# EMBEDDED DEVICE DRIVERS

Linux Device Drivers on Beaglebone Black

# LKM: Interrupts and work

- Interrupt
  - Urgent request for service
  - Needs to be handled “fast”

- ISRs / IRQ handlers
  - Need to be short
  - Execute few lines of code
  - Finish quickly

- But what if some devices
  - Interrupt
  - And ask for more work to be done?
    - Examples:
      - High bandwidth network cards with lots of network data to process?
      - High resolution video/image cameras with lots of frames to ingest?

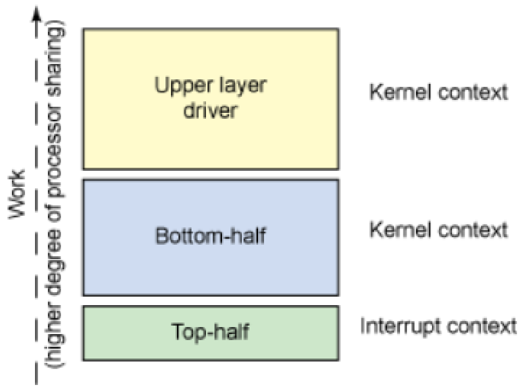
# LKM: Interrupt context

- Interrupt context challenges
  - Some / all interrupts may need to be disabled
    - Increases latencies for other hardware events
  - Hence work done needs to be minimal
    - More work **pushed to kernel context**
      - When CPU can be more gainfully shared
      - And interrupts are enabled

- This bifurcation leads to
  - Better performance
    - Due to ability to deal quickly
    - With high-frequency interrupt events
      - By deferring non time-sensitive work

# LKM: Interrupt bifurcation

- Interrupt context
  - Top half
    - Quick, fast and efficient
    - Pending work “deferred” to bottom half
- Kernel context
  - Bottom half
    - Runs at a “later” time
    - Can take up heavy processing / slow work
  - Driver / module code



# LKM: Interrupt top half

- Technically speaking
  - Top half is the “actual” IRQ handler

- Typically, a top half
  - Ensures interrupt generated by right hardware
    - Specially for shared interrupts
  - Clears pending bit if any
  - Performs immediate work
  - Schedules work for later

- Minimal amount of CPU resource sharing

# LKM: Interrupt bottom half

- Work deferral options
  - Running in atomic context
    - SoftIRQ
      - **Tasklet**
  - Running in process context
    - **Workqueue**

# LKM: SoftIRQ

- SoftIRQ (Software Interrupt)
  - Can pre-empt all other tasks on system
    - Except hardware IRQ handlers
  - One softirq can not preempt another softirq (at same level)
    - But the same softirq can run on different CPUs concurrently

- Historically the first deferral mechanism (v2.3)
  - Statically allocated (only 10!)
    - No dynamic redefinition possible

- Meant for
  - High-frequency threaded job scheduling
  - Rarely used – most drivers prefer **tasklets**

- Run in atomic context
  - Cannot sleep

# LKM: SoftIRQ details (1/2)

- Determined statically at **compile-time**
  - Internal representation

    ```
    struct softirq_action {
        void (*action)(struct softirq_action *);
    };
    ```

  - Take up 1 entry each in the softirq array

    ```
    static struct softirq_action softirq_vec[NR_SOFTIRQS];
    ```

  - Initialized early in boot

    ```
    void open_soft_irq(int nr, void (*action)(struct softirq_action *))
    {
            softirq_vec[nr].action = action;
    }
    ```

- Launched by per-CPU kernel thread
  - Handles unserviced software interrupts
    - ***ksoftirqd/n***

# LKM: SoftIRQ details (2/2)

- Linux has 10 softirq's defined

```
enum
{
    HI_SOFTIRQ=0,              /* High-priority tasklets */
    TIMER_SOFTIRQ,             /* Timers */
    NET_TX_SOFTIRQ,            /* Send network packets */
    NET_RX_SOFTIRQ,            /* Receive network packets */
    BLOCK_SOFTIRQ,             /* Block devices */
    BLOCK_IOPOLL_SOFTIRQ,      /* Block devices with I/O polling */
    TASKLET_SOFTIRQ,           /* Normal Priority tasklets */
    SCHED_SOFTIRQ,             /* Scheduler */
    HRTIMER_SOFTIRQ,           /* High-resolution timers */
    RCU_SOFTIRQ,               /* RCU locking */
    NR_SOFTIRQS                /* Number of softirqs type */
};
```

# LKM: Tasklets

- Tasklets
  - Softirqs that can be allocated and initialized at runtime
  - Built on 2 of the 10 softirq vectors in *softirq_vec[]*
    *HI_SOFTIRQ, TASKLET_SOFTIRQ*

- Scheduled through softirq mechanism
  - On the same CPU core that schedules
    - **Non-preemptive** principle
      - Serially, run to finish, no sleep or locks (only spinlocks)

- Structure

```
struct tasklet_struct {
    struct tasklet_struct *next;
    unsigned long state;
    atomic_t count;
    void (*func)(unsigned long);
    unsigned long data;
    };
```

# LKM: Tasklet API (1/2)

- Creation
  - Static

    ***DECLARE_TASKLET****(name, func);*
    ***DECLARE_TASKLET_DISABLED****(name, func);*

  - Dynamic

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

- Enable/disable

  *void* ***tasklet_enable****(struct tasklet_struct *t);*

  *void* ***tasklet_disable****(struct tasklet_struct *t);*
  *void* ***tasklet_disable_nosync****(struct tasklet_struct *t); // immediate*

# LKM: Tasklet API (2/2)

- Schedule

  *void **tasklet_schedule**(struct tasklet_struct *t);*

  *void **tasklet_hi_schedule**(struct tasklet_struct *t);*

- Kill

  *void **tasklet_kill**(struct tasklet_struct *t);*

  *void **tasklet_kill_immediate**(struct tasklet_struct *t, unsigned int cpu);*

# LKM: Tasklet exercise

- Refer ***mod12*** directory
  - File ***mod12-1.c*** contains code for a ***tasklet***
    - We create a GPIO for a button press
      - And assign it an IRQ handler as earlier
    - We also create and initialize a tasklet dynamically
    - Inside the handler, we schedule a tasklet
      - That prints data passed from interrupt to tasklet
      - And increments ***numPressed***

  - Compile and load the module on BBB
    - Observe ***dmesg*** output when button is pressed
    - Press the button rapidly and study the prints….
      - *Note that tasklet is at lower prio than IRQ*!

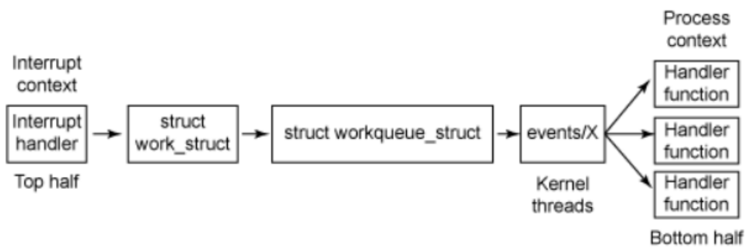
# LKM: Tasklet – notice on LXR

```
/* Tasklets --- multithreaded analogue of BHs.

   This API is deprecated. Please consider using threaded IRQs instead:
   https://lore.kernel.org/lkml/20200716081538.2sivhkj4hcyrusem@linutronix.de

   Main feature differing them of generic softirqs: tasklet
   is running only on one CPU simultaneously.

   Main feature differing them of BHs: different tasklets
   may be run simultaneously on different CPUs.

   Properties:
   * If tasklet_schedule() is called, then tasklet is guaranteed
     to be executed on some cpu at least once after this.
   * If the tasklet is already scheduled, but its execution is still not
     started, it will be executed only once.
   * If this tasklet is already running on another CPU (or schedule is called
     from tasklet itself), it is rescheduled for later.
   * Tasklet is strictly serialized wrt itself, but not
     wrt another tasklets. If client needs some intertask synchronization,
     he makes it with spinlocks.
 */
```

# LKM: Workqueue

- A linked list of tasks that need to be run

- Runs in **pre-emptible** process context
  - Schedulable, can sleep
  - Can run on a different CPU than the one launching it




- Kernel-global work queue
  - ***system_wq***
    - Backed by per-CPU kernel thread
      - ***events/X***

# LKM: Workqueue entities

- Work to be deferred (work item)
  ***struct work_struct*** ***(my_work)***
    - Schedule as soon as possible
  ***struct delayed_work*** ***(my_dwork)***
    - Schedule after a minimum specified delay

- Workqueue itself
  ***struct workqueue_struct*** ***(kernel's / my_work_queue)***
    - A queue of work items

- Worker threads
  - Dedicate threads that run work item functions

- Worker pools
  - Collection of worker threads used for thread management

# LKM: WQ API (1/2)

- API for work function (work item)

  *void (***my_work_func**)(struct work_struct *work);*

- Creation:
  - Static:

    ***DECLARE_WORK****(my_work, my_work_func);*

    ***DECLARE_DELAYED_WORK****(my_dwork, my_work_func);*
  - Dynamic:

    ***INIT_WORK****(my_work, my_work_func);*

    ***INIT_DELAYED_WORK****(my_dwork, my_work_func);*

- Check:

  ***work_pending****(my_work);*

  ***delayed work_pending****(my_work)*

# LKM: WQ API (2/2)

- Schedule

  *int **schedule_work**(struct work_struct *work);*
  *int **schedule_delayed_work**(struct delayed_work *dwork, unsigned long delay);*
  *int **schedule_work_on**(int cpu, struct work_struct *work);*
  *int **schedule_delayed_work_on**(int cpu, struct delayed_work *dwork, unsigned long delay);*

- Cancel – if not already executed

  *int **cancel_work_sync**(struct work_struct *work);*
  *int **cancel_delayed_work_sync**(struct delayed_work *dwork);*

- Delete work from workqueue

  *int **flush_work**(struct work_struct *work);*
  *void **flush_scheduled_work**(void); // entire global workqueue*

# LKM: WQ exercise #1

- Refer ***mod12*** directory
  - ***mod12-2.c*** contains code for statically allocated work
  - ***mod12-3.c*** contains code for dynamically allocated work
  - ***mod12-4.c*** contains code for dynamically allocated delayed work

  - Compile and load the modules on BBB
    - Observe behavior on ***dmesg*** output for button press

# LKM: WQ: Workqueues

- So far we used
  - The kernel-global ***system_wq*** workqueue
  - This works for most use-cases
    - API's are
      - ***schedule_work() / schedule_delayed_work()***

- But what if we wanted our own workqueue?
  - Then we need to create our own workqueue
  - Queue work items on to our own workqueue
  - Destroy the workqueue so created on exit

# LKM: WQ: Own workqueue (1/2)

- Variable:

  *static struct* ***workqueue_struct*** **my_work_queue;*

- Creation:

  *struct workqueue_struct **__*create_workqueue*__*(char *name);*

- Deletion:

  *void* ***destroy_workqueue****(struct workqueue_struct *my_work_queue);*

- Flush:

  *void* ***flush_workqueue****(struct workqueue_struct *wq);*

# LKM: WQ: Own workqueue (2/2)

- Schedule:

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

  *int* ***queue_delayed_work****(struct workqueue_struct *wq, struct delayed_work *dwork, unsigned long delay);*

  *int* ***queue_work_on****(int cpu, struct workqueue_struct *wq, struct work_struct *work);*

  *int* ***queue_delayed_work_on****(int cpu, struct workqueue_struct *wq, struct delayed_work *dwork, unsigned long delay);*

# LKM: WQ exercise #2

- Refer ***mod12*** directory
  - ***mod12-5.c*** contains code for dynamically allocated work on own workqueue
    - Notice we allocate our own workqueue
    - Queue work on this workqueue
    - Destroy it on exit
  - Compile and load the module on BBB
    - Observe ***dmesg*** output for button press

# THANK YOU!