Linux Device Drivers

- Deferred Works

सी डैक
CDAC

Agenda

- Concept of jiffies
- Kernel Timers
- Tasklets
- Work Queues
- Shared Queue

Concept of jiffies

Concept of Jiffies

- Timer interrupts are generated by the system's timing hardware at regular intervals
- The interval, as per the Linux kernel is given by '*HZ*' ticks per second. For example, if the value of HZ is 250(say), that means that the timer interrupts 250 times per second.
- Scheduling and other timely functionalities depends on this value of HZ.
- Each time a timer interrupt occurs, the value of an internal 64-bit(even on 32-bit platforms) counter, named *jiffies_64* is incremented.
- Driver authors can normally access *jiffies* which is an *unsigned long* of the least 32-bits of jiffies_64.

Applying jiffies : Profiling

- unsigned long t1, t2, diff, msecs;

  ...

  t1 = jiffies;

  /* Code you wish to profile */

  t2 = jiffies;

  diff = t2 - t1;

  msecs = diff * 1000 / HZ;

Applying jiffies : Generating delays

- The following macros are used to work with jiffies
- <linux/jiffies.h>

```
int time_after(unsigned long a, unsigned long b);
int time_before(unsigned long a, unsigned long b);
int time_after_eq(unsigned long a, unsigned long b);
int time_before_eq(unsigned long a, unsigned long b);
```

Applying jiffies : Generating delays

- General delay calculation :

  unsigned long d = jiffies + MILLI_DELAY * HZ /1000;

- while (time_before(jiffies, d))

  schedule();

Short delays

- Busy waiting
  - void ndelay(unsigned long nsecs);
  - void udelay(unsigned long usecs);
  - void mdelay(unsigned long msecs);
- Delayed sleeping
  - void msleep(unsigned long msecs);
  - void ssleep(unsigned long secs);

Kernel Timers

Kernel Timers

- A kernel timer is a data structure that instructs the kernel to execute a user-defined function with a user-defined argument at a user defined time.
- Kernel timers are run as the result of a “software interrupt” i.e. they be executed atomically, without any sleeps involved.
- The timer API :
  - <linux/timer.h>

```
struct timer_list {
    /* ... */
    unsigned long expires;
    void (*function)(unsigned long);
};
```

Kernel Timers : Usage

- /* Initialization */

  void timer_setup(struct timer_list *timer,void (*fun)(struct timer_list*),flags); // refer header-file for flags.

- /* Timer Operations */

  void add_timer(struct timer_list *timer);

  int mod_timer(struct timer_list *timer, unsigned long expires);

- /* Deletion */

  int del_timer_sync(struct timer_list *timer);

Tasklets

Tasklets

- Tasklets resemble kernel timers in some ways. Unlike kernel timers, however, you can't ask to execute the function at a specific time.
- By scheduling a tasklet, you simply ask for it to be executed at a later time chosen by the kernel.
- This behavior is especially useful with interrupt handlers, where the hardware interrupt must be managed as quickly as possible, but most of the *data management can be safely delayed to a later time.*
- Tasklet API :
  - <linux/interrupt.h>

```
struct tasklet_struct {
    /* ... */
    void (*function)(unsigned long);
    unsigned long data;
};
```

Tasklets : Usage

- /* Initialization */

  tasklet_init(struct tasklet_struct *t, void (*func)(unsigned long),
  unsigned long data);

- /* Operations on tasklets */

  tasklet_schedule(struct tasklet *t);

  tasklet_disable(struct tasklet *t);

  tasklet_enable(struct tasklet *t);

- /* Destroying the tasklet */

  tasklet_kill(struct tasklet *t);

Work Queues

Work Queues

- *Workqueues* are, superficially, similar to tasklets, but with a difference that workqueues run in process context.
- Usage :

  <linux/workqueue.h>

  struct workqueue_struct;

  struct work_struct;

Work Queues : Usage

- /* Work queue creation */

  struct workqueue_struct *create_singlethreaded_workqueue(char *name);

- /* Create a work * /

  INIT_WORK(struct work_struct *work, void (*func)(struct work *));

- /* Assign the work to the work queue */

  int queue_work(struct workqueue_struct *queue,
  struct work_struct *work);

- /* Operations on work queues */

  void flush_workqueue(struct workqueue *queue);

  void destroy_workqueue(struct workqueue *queue);

Shared Queue

- It is similar to that of a work queue except that the queue is maintained
  by the kernel itself.
- Usage :
  - <linux/workqueue.h>
    struct work_struct;
  - /* Initialise the work */
    INIT_WORK(struct work_struct *work, void (*func)(struct work *));
  - /* Schedule the work to execute */
    int schedule_work(struct work_struct *work);

References

- Jonathan Corbet, Alessandro Rubini and Greg Kroah-Hartman,"Linux Device Drivers",3rd Edition, O'Reilly Publications
- Robert Love, "Linux Kernel Development", 3rd Edition, Developer's Library

Thank You :)