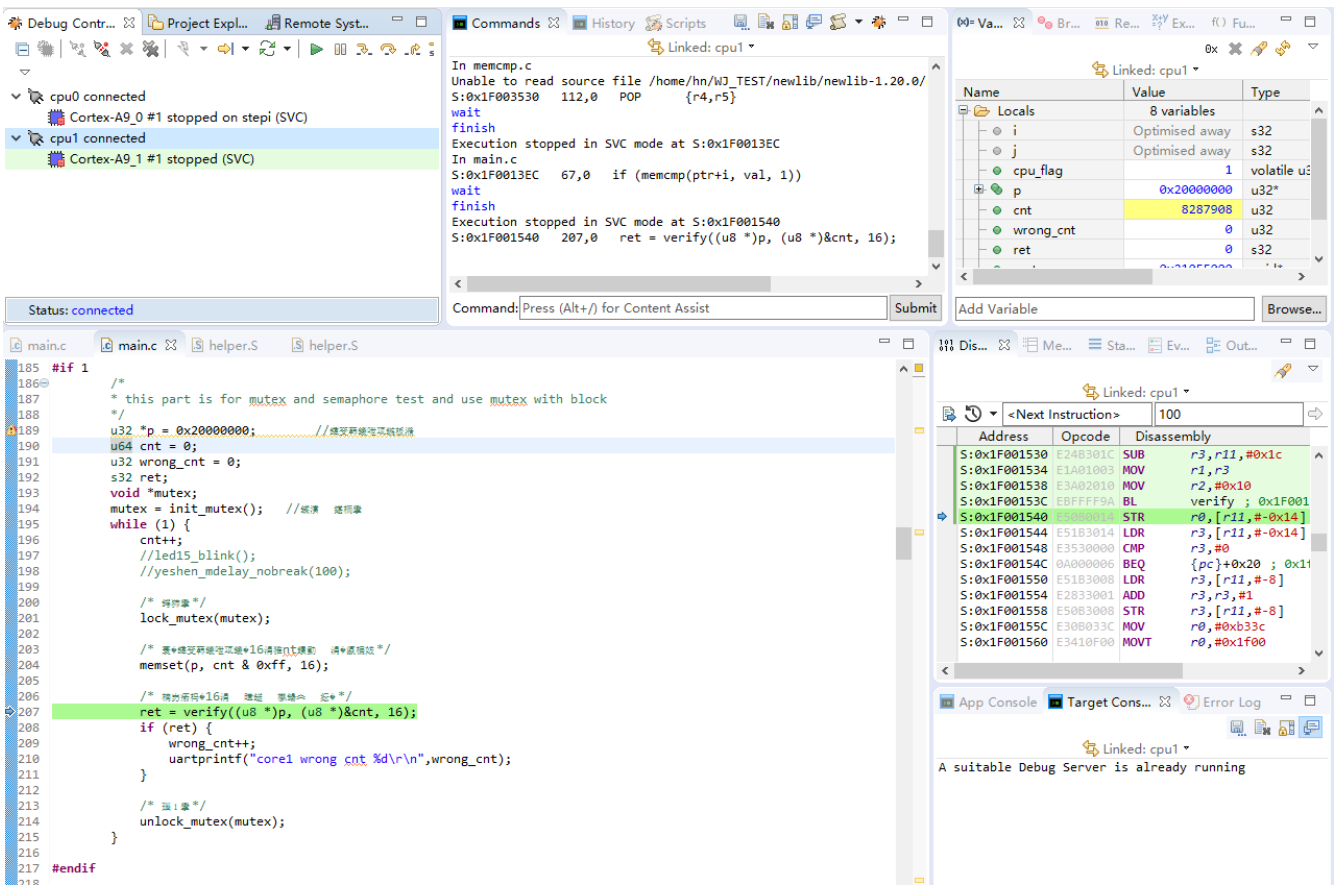


以上展示的是两个核在没有任何同步保护机制的情况下对同一块共享内存进行写入然后读取验证。wrong_cnt记录验证错误的次数，也就是读出来的值不是之前写进去的值的次数，从图中看到两个核在进行一定次数的测试后分别发生了1079次和1089次的验证数据错误，因此没有互斥锁等同步机制的情况下访问共享资源是不安全的。

有互斥锁：

阻塞版：

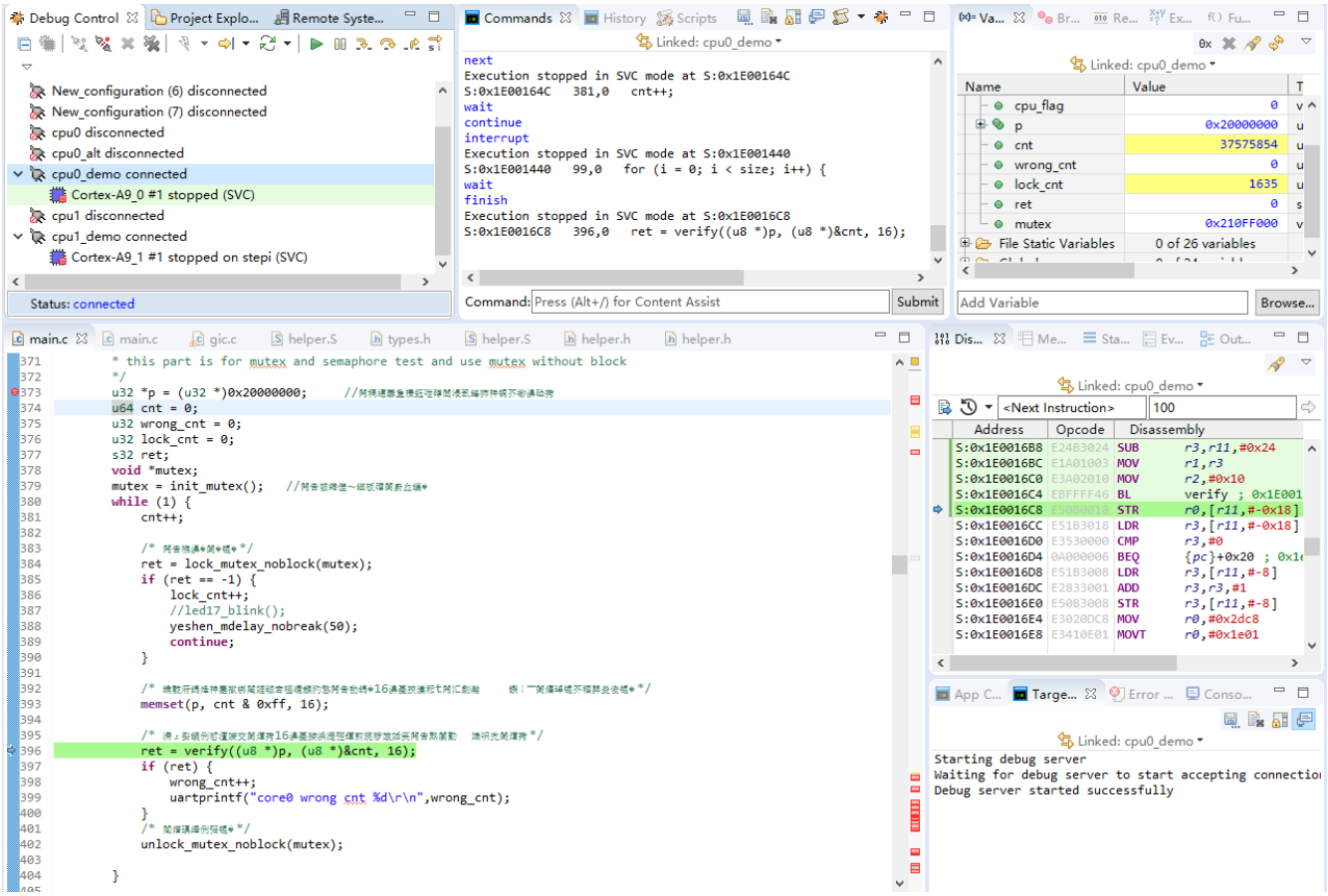

Debug Contr...
Project Expl...
Remote Syst...
cpu0 connected
Cortex-A9_0 #1 stopped on stepi (SVC)
cpu1 connected
Cortex-A9_1 #1 stopped (SVC)
Status: connected
Commands
History
Scripts
Linked: cpu0
S:0x1E001654 240,0 if (ret) {
wait
next
Execution stopped in SVC mode at S:0x1E00167C
S:0x1E00167C 246,0 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++;
Command:
Press (Alt+/) for Content Assist
Submit
Va...
Br...
Re...
Ex...
Fu...
Linked: cpu0
Name
Value
Type
Locals
8 variables
i
Optimised away
s32
j
Optimised away
s32
cpu_flag
0
volatile u3
p
0x20000000
u32*
cnt
29913879
u32
wrong_cnt
0
u32
ret
0
s32
Add Variable
Browse...
main.c
main.c
helper.S
helper.S
#if 1
/*
* this part is for mutex and semaphore test and use mutex with block
*/
u32 *p = 0x20000000; //
u64 cnt = 0;
u32 wrong_cnt = 0;
s32 ret;
void *mutex;
mutex = init_mutex(); //
while (1) {
cnt++;
//led17_blink();
//yeshen_mdelay_nobreak(100);
/*
lock_mutex(mutex);
/*
memset(p, cnt & 0xff, 16);
/*
ret = verify((u8 *)p, (u8 *)&cnt, 16);
if (ret) {
wrong_cnt++;
uartprintf("core0 wrong cnt %d\r\n",wrong_cnt);
}
/*
unlock_mutex(mutex);
}
#endif
Dis...
Me...
Sta...
Ev...
Out...
Linked: cpu0
<Next Instruction>
100
Address
Opcode
Disassembly
S:0x1E001600 E3A03000 MOV r3,#0
S:0x1E001604 E50B3008 STR r3,[r11,#-8]
S:0x1E001608 EB000834 BL init_mutex ; 0x1
S:0x1E00160C E50B0010 STR r0,[r11,#-0x10]
S:0x1E001610 E51B301C LDR r3,[r11,#-0x1c]
S:0x1E001614 E2833001 ADD r3,r3,#1
S:0x1E001618 E50B301C STR r3,[r11,#-0x1c]
S:0x1E00161C E51B0010 LDR r0,[r11,#-0x10]
S:0x1E001620 EB0007D6 BL lock_mutex ; 0x1
S:0x1E001624 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
Target Cons...
Error Log
Linked: cpu0
Starting debug server
Waiting for debug server to start accepting connectio
Debug server started successfully

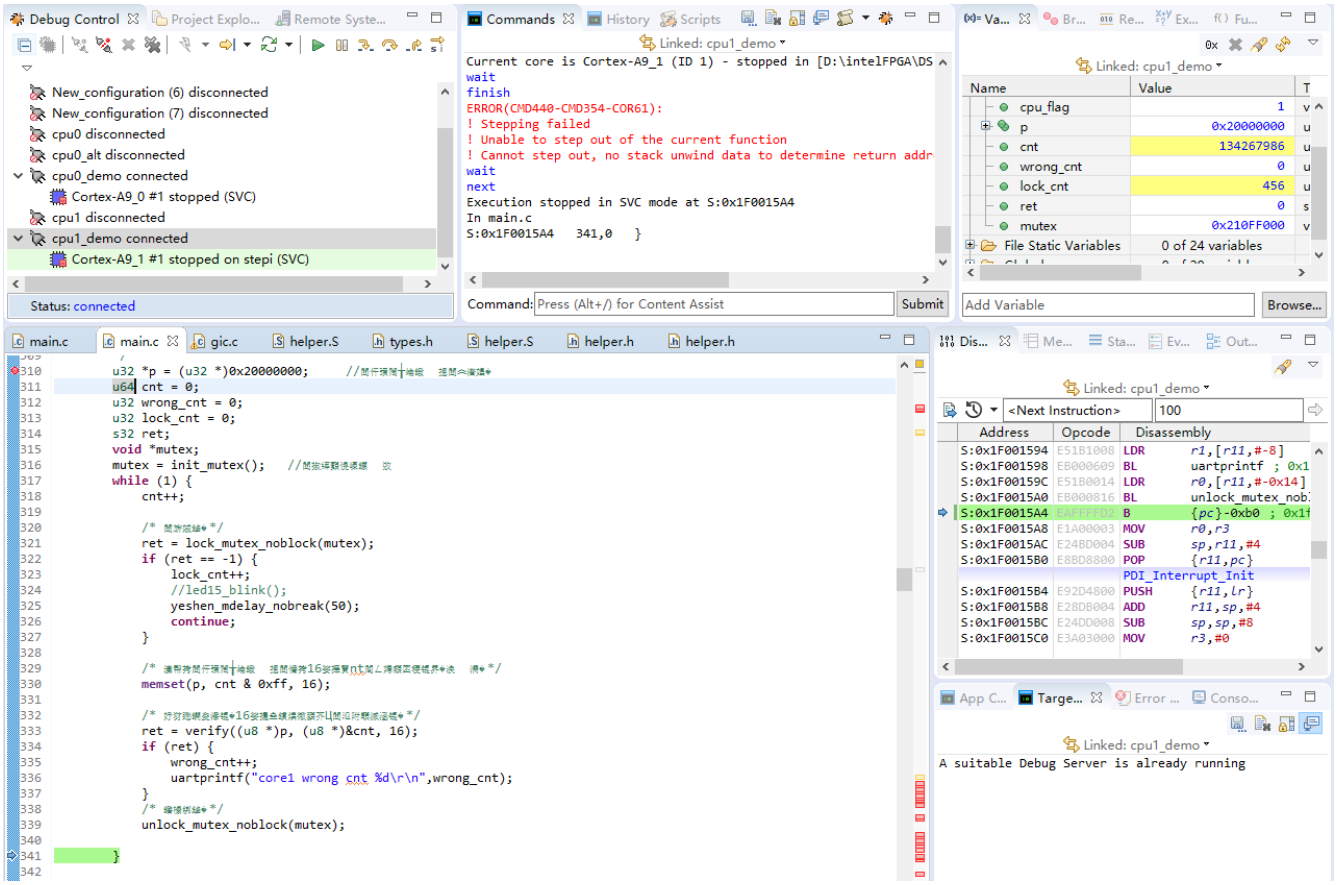

Debug Contr...
Project Expl...
Remote Syst...
cpu0 connected
Cortex-A9_0 #1 stopped on stepi (SVC)
cpu1 connected
Cortex-A9_1 #1 stopped (SVC)
Status: connected
Commands
History
Scripts
Linked: cpu1
In memcmp.c
Unable to read source file /home/hn/WJ_TEST/newlib/newlib-1.20.0/
S:0x1F003530 112,0 POP {r4,r5}
wait
finish
Execution stopped in SVC mode at S:0x1F0013EC
In main.c
S:0x1F0013EC 67,0 if (memcmp(ptr+i, val, 1))
wait
finish
Execution stopped in SVC mode at S:0x1F001540
S:0x1F001540 207,0 ret = verify((u8 *)p, (u8 *)&cnt, 16);
Command:
Press (Alt+/) for Content Assist
Submit
Va...
Br...
Re...
Ex...
Fu...
Linked: cpu1
Name
Value
Type
Locals
8 variables
i
Optimised away
s32
j
Optimised away
s32
cpu_flag
1
volatile u3
p
0x20000000
u32*
cnt
8287908
u32
wrong_cnt
0
u32
ret
0
s32
Add Variable
Browse...
main.c
main.c
helper.S
helper.S
#if 1
/*
* this part is for mutex and semaphore test and use mutex with block
*/
u32 *p = 0x20000000; //
u64 cnt = 0;
u32 wrong_cnt = 0;
s32 ret;
void *mutex;
mutex = init_mutex(); //
while (1) {
cnt++;
//led15_blink();
//yeshen_mdelay_nobreak(100);
/*
lock_mutex(mutex);
/*
memset(p, cnt & 0xff, 16);
/*
ret = verify((u8 *)p, (u8 *)&cnt, 16);
if (ret) {
wrong_cnt++;
uartprintf("core1 wrong cnt %d\r\n",wrong_cnt);
}
/*
unlock_mutex(mutex);
}
#endif
Dis...
Me...
Sta...
Ev...
Out...
Linked: cpu1
<Next Instruction>
100
Address
Opcode
Disassembly
S:0x1F001530 E24B301C SUB r3,r11,#0x1c
S:0x1F001534 E1A01003 MOV r1,r3
S:0x1F001538 E3A02010 MOV r2,#0x10
S:0x1F00153C EBFFFF9A BL verify ; 0x1F001
S:0x1F001540 E50B0014 STR r0,[r11,#-0x14]
S:0x1F001544 E51B3014 LDR r3,[r11,#-0x14]
S:0x1F001548 E3530000 CMP r3,#0
S:0x1F00154C 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 MOVT r0,#0x1f00
App Console
Target Cons...
Error Log
Linked: cpu1
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：

上诉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。





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。