CS 423
Operating System Design: OS support for Synchronization

Tianyin Xu

* Thanks for Prof. Adam Bates for the slides.

Implementing Synchronization

- Take 1: using memory load/store
  - See too much milk solution/Peterson's algorithm
- Take 2: *(corrected from last class!)*

```
Lock::acquire() {
    disableInterrupts();
}
```

```
Lock::release() {
    enableInterrupts();
}
```

Above solution "works" on single processor…

Let's write some simple code

Let's write a smarter implementation of acquire/release

- The key idea is to enable interrupts back ASAP

```
Lock::acquire() {
    disableInterrupts();

    enableInterrupts();
}
```

```
Lock::release() {
    disableInterrupts();
                              ;

    enableInterrupts();
}
```

Let's write some simple code

Let's write a smarter implementation of acquire/release

- The key idea is to enable interrupts back ASAP
- Use queues – ready queue and wait queue

```
Lock::acquire() {
    disableInterrupts();





    enableInterrupts();
}
```

```
Lock::release() {
    disableInterrupts();



    enableInterrupts();
}
```

Let's write some simple code

- Let's use two queues: a read queue and a wait queue
- You can use queue.add()/remove()
- Please use 7.5 minutes to write the acquire and release

```
Lock::acquire() {
    disableInterrupts();




    enableInterrupts();
}
```

```
Lock::release() {
    disableInterrupts();



    enableInterrupts();
}
```

Queueing Lock Implementation (1 Proc)

```
Lock::acquire() {
    disableInterrupts();
    if (value == BUSY) {
        waiting.add(myTCB);
        myTCB->state = WAITING;
        next = readyList.remove();
        switch(myTCB, next);
        myTCB->state = RUNNING;
    } else {
        value = BUSY;
    }
    enableInterrupts();
}
```

```
Lock::release() {
    disableInterrupts();
    if (!waiting.Empty()) {
        next = waiting.remove();
        next->state = READY;
        readyList.add(next);
    } else {
     value = FREE;
    }
    enableInterrupts();
}
```

Question

Why won't this work for multiprocessing?

Multiprocessor Sync Tool!

- Read-modify-write (RMW) instructions
  - Atomically read a value from memory, operate on it, and then write it back to memory
  - Intervening instructions prevented in hardware
- Examples
  - Test and set
  - Intel: xchgb, lock prefix
  - Compare and swap
- Any of these can be used for implementing locks and condition variables!

Test-and-set

- The **test-and-set** instruction is an instruction used to write 1 (set) to a memory location and return its old value as a single **atomic** (i.e., non-interruptible) operation. If multiple processes may access the same memory location, and if a process is currently performing a test-and-set, no other process may begin another test-and-set until the first process's test-and-set is finished.
- Please implement a lock using test-and-set (5 minutes)

```
lock:acquire() {

}

lock:release() {

}
```

Spinlocks

- A spinlock is a lock where the processor waits in a loop for the lock to become free
  - Assumes lock will be held for a short time
  - Used to protect the CPU scheduler and to implement locks

```
Spinlock::acquire() {
   while (testAndSet(&lockValue) == BUSY)
      ;
}

Spinlock::release() {
   lockValue = FREE;
   memorybarrier();
}
```

Question

Neat. So how many spinlocks do we need?

What thread is currently running?

- Thread scheduler needs to find the TCB of the currently running thread
  - To suspend and switch to a new thread
  - To check if the current thread holds a lock before acquiring or releasing it
- On a uniprocessor, easy: just use a global
- On a multiprocessor, various methods:
  - Compiler dedicates a register (e.g., r31 points to TCB running on the this CPU; each CPU has its own r31)
  - If hardware has a special per-processor register, use it
  - Fixed-size stacks: put a pointer to the TCB at the bottom of its stack
  - Find it by masking the current stack pointer

Queueing Lock Implementation (Multiproc)

Lock implementation —

```
Lock::acquire() {
    disableInterrupts();
    spinLock.acquire();
    if (value == BUSY) {
        waiting.add(myTCB);
        scheduler->
          suspend(&spinlock);
    } else {
        value = BUSY;
    }
    spinLock.release();
    enableInterrupts();
}
```

```
Lock::release() {
    TCB *next;

    disableInterrupts();
    spinLock.acquire();
    if (!waiting.Empty()) {
        next = waiting.remove();
        scheduler->makeReady(next);
    } else {
        value = FREE;
    }
    spinLock.release();
    enableInterrupts();
}
```

Queueing Lock Implementation (Multiproc)

Scheduler implementation (7.5 minutes)

```
Sched::suspend(SpinLock *lock) {

}
```

```
Sched::makeReady(TCB *thread) {

}
```

Queueing Lock Implementation (Multiproc)

Lock implementation (7.5 minutes)

```
Lock::acquire() {
    disableInterrupts();
    spinLock.acquire();
    if (value == BUSY) {
        waiting.add(myTCB);
        scheduler->
          suspend(&spinlock);
    } else {
        value = BUSY;
    }
    spinLock.release();
    enableInterrupts();
}
```

```
Lock::release() {
    TCB *next;

    disableInterrupts();
    spinLock.acquire();
    if (!waiting.Empty()) {
        next = waiting.remove();
        scheduler->makeReady(next);
    } else {
        value = FREE;
    }
    spinLock.release();
    enableInterrupts();
}
```

Queueing Lock Implementation (Multiproc)

Scheduler implementation —

```
Sched::suspend(SpinLock *lock) {
    TCB *next;

    disableInterrupts();
    schedSpinLock.acquire();
    lock->release();
    myTCB->state = WAITING;
    next = readyList.remove();
    thread_switch(myTCB, next);
    myTCB->state = RUNNING;
    schedSpinLock.release();
    enableInterrupts();
}
```

```
Sched::makeReady(TCB *thread) {

    disableInterrupts ();
    schedSpinLock.acquire();
    readyList.add(thread);
    thread->state = READY;
    schedSpinLock.release();
    enableInterrupts();
}
```

Locks for user space??

- Kernel-managed threads
  - Manage data structures in kernel space
  - System calls to communicate w/ scheduler
- User-managed threads
  - Implement functionality in thread library
  - Can't disable interrupts, but can temporarily disable upcalls to avoid preemption in library scheduler, etc.

Spinning vs Context Switch

- What's the tradeoff?

Locks in Linux

- Most locks are free most of the time. Linux implementation takes advantage of this fact!
- Fast path:
  - If lock is FREE, and no one is waiting, two instructions to acquire the lock
  - If no one is waiting, two instructions to release the lock
- Slow path
  - If lock is BUSY or someone is waiting, use multiproc impl.
- User-level locks also optimized:
  - Fast path: count is mapped to proc address space, no sys call needed when count is 0.
  - Slow path: system call to kernel, use kernel lock when waiting thread

Locks in Linux

Lock struct contains 3 (not two) states…

```
struct mutex {
 /* 1: unlocked ;
    0: locked;
     negative : locked, possible waiters */
 atomic_t count;
 spinlock_t wait_lock;
 struct list_head wait_list;
};
```

Lock acquire code is a macro (to avoid proc call)…

```
lock decl (%eax)                  // atomic decrement
                                  // %eax is pointer to count
jns 1f                            // jump if not signed
                                  // (i.e., if value is now 0)
call slowpath_acquire
1: …
```

Synchronization: Semaphores

- Semaphore has a non-negative integer value
  - P() atomically waits for value to become > 0, then decrements
  - V() atomically increments value (waking up waiter if needed)
- Semaphores are like integers except:
  - Only operations are P and V
  - Operations are atomic
    - If value is 1, two P's will result in value 0 and one waiter

Compare Implementations

Lock implementation —

```
Lock::acquire() {
    disableInterrupts();
    spinLock.acquire();
    if (value == BUSY) {
        waiting.add(myTCB);
        suspend(&spinlock);
    } else {
        value = BUSY;
    }
    spinLock.release();
   enableInterrupts();
}
```

```
Lock::release() {
    disableInterrupts();
    spinLock.acquire();
    if (!waiting.Empty()) {
        next = waiting.remove();
        scheduler->makeReady(next);
    } else {
     value = FREE;
    }
    spinLock.release();
    enableInterrupts();
}
```

Compare Implementations

Semaphore implementation —

```
Semaphore::P() {
    disableInterrupts();
    spinLock.acquire();
    if (value == 0) {
        waiting.add(myTCB);
        suspend(&spinlock);
    } else {
        value--;
    }
    spinLock.release();
   enableInterrupts();
}
```

```
Semaphore::V() {
    disableInterrupts();
    spinLock.acquire();
    if (!waiting.Empty()) {
        next = waiting.remove();
        scheduler->makeReady(next);
    } else {
     value++;
    }
    spinLock.release();
    enableInterrupts();
}
```

Semaphores Harmful?

- Semaphores conflate the roles of locks and condition variables (mutual exclusion, shared data).
  - Simpler code verification w/o: *prove every lock is eventually unlocked.*
- Semaphores have state!
  - What does value=3 mean? Programmer must carefully map object state to semaphore value.
  - CVs, in contrast, allows us to wait on arbitrary state/predicate, and are thus a better abstraction.
- However, semaphores have good uses, including…
  - Unlocked waits, e.g., interrupt handler that synchronizes communication between I/O device and waiting threads.

Semaphore Bounded Queue

```
get() {
    fullSlots.P();
    mutex.P();
    item = buf[front % MAX];
    front++;
    mutex.V();
    emptySlots.V();
    return item;
}
```

```
put(item) {
    emptySlots.P();
    mutex.P();
    buf[last % MAX] = item;
    last++;
    mutex.V();
    fullSlots.V();
}
```

Initially: front = last = 0; MAX is buffer capacity
mutex = 1; emptySlots = MAX; fullSlots = 0;

Implementing CVs w/ Semaphores

How can we implement Condition Variables using semaphores

Take 1:

```
wait(lock) {
    lock.release();
    semaphore.P();
    lock.acquire();
}
signal() {
    semaphore.V();
}
```

Problems?

Implementing CVs w/ Semaphores

How can we implement Condition Variables using semaphores

Take 2:

```
wait(lock) {
    lock.release();
    semaphore.P();
    lock.acquire();
}
signal() {
    if (semaphore is not empty)
        semaphore.V();
}
```

Problems?

Implementing CVs w/ Semaphores

How can we implement Condition Variables using semaphores

Take 3:

```
wait(lock) {
    semaphore = new Semaphore;
    queue.Append(semaphore);   // queue of waiting threads
    lock.release();
    semaphore.P();
    lock.acquire();
}
signal() {
    if (!queue.Empty()) {
        semaphore = queue.Remove();
        semaphore.V();   // wake up waiter
    }
}
```

Problems?

Implementing CVs w/ Semaphores

Implementation used for Microsoft Windows before native support was offered:

Take 4:

```
//Put thread on queue of waiting threads….
void CV::wait(Lock *lock){
   semaphore = new Semaphore(0);
   waitQueue.Append(semaphore)
   lock.release();
   semaphore.P();
   lock.acquire();
}
```

```
//Wake up one waiter if any.
void CV::signal() {
 if(!waitQueue.isEmpty()) {
    semaphore = queue.Remove();
    semaphore.V();
   }
}
```