International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

The Linux Hardware Monitoring Subsystem: Architecture, Implementation, and Programming Interface

Anish Kumar

Email: *yesanishhere[at]gmail.com*

Abstract: ***The Linux Hardware Monitoring (HWMON) subsystem provides a standardized framework for monitoring hardware sensors in Linux systems. This paper presents a comprehensive analysis of the HWMON subsystem's architecture, implementation details, and programming interface. We examine the core components, sensor types, data structures, and interfaces that enable hardware monitoring across diverse devices. The paper includes implementation examples, discusses the sysfs interface, and explores the relationship between device drivers, event handlers, and user-space applications in the Linux kernel.***

Keywords: Linux kernel, HWMON, Sensors

1. Introduction

Hardware monitoring is crucial for system health and performance management in modern computing systems. The Linux HWMON subsystem, developed by Guenter Roeck, provides a unified interface for accessing sensor data from various hardware components, including temperature sensors, voltage monitors, fan speed controllers, and power meters. This standardized approach simplifies driver development and ensures consistent access to hardware monitoring data across different platforms and architectures.

2. Historical Development

The HWMON subsystem has evolved significantly since its initial development:

- Initial integration in kernel 2.3 development series
- Limited support in kernel 2.4 (basic sensors)
- Full integration in kernel 2.5/2.6
- Extended support for modern hardware monitoring features

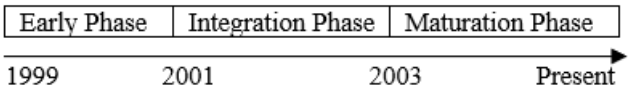

Figure 1: HWMON Development Timeline

3. Architecture Overview

The HWMON subsystem consists of three main layers:

a) Driver Layer

- Interfaces with specific hardware sensors
- Translates device-specific signals
- Handles device initialization and resource management

b) Core Layer

- Manages device registration
- Provides standardized APIs
- Handles data processing

c) Interface Layer

- Exposes sysfs interface
- Provides user-space access
- Manages attribute handling

4. Implementation Details

a) Device Registration

The HWMON subsystem provides two main registration functions:

```
struct device *hwmon_device_register_with_info(
    struct device *dev,
    const char *name,
    void *drvdata,
    const struct hwmon_chip_info *info,
    const struct attribute_group **extra_groups);

struct device *devm_hwmon_device_register_with_info(
    struct device *dev,
    const char *name,
    void *drvdata,
    const struct hwmon_chip_info *info,
    const struct attribute_group **extra_groups);
```

b) Data Structures

The core data structures include:

```
struct hwmon_chip_info {
    const struct hwmon_ops *ops;
    const struct hwmon_channel_info * const *info;
};

struct hwmon_ops {
    umode_t (*is_visible)(const void *,
                          enum hwmon_sensor_types type,
                          u32 attr, int);
    int (*read)(struct device *,
                enum hwmon_sensor_types type,
                u32 attr, int, long *);
    int (*write)(struct device *,
                 enum hwmon_sensor_types type,
                 u32 attr, int, long);
};
```

5. Sensor Types and Attributes

a) Supported Sensor Types

The HWMON subsystem supports various sensor types:

Type	Code	Description
hwmon_chip	Virtual	General chip attributes
hwmon_temp	Temperature	Temperature sensors
hwmon_in	Voltage	Voltage sensors
hwmon_curr	Current	Current sensors
hwmon_power	Power	Power sensors
hwmon_energy	Energy	Energy sensors
hwmon_humidity	Humidity	Humidity sensors
hwmon_fan	Fan	Fan speed sensors

6. Driver Implementation

a) Driver Private Data Structure

```
struct example_hwmon_data {
    struct device *hwmon_dev;
    long temp1_input; /* Temperature value */
    long fan1_input;  /* Fan speed value */
};
```

b) Core Driver Functions

The core functions for reading sensor data and determining attribute visibility are implemented as follows:

```
/* Read function for the HWMON subsystem */
static int example_read(struct device *dev, enum hwmon_sensor_types type,
                        u32 attr, int channel, long *val) {
    struct example_hwmon_data *data = dev_get_drvdata(dev);

    switch (type) {
    case hwmon_temp:
        *val = data->temp1_input;
        return 0;
    case hwmon_fan:
        *val = data->fan1_input;
        return 0;
    default:
        return -EOPNOTSUPP;
    }
}

/* Visibility function for HWMON attributes */
static umode_t example_is_visible(const void *data,
                                  enum hwmon_sensor_types type,
                                  u32 attr, int channel) {
    switch (type) {
    case hwmon_temp:
    case hwmon_fan:
        return 0444;  /* Read-only attributes */
    default:
        return 0;
    }
}
```

c) HWMON Operations and Chip Info

The HWMON operations and chip info structures are defined as:

```
/* HWMON device operations */
static const struct hwmon_ops example_hwmon_ops = {
    .is_visible = example_is_visible,
    .read = example_read,
};

/* HWMON chip info */
static const struct hwmon_chip_info example_chip_info = {
    .ops = &example_hwmon_ops,
    .info = example_info,
};
```

d) Driver Probe Function

The probe function for initializing the driver is implemented as:

```
/* Probe function */
static int example_probe(struct platform_device *pdev) {
    struct example_hwmon_data *data;

    data = devm_kzalloc(&pdev->dev, sizeof(*data), GFP_KERNEL);
    if (!data)
        return -ENOMEM;

    data->temp1_input = 25000;  /* 25.000 degrees Celsius */
    data->fan1_input = 1200;    /* 1200 RPM */

    data->hwmon_dev = devm_hwmon_device_register_with_info(
        &pdev->dev, "example", data, &example_chip_info, NULL);
    return PTR_ERR_OR_ZERO(data->hwmon_dev);
}
```

7. Testing and Verification

a) Tools and Utilities

The HWMON subsystem can be tested using:

- Sensors-detect: Automatic sensor detection
- Sensors: Display current readings
- Pwmconfig: Fan control configuration
- Fancontrol: Automated fan control daemon

b) Verification Steps

- Run sensors-detect to identify hardware
- Check sensor readings with sensors command
- Verify sysfs interface functionality
- Test fan control operations
- Validate temperature thresholds

8. Advanced Features

Power Management Integration

The HWMON subsystem integrates with power management:

- Thermal zone registration
- Cooling device support
- Power capping interface
- Energy monitoring

9. Conclusion

The Linux HWMON subsystem provides a robust and

flexible framework for hardware monitoring. Its standardized interface simplifies driver development while ensuring consistent access to sensor data across different platforms. The subsystem's extensible design ensures support for new hardware monitoring features and devices as they emerge.

References

[1] Linux Kernel Documentation, "Linux hardware monitoring," [Online]. Available: https://www.kernel.org/doc/Documentation/hwmon/sysfs-interface
[2] Linux Kernel Source, "include/linux/hwmon.h," [Online]. Available: https://github.com/torvalds/linux/blob/master/include/linux/hwmon.h
[3] Linux Kernel Source, "drivers/hwmon/hwmon.c," [Online]. Available: https://github.com/torvalds/linux/blob/master/drivers/hwmon/hwmon.c