日曜研究室

技術的な観点から日常を綴ります

[xv6 #35] Chapter 3 – Locking – Code: Locks

テキストの45〜46ページ

本文

xv6は、spinlock構造体としてロックを表現する。
その構造体のなかで重要なフィールドはlockedであり、ロック可能なときにゼロとなり、すでにロックされているときにゼロ以外の値となる。
理論上、xv6は、以下のようなコードを実行することによってロックを獲得しなければならない。

```
01 void
02 acquire(struct spinlock *lk)
03 {
04   for(;;) {
05     if(!lk->locked) {
06       lk->locked = 1;
07       break;
08     }
09   }
10 }
```

残念ながら、この実装は、最近のマルチプロセッサ上では排他を保証しない。
２つ（もしくはそれ以上の）CPUが同時にif文に到達し、lockedがゼロなのを見て、そして両方ともif文の中のブロックを実行することによってロックをつかむ可能性がある。
この時点で、2つの別のCPUがロックを保持し、排他的特性を破ってしまう。
競合状態を避けるために役に立つどころか、このacquireの実装はそれ自身が競合状態を持つ。
この問題は、lockedのチェックとlockedへの代入（if(!lk->locked) と lk->locked = 1;の行）が別の文で行われているところにある。
上記のルーチンを正すため、lockedのチェックとlockedへの代入は、ひとつのアトミックな処理（これ以上分割できない）で実行しなければならない。

spinlock.h

```
01 // Mutual exclusion lock.
02 struct spinlock {
03   uint locked;       // Is the lock held?
```

```
04
05    // For debugging:
06    char *name;         // Name of lock.
07    struct cpu *cpu;    // The cpu holding the lock.
08    uint pcs[10];       // The call stack (an array of program
    counters)
09                        // that locked the lock.
10  };
```

それら2つの行をアトミックに実行するために、xv6はxchgという386の特殊なハードウェア命令に頼る。
ひとつのアトミックな操作で、xchgは、メモリ上の1ワード（ここでは多分32ビットかな）とレジスタの内容を入れ替える。
実際のacquire関数は、ループ内でこのxchg命令を繰り返す。
ループ内で毎回lk->lockedを読み取りながら、アトミックにそれに1をセットする。（acquire関数のwhile文のところ）
すでにロックが確保されていたら、lk->lockedはすでに1になっているだろう。
そしてxchgは1を返し、ループは継続される。
xchgが0を返した場合、acquireはロックの獲得に成功した状態であり（0だったlockedが1になった状態）、それによってループは終わる。
一度ロックが確保されたら、acquireは、デバッグのため、ロックを確保したCPUとスタックトレースを記録する。
プロセスが、ロックを確保しておきながら解放するのを忘れたとき、この情報は、原因を特定するのに役立つ。
それらデバッグ情報用のフィールドは、ロックによって保護され、ロックを保持してるときだけ変更出来る。

x86.hのxchgインライン関数（xchg命令をラップしている）

```
01  static inline uint
02  xchg(volatile uint *addr, uint newval)
03  {
04    uint result;
05
06    // The + in "+m" denotes a read-modify-write operand.
07    asm volatile("lock; xchgl %0, %1" :
08                 "+m" (*addr), "=a" (result) :
09                 "1" (newval) :
10                 "cc");
11    return result;
12  }
```

spinlock.cのacquire関数

```
01  // Acquire the lock.
02  // Loops (spins) until the lock is acquired.
03  // Holding a lock for a long time may cause
04  // other CPUs to waste time spinning to acquire it.
05  void
06  acquire(struct spinlock *lk)
07  {
```

```
08    pushcli(); // disable interrupts to avoid deadlock.
09    if(holding(lk))
10      panic("acquire");
11
12    // The xchg is atomic.
13    // It also serializes, so that reads after acquire are not
14    // reordered before it.
15    while(xchg(&lk->locked, 1) != 0)
16      ;
17
18    // Record info about lock acquisition for debugging.
19    lk->cpu = cpu;
20    getcallerpcs(&lk, lk->pcs);
21  }
```

release関数は、acquireの対になるものであり、デバッグ情報用のフィールドをクリアし、そしてロックを解放する。

spinlock.cのrelease関数

```
01  // Release the lock.
02  void
03  release(struct spinlock *lk)
04  {
05    if(!holding(lk))
06      panic("release");
07
08    lk->pcs[0] = 0;
09    lk->cpu = 0;
10
11    // The xchg serializes, so that reads before release are
12    // not reordered after it.  The 1996 PentiumPro manual (Volume 3,
13    // 7.2) says reads can be carried out speculatively and in
14    // any order, which implies we need to serialize here.
15    // But the 2007 Intel 64 Architecture Memory Ordering White
16    // Paper says that Intel 64 and IA-32 will not move a load
17    // after a store. So lock->locked = 0 would work here.
18    // The xchg being asm volatile ensures gcc emits it after
19    // the above assignments (and after the critical section).
20    xchg(&lk->locked, 0);
21
22    popcli();
23  }
```

感想

詳細なロックの実装についてです。

xchg命令のところが分かりづらいですが、lk->lockedが1だった場合（すでに他でロックが獲得されている場合）、値1と交換した結果、交換直前のlk->lockedの値である1が返ってきて、while文の条件部が真となり、ループが続く（ロックが解放されるまで待つ）ことになります。
lk->lockedが0だった場合（ロックが獲得可能な場合）、値1と交換した結果、交換直前のlk->lockedの値である0が返ってきて、while文の条件部が偽となり、ループを抜け、処理を続行できることになります。

スピンロックについてはスピンロック – Wikipediaにもう少し詳しく書いてあります。

release関数内に長めのコメントがありますが、多分Wikipediaのスピンロックのページの最適化のセクションに書かれてるような最適化出来る条件を満たさないので、そのままxchgを使ってますよという事だと思います。

カテゴリー: 技術 | タグ: xv6 | 投稿日: 2012/3/11 日曜日 [http://peta.okechan.net/blog/archives/1421] |