Thread and Synchronization

Synchronization Mechanisms
(Module 21)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014
Kernel Semaphores

- **struct semaphore**: count, wait queue, and number of sleepers
  ```c
  void sem_init(struct semaphore *sem, int val);
  // Initialize a semaphore’s counter sem->count to given value
  inline void down(struct semaphore *sem);
  // try to lock the critical section by decreasing sem->count
  inline void up(struct semaphore *sem); // release the semaphore
  ```

- **blocked thread** can be in **TASK_UNINTERRUPTIBLE** or **TASK_INTERRUPTIBLE** (by timer or signal)

- **Special case** – mutexes (binary semaphores)
  ```c
  void init_MUTEX (struct semaphore *sem)
  void init_MUTEX_LOCKED(struct semaphore *sem)
  ```

- **Read/Write semaphores**
Spin lock vs Semaphore

- Only a spin lock can be used in interrupt context,
- Only a semaphore can be held while a task sleeps.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Recommended Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low overhead locking</td>
<td>Spin lock</td>
</tr>
<tr>
<td>Short lock hold time</td>
<td>Spin lock</td>
</tr>
<tr>
<td>Long lock hold time</td>
<td>Semaphore</td>
</tr>
<tr>
<td>Need to lock from interrupt context</td>
<td>Spin lock</td>
</tr>
<tr>
<td>Need to sleep while holding lock</td>
<td>Semaphore</td>
</tr>
</tbody>
</table>

Other mechanisms:
- Completion: synchronization among multiprocessors
- The global kernel lock (a.k.a big kernel lock, or BKL)
  - `Lock_kernel()`, `unlock_kernel()`
- RCU – read-copy update, for mostly-read access
Blocking Mechanism in Linux Kernel

- **ISR can wake up a block kernel thread**
  - which is waiting for the arrival of an event

- **Wait queue**

- **Wait_for_completion_timeout**
  - specify “completion” condition, timeout period, and action at timeout
  - “complete” to wake up thread in wait queue
  - wake-one or wake-many

```c
struct semaphore {
    raw_spinlock_t lock;
    unsigned int count;
    struct list_head wait_list;
};
```

```c
struct completion {
    unsigned int done;
    wait_queue_head_t wait;
};
```

```c
struct __wait_queue_head {
    spinlock_t lock;
    struct list_head task_list;
};
```
Wait_for_Completion Example

- In `i2c-designware-core.c`
- Threads call `i2c_dw_xfer` will do
  - `INIT_COMPLETION(dev->cmd_complete);`
  - `i2c_dw_xfer_init(dev);`
  - `ret = wait_for_completion_interruptible_timeout(&dev->cmd_complete, HZ);`
- In `i2c_dw_xfer_init`, interrupt get enabled
- In `i2c_dw_isr`, when xfer is done
  - `complete(&dev->cmd_complete);`
Mutex in Linux

- Two states: locked and unlocked.
  - if locked, wait until it is unlocked
  - only the thread that locked the mutex may unlock it

- Various implementations for performance/function tradeoffs
  - Speed or correctness (deadlock detection)
  - lock the same mutex multiple times
  - priority-based and priority inversion
  - forget to unlock or terminate unexpectedly

- Available types
  - normal
  - fast
  - error checking
  - recursive: owner can lock multiple times (counting)
  - robust: return an error code when crashes while holding a lock
  - RT: priority inheritance
Pthread Futex

- Lightweight and scalable
- In the noncontended case can be acquired/released from userspace without having to enter the kernel.
  - lock is a user-space address, e.g. a 32-bit lock variable field.
  - “uncontended” and “waiter-pending”
  - kernel provides futex queue, and sys_futex system call
  - invoke sys_futex only when there is a need to use futex queue
  - need atomic operations in user space
  - race condition: atomic update of ulock and system call are not atomic

```c
typedef struct ulock_t {
    long status;
} ulock_t;
```
Reader/Writer -- ISR and Buffering

- **Input:** single producer (ISR) and single consumer (thread)

- **If a read is initiated by the thread**
  - calls "read" with a buffer of $n$ bytes
  - initiate IO operation, enable interrupt
  - ISR reads input and store in the buffer.
  - If done, signal the completion

- **Blocking or nonblocking**
  - in thread context (e.g. vxWorks) – semaphore, lock
  - in kernel context (Linux) – wait queue

- **Guarded access**
  - Lock (mutex) and interrupt lock (disable)