SpinLock Usage in Loop

List of Lock in Loop

./core/kernel/kthread.c:107:partition_t *CreatePartition(struct xmcPartition *cfg) {

./core/objects/ttnocports.c:378:xm_s32_t __VBOOT SetupComm(void) {
./core/objects/commports.c:1076:xm_s32_t __VBOOT SetupComm(void) {

./core/include/kthread.h:170:static inline void SetPartitionHwIrqPending(partition_t *p, xm_s32_t irq) {
./core/include/kthread.h:205:static inline int ArePartitionExtIrqPendingSet(partition_t *p, xm_s32_t irq) {
./core/include/kthread.h:239:static inline void SetPartitionExtIrqPending(partition_t *p, xm_s32_t irq) {

./core/include/sched.h:158:static inline void SUSPEND_PARTITION(xmId_t id) {
./core/include/sched.h:170:static inline void RESUME_PARTITION(xmId_t id) {
./core/include/sched.h:198:static inline void HALT_PARTITION(xmId_t id) {

./core/kernel/mmu/physmm.c:189:void PmmResetPartition(partition_t *p) {

./core/kernel/mmu/vmmap.c:127:xmAddress_t SetupPageTable(partition_t *p, xmAddress_t pgTb, xmSize_t size) {
./core/kernel/mmu/hypercalls.c:161:static void SetPtd(xmAddress_t pAddr, struct physPage *pagePtd, xm_u32_t type) {
./core/kernel/mmu/hypercalls.c:188:static void SetPte(xmAddress_t pAddr, struct physPage *pagePte, xm_u32_t type) {
./core/kernel/mmu/hypercalls.c:123:static void UnsetPtd(xmAddress_t pAddr, struct physPage *pagePtd, xm_u32_t type) {
./core/kernel/mmu/hypercalls.c:142:static void UnsetPte(xmAddress_t pAddr, struct physPage *pagePte, xm_u32_t type) {

SpinLock Implementation

SpinLock

```
static inline void __ArchSpinLock(archSpinLock_t *lock) {
#ifdef CONFIG_SMP
   __asm__ __volatile__("\n1:\n\t" \
                  "ldstuba [%0] 1, %%g2\n\t" /* ASI_LEON23_DCACHE_MISS */ \
                  "orcc  %%g2, 0x0, %%g0\n\t" \
                  "bne,a2f\n\t" \
                  "ldub [%0], %%g2\n\t" \
                  ".subsection 2\n" \
                  "2:\n\t" \
                  "orcc %%g2, 0x0, %%g0\n\t" \
                  "bne,a 2b\n\t" \
                  "ldub [%0], %%g2\n\t" \
                  "b,a 1b\n\t" \
                  "nop\n\t" \
                  ".previous\n" \
                  : /* no outputs */ \
                  : "r" (&lock->lock) \
                  : "g2", "memory", "cc");
#endif
}
```

SpinUnlock

```
static inline void __ArchSpinUnlock(archSpinLock_t *lock) {
#ifdef CONFIG_SMP
   __asm__ __volatile__("stb %%g0, [%0]" : : "r" (&lock->lock) : "memory");
#endif
}
```

Who Uses SpinLock

$ grep -lr "SpinLock" .

./core/drivers/leon_uart.c # → checked; not likely to have “Lock in Loop”

./core/kernel/mmu/physmm.c # → Case 3

~~./core/kernel/arch/leon_timers.c~~ # → checked; all commented

./core/objects/console.c # → checked; less likely to have “Lock in Loop”

./core/objects/ttnocports.c # → Case 1

./core/objects/commports.c # → Case 1

./core/klibc/stdio.c # → checked; not likely to have

~~./core/include/spinlock.h~~

./core/include/list.h # → Case 1

~~./core/include/arch/spinlock.h~~

./core/include/physmm.h # → Case 3

./core/include/kthread.h # → Case 2

./core/include/logstream.h # → Case 4

Case 1: List

```
struct dynList {
    struct dynListNode *head;
    xm_s32_t noElem;
    spinLock_t lock;
};

static inline void DynListInit(struct dynList *l) {
    l->lock=SPINLOCK_INIT;
    SpinLock(&l->lock);
    l->noElem=0;
    l->head=0;
    SpinUnlock(&l->lock);
}

static inline xm_s32_t DynListInsertHead(struct dynList *l, struct dynListNode *e) {
    SomethingHere()
    SpinLock(&l->lock);
    SomethingHere()
    SpinUnlock(&l->lock);
    return 0;
}
```

Similar SpinLock & SpinUnlock for other functions in core/include/spinlock.h
And also for syntax sugar
DYNLIST_FOR_EACH_ELEMENT_BEGIN
DYNLIST_FOR_EACH_ELEMENT_END

Who uses DynList

./core/kernel/ktimer.c # → checked; No “Lock in Loop”
./core/kernel/mmu/physmm.c # → checked; functions are less likely to be called in loop
./core/kernel/kthread.c # → function CreatePartition()
./core/objects/ttnocports.c # → function SetupComm()
./core/objects/commports.c # → function SetupComm()

SetupComm

```
/* create the channels */
for (e=0; e<xmcTab.noCommChannels; e++) {
    switch(xmcCommChannelTab[e].type) {
    case XM_QUEUING_CHANNEL:
        GET_MEMZ(channelTab[e].q.msgPool, sizeof(struct
msg)*xmcCommChannelTab[e].q.maxNoMsgs);
        DynListInit(&channelTab[e].q.freeMsgs);
        DynListInit(&channelTab[e].q.recvMsgs);
        for (i=0; i<xmcCommChannelTab[e].q.maxNoMsgs; i++) {
            GET_MEMZ(channelTab[e].q.msgPool[i].buffer,
xmcCommChannelTab[e].q.maxLength);
            if(DynListInsertHead(&channelTab[e].q.freeMsgs,
&channelTab[e].q.msgPool[i].listNode)) {
                SystemPanic();
            }
        }
    }
}
```

Channel is a union structure, used for representing Sampling Channel, TTNoC Channel or Queuing Channel.
Only Queuing Channel needs a DynList for storing Msg.
GET_MEMZ located maxNoMsgs of msg to channel message pool.
And feed all the dummy msg into freeMsgs DynList

This is a “Lock in Loop” condition.

Case 2: kThread_t Flags & IrqsPend

kthread.c & kthead.h

kthread.c

There is one function called
`static inline kThread_t *AllocKThread(xmId_t id);`
which used **DynListInit()**; and this function is called from **CreatePartition()** function’s loop.

```
for (i=0; i<cfg->noVCpus; i++) {
    SomethingHere();
    p->kThread[i]=k=AllocKThread(PART_VCPU_ID2KID(cfg->id, i));
    SomethingHere();
}
```

kthead.h

core/include/kthread.h
First of all, kThread contains spinLock_t for sure.

Secondly,
// Write to Flags
SetKThreadFlags
ClearKThreadFlags
// Read from Flags
AreKThreadFlagsSet
These 3 functions contain similar format of using SpinLock and SpinUnlock. for example:

```
static inline void SetKThreadFlags(kThread_t *k, xm_u32_t f) {
   SpinLock(&k->ctrl.lock);
   k->ctrl.flags |= f;
   if (k->ctrl.g && k->ctrl.g->partCtrlTab)
      k->ctrl.g->partCtrlTab->flags |= f;
   SpinUnlock(&k->ctrl.lock);
}
```

The reason we need SpinLock when setting kThread Flags is that XtratuM is using Flags to check current thread running status, for example *KTHREAD_HALTED_F*, *KTHREAD_TRAP_PENDING_F*, etc.
Not even flags, HwIrq Pending and ExtIrq Pending signals also need SpinLock to make sure the IRQ is passed to partitions correctly.
So the functions:
SetPartitionHwIrqPending # → has “Lock in Loop”
SetExtIrqPending
ArePartitionExtIrqPendingSet # → has “Lock in Loop”
AreExtIrqPendingSet
SetPartitionExtIrqPending # → has “Lock in Loop”

The “Lock in Loop” conditions happens when it is setting/checking partitions’ IRQ. It is needed to have a for loop like following.

```
for (e = 0; e < p->cfg->noVCpus; e++) {
   FlagOperationHere();
}
```

The flag operations mentioned above are used in other for loop as well, such as core/include/sched.h:l158 SUSPEND_PARTITION; l170 RESUME_PARTITION
Because of noVCpus → nokThread, it needs a for loop to change each kThread.

Case 3: Physical Memory Manager

core/kernel/mmu/physmm.c

void PmmResetPartition(partition_t *p)

```
for (e=0; e<p->cfg->noPhysicalMemoryAreas; e++) {
   SomethingHere();
   if (memRegion->flags&XMC_REG_FLAG_PGTAB)
      for (addr=memArea->startAddr; addr<memArea->startAddr+memArea->size;
addr+=PAGE_SIZE) {
         SomethingHere();
         SpinLock(&page->lock);
         page->type=PPAG_STD;
         page->counter=0;
         SpinUnlock(&page->lock);
         SomethingHere();
      }
}
```

Each page (struct phyPage) contains one spinlock_t

```
struct physPage {
   struct dynListNode listNode;
#ifndef CONFIG_ARCH_MMU_BYPASS
   xmAddress_t vAddr;
#endif
   xm_u32_t mapped:1, unlocked:1, type:3, counter:27;
   spinLock_t lock;
};
```

PmmResetPartition is called from ResetPartition, and before which, all the VCpus are halted

core/include/phymm.h

PPagIncCounter & PPagDecCounter

```
SpinLock(&page->lock);
cnt=page->counter;
page->counter++;
SpinUnlock(&page->lock);
```

Function PPagIncCounter is called from a loop in core/kernel/mmu/vmmap.c: SetupPageTable() and loop in core/kernel/mmu/hypercalls.c: SetPtd() SetPte()

Function PPagDecCounter is called from a loop in core/kernel/mmu/hypercalls.c: UnsetPtd() UnsetPte()

Case 4: LogStream

LogStreamInsert

core/include/logstream.h

```
static inline xm_s32_t LogStreamInsert(struct logStream *lS, void *log) {
    SomethingHere();
    SpinLock(&lS->lock);
    SomethingHere();
    SpinUnlock(&lS->lock);
}
```

Though other LogStream operations are in this format too, this insert function is more likely to be used in a loop.

It is found that this function is called in a loop from function **WriteTrace()**, from core/objects/trace.c

The reason of using SpinLock may because the health monitor is shared by different partitions. Hence, the operations of log need well-controlled.

Summary

XtratuM does have several “Lock in Loop” conditions. Luckily, it is found that SpinLock and SpinUnlock are always used properly. There is always a SpinUnlock() before return or at the corresponding possition of SpinLock().