Kernel Programming

What to Expect?

- How to do programming in “Kernel C” for
  - Achieving Concurrency
  - Keeping Time
  - Providing Delays
  - Timer Control

Concurrency

Concurrency with Locking

- Mutexes
  - Header: <linux/mutex.h>
  - Type: struct mutex
  - APIs
    - DEFINE_MUTEX
    - mutex_is_locked
    - mutex_lock, mutex_trylock, mutex_unlock
- Semaphores
  - Header: <linux/semaphore.h>
  - Type: struct semaphore
  - APIs
    - sema_init
    - down, down_trylock, down_interruptible, up

Concurrency w/ Locking (cont.)

- Spin Locks
  - Header <linux/spinlock.h>
  - Type: spinlock_t
  - APIs
    - spin_lock_init
    - spin_[try]lock, spin_unlock
- Reader-Writer Locks
  - Header: <linux/spinlock.h>
  - Type: rwlock_t
  - APIs
    - read_lock, read_unlock
    - write_lock, write_unlock

Concurrency without Locking

- Atomic Variables
  - Header: <asm-generic/atomic.h>
  - Type: atomic_t
  - Macros
    - ATOMIC_INIT
    - atomic_read, atomic_set
    - atomic_add, atomic_sub, atomic_inc, atomic_dec
    - atomic_xchg

Concurrency w/o Locking (cont.)

- Atomic Bit Operations
  - Header: <linux/bitops.h>
  - APIs
    - rol8, rol16, rol32, ror8, ror16, ror32
    - find_first_bit, find_first_zero_bit
    - find_last_bit
    - find_next_bit, find_next_zero_bit
  - Header: <asm-generic/bitops.h>
  - APIs
    - set_bit, clear_bit, change_bit
    - test_and_set_bit, test_and_clear_bit, test_and_change_bit

Wait Queues

- Wait Queues
  - Header: <linux/wait.h>
  - Wait Queue Head APIs
    - DECLARE_WAIT_QUEUE_HEAD(wq);
    - wait_event_interruptible(wq, cond);
    - wait_event_interruptible_timeout(wq, cond, timeout);
    - wake_up_interruptible(&wq);
    - ... (non-interruptible set)
  - Wait Queue APIs
    - DECLARE_WAITQUEUE(w, current);
    - add_wait_queue(&wq, &w);
    - remove_wait_queue(&wq, &w);

Time Keeping

Time since Bootup

- tick – Kernel's unit of time. Also called jiffy
- HZ – ticks per second
  - Defined in Header: <linux/param.h>
  - Typically, 1000 for desktops, 100 for embedded systems
- 1 tick = 1ms (desktop), 10ms (embedded systems)
- Variables: jiffies & jiffies_64
  - Header: <linux/jiffies.h>
  - APIs
    - time_after, time_before, time_in_range, ...
    - get_jiffies_64, ...
    - msec_to_jiffies, timespec_to_jiffies, timeval_to_jiffies, ...
    - jiffies_to_msec, jiffies_to_timespec, jiffies_to_timeval, ...

Time since Bootup (cont.)

- Platform specific “Time Stamp Counter”
  - On x86
    - Header: <asm/msr.h>
    - API: rdtsc(ul low_tsc_ticks, ul high_tsc_ticks);
  - Getting it generically
    - Header: <linux/timex.h>
    - API: read_current_timer(unsigned long *timer_val);

Absolute Time

- Header: <linux/time.h>
- APIs
  - mktime(y, m, d, h, m, s) – Seconds since Epoch
  - void do_gettimeofday(struct timeval *tv);
  - struct timespec current_kernel_time(void);

Delays

Long Delays

- Busy wait: cpu_relax

```
while (time_before(jiffies, j1))
        cpu_relax();
```

- Yielding: schedule/schedule_timeout

```
while (time_before(jiffies, j1))
        schedule();
```

Short Delays but Busy Waiting

- Header: <linux/delay.h>
- Arch. specific Header: <asm/delay.h>
- APIs
  - void ndelay(unsigned long ndelays);
  - void udelay(unsigned long udelays);
  - void mdelay(unsigned long mdelays);

Long Delays: Back to Yielding

- Header: <linux/delay.h>
- APIs
  - void msleep(unsigned int millisecs);
  - unsigned long msleep_interruptible(unsigned int millisecs);
  - void ssleep(unsigned int secs);

Timers

Kernel Timers

- Back end of the various delays
- Header: <linux/timer.h>
- Type: struct timer_list
- APIs
  - void init_timer(struct timer_list *); /* Nullifies */
  - struct timer_list TIMER_INITIALIZER(f, t, p);
  - void add_timer(struct timer_list *);
  - void del_timer(struct timer_list *);
  - int mod_timer(struct timer_list *, unsigned long);
  - int del_timer_sync(struct timer_list *);

Tasklets

- Timers without specific Timing
- Header: <linux/interrupt.h>
- Type: struct tasklet_struct
- APIs
  - void tasklet_init(struct tasklet_struct *t, void (*func) (unsigned long), unsigned long data);
  - void tasklet_kill(struct tasklet_struct *t);
  - DECLARE_TASKLET(name, func, data);
  - tasklet_enable(t), tasklet_disable(t)
  - tasklet_[hi_]schedule(t);

Work Queues

- In context of “Special Kernel Thread”
- Header: <linux/workqueue.h>
- Types: struct workqueue_struct, struct work_struct
- Work Queue APIs
  - q = create_workqueue(name);
  - q = create_singlethread_workqueue(name);
  - flush_workqueue(q);
  - destroy_workqueue(q);
- Work APIs
  - DECLARE_WORK(w, void (*function)(void *), void *data);
  - INIT_WORK(w, void (*function)(void *), void *data);
- Combined APIs
  - int queue_work(q, &w);
  - int queue_delayed_work(q, &w, d);
  - int cancel_delayed_work(&w);
- Global Shared Work Queue API
  - schedule_work(&w);

Helper Interfaces

Other Helper Interfaces in Latest Kernels

- User Mode Helper
- Linked Lists
- Hash Lists
- Notifier Chains
- Completion Interface
- Kthread Helpers

What to Expect?

- How to do programming in “Kernel C” for
  - Achieving Concurrency
    - With & without Locking
    - Wait Queues
  - Keeping Time
    - Relative & Absolute
  - Providing Delays
    - Long and Short
    - Busy Wait and Yielding
  - Timer Control
    - Kernel Timers
    - Tasklets
    - Work Queues

Any Queries?