Linux Systems Internals and Programming (contd…)

4. Kernel Space Operations

4.1. Direct Hardware Access

Kernel space is a privileged execution environment where the operating system has full control over the hardware. This space is separate from user space, ensuring security and stability by restricting direct hardware access from user applications.

4.1.1. Kernel Space's Direct Access to Hardware

- The kernel can directly interact with hardware components like the CPU, RAM, and I/O devices.
- It achieves this through special CPU instructions and memory-mapped I/O (MMIO).
- Unlike user-space applications, which must use system calls to request hardware access, the kernel can read and write hardware registers directly.
- Examples include managing disk I/O, network packets, and scheduling CPU processes.

4.1.2. Execution of Device Drivers in Kernel Space

- Device drivers are software components that allow the OS to communicate with hardware devices (e.g., USB, GPU, network card).
- Most drivers execute in kernel space to ensure high performance and low-latency interactions with hardware.
- A kernel module is a piece of code that can be loaded into the kernel dynamically, allowing new hardware support without recompiling the entire OS.

Example of a simple Linux kernel module:

```
#include <linux/module.h>
#include <linux/kernel.h>

static int __init my_driver_init(void) {
    printk(KERN_INFO "Driver loaded: Kernel space access initialized.\n");
    return 0;
}

static void __exit my_driver_exit(void) {
    printk(KERN_INFO "Driver unloaded: Kernel space access removed.\n");
}

module_init(my_driver_init);
module_exit(my_driver_exit);
```

```
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Kernel Module");
MODULE_AUTHOR("Your Name");
```

To compile and insert this module:

```
make
sudo insmod my_driver.ko
sudo rmmod my_driver
dmesg | tail
```

4.2. Core Functions in Kernel Space

4.2.1. Kernel Space as the Core of the Operating System

- The kernel is responsible for managing all hardware resources.
- It provides essential services such as process management, memory allocation, and inter-process communication (IPC).
- The kernel also handles file system operations and networking.

4.2.2. Resource Management by the Kernel

- **Memory Management:** The kernel allocates and deallocates memory for processes and ensures isolation between them.
- **Process Scheduling:** The kernel schedules processes to run on the CPU efficiently using scheduling algorithms like Completely Fair Scheduler (CFS).
- **File System Management:** The kernel manages file operations through virtual file systems (VFS).
- **Networking:** The kernel implements network protocols like TCP/IP and handles packet routing.

---

5. User Space Restrictions

5.1. Security and Stability

5.1.1. Ensuring Security through User Space Restrictions

- User applications are restricted from directly accessing hardware or modifying critical system memory.
- This prevents malicious software from compromising the system.
- The kernel enforces access control using permissions, capabilities, and sandboxing.

5.1.2. Maintaining System Stability

- If a user-space process crashes, it does not affect the entire system, unlike kernel crashes.
- System calls provide a controlled way for applications to request resources.

Example: If a user program tries to access `/dev/mem`, it will be blocked unless explicitly permitted.

5.2. Controlled Interfaces (Syscalls)

5.2.1. Limitations on Direct Hardware Access

- User-space applications cannot access hardware directly; they must use system calls.
- The kernel ensures that only authorized requests reach the hardware.

5.2.2. Role of Syscalls as Controlled Interfaces

- A system call acts as a gateway between user space and kernel space.
- Examples include `open()`, `read()`, `write()`, and `ioctl()`.

Example: Using a system call in C

```
#include <stdio.h>
#include <unistd.h>

int main() {
    write(1, "Hello from user space!\n", 23);
    return 0;
}
```

Here, `write()` is a system call that requests the kernel to write data to standard output.

5.3. Benefits of Isolation

5.3.1. Advantages of Isolating User Space from Kernel Space

- **Security:** Prevents unauthorized access to system resources.
- **Stability:** Ensures that user applications cannot crash the entire system.
- **Efficiency:** The kernel optimizes resource allocation without interference from applications.