7. Timer

NXKR_YoungSikYang

Exported on 03/11/2024

Table of Contents

1. Jiffies

The kernel uses jiffies to measure time intervals when low resolution is tolerable.

A jiffy is the duration between two consecutive ticks of the system timer.

1.1 Uses

- **Timekeeping**: They are used for time-related functions within the kernel. For example, The scheduler uses jiffies to decide when to switch between processes, allowing the kernel to allocate CPU time effectively among running processes.
- **Timer Management**: Timers and time-based delays (like `msleep()`) are implemented using jiffies, allowing the kernel and kernel modules to perform actions after a specified number of jiffies has passed.

1.2 Limitation

- **Resolution**: For tasks requiring higher time resolution than jiffies can provide, the kernel uses other mechanisms, such as high-resolution timers (hrtimers), which can offer nanosecond precision.

2. Hrtimer

2.1 Advantages over traditional timers

- **Precision**: The main advantage of hrtimers is their ability to provide precise timing, crucial for applications where timing accuracy is paramount. Hrtimers can provide timing resolutions in the nanosecond range, far exceeding the granularity offered by jiffies.
- **Dynamic**: They adjust dynamically to the resolution supported by the underlying hardware, making the most of available timer hardware capabilities.
- **Kernel Integration**: Hrtimers are fully integrated with the kernel's timekeeping infrastructure, allowing them to be used for a wide range of high-precision timing tasks.

2.2 Uses

Hrtimers are suitable for tasks that require high precision.

- **Networking**: High-resolution timers are crucial in networking for tasks such as managing packet transmission intervals, timeout handling, and implementing precise time-sensitive protocols.
- **Multimedia**: For multimedia applications, precise timing is essential for audio and video synchronization, playback control, and real-time streaming.
- **Real-Time Applications**: Real-time systems that require deterministic behavior and precise timing, such as industrial control systems, robotics, and real-time simulations, rely heavily on hrtimers.

3. getnstimeofday

`getnstimeofday()` retrieves the current time with nanosecond precision. It fills a data structure with the current time, broken down into seconds and nanoseconds.

3.1 Overview

- **Purpose**: The primary goal of `getnstimeofday()` is to obtain the current time with high precision. It is used in scenarios where detailed time information is necessary, surpassing the granularity that seconds or milliseconds can offer.
- **Return Type**: It populates a `timespec` (or `timespec64` in newer kernels) structure, which consists of two fields:
    - `tv_sec` : Represents the number of seconds elapsed since the Unix epoch (00:00:00 UTC on 1 January 1970).
    - `tv_nsec` : Represents the number of nanoseconds past the second. The value is in the range of 0 to 999,999,999 nanoseconds.

4. Practice

4.1 Jiffies

4.1.1 Source code

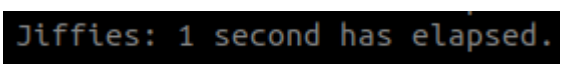
- **Access to Jiffies:** Jiffies are stored as a global variable (unsigned long int) that is incremented by the system timer interrupt at each timer tick. The current value of jiffies can be accessed directly to measure elapsed time or schedule future actions.
- **HZ:** The frequency at which jiffies are incremented is defined by the `HZ` value in the kernel. This value represents the number of timer ticks per second and varies across different hardware platforms and kernel configurations. Higher `HZ` values offer finer-grained time resolution but can increase overhead due to more frequent timer interrupts.
    - e.g. 250 HZ = trigger per 1/250 second = resolution of 4 ms

This function prints a message when 1 second has elapsed.

```
#include <linux/module.h>  // Needed by all modules
#include <linux/kernel.h>  // Needed for KERN_INFO
#include <linux/init.h>    // Needed for the macros
#include <linux/jiffies.h> // Needed for jiffies
#include <linux/gpio.h>
#include <linux/interrupt.h>

#define button 3 // Example GPIO number

static unsigned int irq_number;

// Interrupt handler function(top-half)
static irqreturn_t btn_irq_handler(int irq, void *dev_id)
{
    return IRQ_WAKE_THREAD;
}

static irqreturn_t bottom_half(int irq, void *dev_id)
{
    unsigned long jiffies_at_start = jiffies; // Capture the current value of jiffies
    unsigned long delay = 1 * HZ; // Delay of 1 second; HZ is the number of jiffies
per second
    // Busy wait until the specified delay has elapsed
    while (time_before(jiffies, jiffies_at_start + delay))
    {
        // This loop will keep running until the delay has passed
```

```
        // In a real application, you generally don't want to use busy waiting!
        // This is just for demonstration.
    }
    printk(KERN_INFO "Jiffies: 1 second has elapsed.\n");
    return IRQ_HANDLED;
}

static int set_irq(int gpio, irq_handler_t bottom_half, unsigned long flag)
{
    int result_irq;
    // Get the IRQ number for our GPIO
    irq_number = gpio_to_irq(gpio);
    if (irq_number < 0) {
        printk(KERN_INFO "GPIO to IRQ mapping failed\n");
        return 1;
    }
    // Request the IRQ line
    result_irq = request_threaded_irq(irq_number, btn_irq_handler, bottom_half, flag,
"btn_irq", NULL);
    if (result_irq) {
        printk(KERN_INFO "IRQ request failed\n");
        return 1;
    }
    return 0;
}

static int __init testdev_init(void)
{
    set_irq(button, bottom_half, IRQF_TRIGGER_FALLING);
    return 0;
}

// Module exit
static void __exit testdev_exit(void)
{
    free_irq(irq_number, NULL);
}

module_init(testdev_init);
module_exit(testdev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("Practice module");
```

4.1.2 Result



4.2 Getnstimeofday

getnstimeofday() has been deprecated and replaced with **timesepc64** and **ktime_get_real_ts64()** to solve the year 2038 problem.

4.2.1 Source code

Same code as above except **bottom_half()**

```
static irqreturn_t bottom_half(int irq, void *dev_id)
{
    struct timespec now;
    // Get current time
    getnstimeofday(&now);
    // Print current time
    printk(KERN_INFO "Current time: %ld seconds and %ld nanoseconds\n", now.tv_sec,
now.tv_nsec);
    return IRQ_HANDLED;
}
```

4.2.2 Result

```
Current time: 1600602783 seconds and 493494300 nanoseconds
```

4.3 Hrtimer

4.3.1 Source code

timer_callback() is called when the hrtimer has fired

```
#include <linux/module.h>  // Needed by all modules
#include <linux/kernel.h>  // Needed for KERN_INFO
#include <linux/init.h>    // Needed for the macros
#include <linux/gpio.h>
#include <linux/interrupt.h>
#include <linux/timekeeping.h>

#define button 3 // Example GPIO number

static unsigned int irq_number;
static struct hrtimer my_timer;
static ktime_t timer_period, start_time;
```

```
// Interrupt handler function(top-half)
static irqreturn_t btn_irq_handler(int irq, void *dev_id)
{
    return IRQ_WAKE_THREAD;
}

enum hrtimer_restart timer_callback(struct hrtimer *timer)
{
    ktime_t elapsed;
    elapsed = ktime_sub(ktime_get(), start_time); // Calculate elapsed time
    printk(KERN_INFO "Timer fired. Requested duration: %lld ns, Actual duration: %lld
ns\n",
           ktime_to_ns(timer_period), ktime_to_ns(elapsed));
    return HRTIMER_NORESTART;
}

static irqreturn_t bottom_half(int irq, void *dev_id)
{
    // 1ms = 1,000,000ns
    timer_period = ktime_set(0, 100 * 1000000); // 100ms in ns
    // Initialize the high-resolution timer
    hrtimer_init(&my_timer, CLOCK_MONOTONIC, HRTIMER_MODE_REL);
    my_timer.function = timer_callback;
    start_time = ktime_get(); // Capture start time
    hrtimer_start(&my_timer, timer_period, HRTIMER_MODE_REL);
    return IRQ_HANDLED;
}

static int set_irq(int gpio, irq_handler_t bottom_half, unsigned long flag)
{
    int result_irq;
    // Get the IRQ number for our GPIO
    irq_number = gpio_to_irq(gpio);
    if (irq_number < 0) {
        printk(KERN_INFO "GPIO to IRQ mapping failed\n");
        return 1;
    }
    // Request the IRQ line
    result_irq = request_threaded_irq(irq_number, btn_irq_handler, bottom_half, flag,
"btn_irq", NULL);
    if (result_irq) {
        printk(KERN_INFO "IRQ request failed\n");
        return 1;
    }
    return 0;
}

static int __init testdev_init(void)
{
    set_irq(button, bottom_half, IRQF_TRIGGER_FALLING);
    return 0;
}
```

```
// Module exit
static void __exit testdev_exit(void)
{
    int ret;
    free_irq(irq_number, NULL);
    ret = hrtimer_cancel(&my_timer);
    if (ret) {
        printk(KERN_INFO "The timer was still in use.\n");
    }
}

module_init(testdev_init);
module_exit(testdev_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Your Name");
MODULE_DESCRIPTION("Practice module");
```

4.3.2 Result

hrtimer is functional but is expected to support nano second precision, which the target board does not support.

```
Timer fired. Requested duration: 100000000 ns, Actual duration: 100104700 ns
```

Reference

- https://embetronicx.com/tutorials/linux/device-drivers/using-kernel-timer-in-linux-device-driver/
- https://dataonair.or.kr/db-tech-reference/d-lounge/technical-data/?mod=document&uid=236817