Communication between Userspace and Kernel Space

There are several ways to communicate between the userspace and kernel space in Linux:

- **IOCTL**
- **Procfs**
- **Sysfs**
- **Configfs**
- **Debugfs**
- **Sysctl**
- **UDP Sockets**
- **Netlink Sockets**

In this tutorial, we will focus on **Sysfs**.

SysFS in Linux Kernel

Sysfs is a virtual filesystem exported by the kernel, similar to **/proc**. The files in Sysfs contain information about devices and drivers. Some files in Sysfs are even writable, allowing configuration and control of devices attached to the system. Sysfs is always mounted on **/sys**.

Differences Between Various Kernel Filesystems

Filesystem	Purpose
Sysfs	Used to export system information from the kernel space to the user space for specific devices.
Procfs	Used to export process-specific information.
Debugfs	Used to export debug information by the developer.

---

kobject (Kernel Object)

The heart of the Sysfs model is the **kernel object (kobject)**. A `kobject` acts as a glue between Sysfs and the kernel. It is represented by `struct kobject`, which is defined in `<linux/kobject.h>`.

Structure of kobject

```
struct kobject {
    char *k_name;
    char name[KOBJ_NAME_LEN];
    struct kref kref;
    struct list_head entry;
    struct kobject *parent;
    struct kset *kset;
    struct kobj_type *ktype;
    struct dentry *dentry;
```

```
};
```

Fields in kobject

- **`k_name`**: Pointer to dynamically allocated name, if needed.
- **`name`**: Name of the kobject, which appears in Sysfs.
- **`kref`**: Reference counter to ensure the object is freed only when its reference count reaches zero.
- **`kset`**: A collection of related kobjects.
- **`entry`**: Used for linking the kobject into a list.
- **`parent`**: Parent kobject, forming a hierarchy.
- **`ktype`**: Defines operations and attributes associated with the kobject.

Steps to Create and Use Sysfs

1. Creating a Directory in Sysfs

To create a directory in **/sys**, we use the `kobject_create_and_add` function:

```
struct kobject *kobject_create_and_add(const char *name, struct kobject
*parent);
```

Arguments

- **`name`** – Name of the kobject.
- **`parent`** – Parent kobject (can be `kernel_kobj` to create under `/sys/kernel/`).

Behavior

Parent kobject	Directory Path Created
`kernel_kobj`	`/sys/kernel/<name>`
`firmware_kobj`	`/sys/firmware/<name>`
`fs_kobj`	`/sys/fs/<name>`
`NULL`	`/sys/<name>`

Cleanup

To remove the created directory, use:

```
kobject_put(my_kobject);
```

---

2. Creating a File in Sysfs

To interact between userspace and kernel space, we create a **sysfs file** using **sysfs attributes**.

Kobj_attribute Structure

```
struct kobj_attribute {
    struct attribute attr;
```

```
    ssize_t (*show)(struct kobject *kobj, struct kobj_attribute *attr, char
*buf);
    ssize_t (*store)(struct kobject *kobj, struct kobj_attribute *attr, const
char *buf, size_t count);
};
```

Fields in kobj_attribute

- **`attr`** – The attribute representing the file.
- **`show`** – Function pointer for reading the file.
- **`store`** – Function pointer for writing to the file.

Defining an Attribute Using `__ATTR` Macro

```
__ATTR(name, permission, show_ptr, store_ptr);
```

Show and Store Functions

```
ssize_t show_file(struct kobject *kobj, struct kobj_attribute *attr, char *buf)
{
    return sprintf(buf, "Hello from Kernel!\n");
}

ssize_t store_file(struct kobject *kobj, struct kobj_attribute *attr, const char
*buf, size_t count) {
    printk(KERN_INFO "Sysfs Write: %s", buf);
    return count;
}
```

Creating a Sysfs File

```
static struct kobj_attribute my_attr = __ATTR(my_file, 0664, show_file,
store_file);
```

Adding the Attribute to Sysfs

```
sysfs_create_file(kobj, &my_attr.attr);
```

Removing the Sysfs File

```
sysfs_remove_file(kobj, &my_attr.attr);
```

Creating a Group of Attributes

If multiple attributes are needed, use `sysfs_create_group`:

```
static struct attribute *attrs[] = {
    &my_attr.attr,
    NULL // Terminate the list
};

static struct attribute_group my_attr_group = {
    .name = "my_group",
    .attrs = attrs,
};

sysfs_create_group(kobj, &my_attr_group);
```

Removing the Attribute Group

```
sysfs_remove_group(kobj, &my_attr_group);
```

Summary

1. **Create a kobject**

   ```
   my_kobject = kobject_create_and_add("my_sysfs", kernel_kobj);
   ```

2. **Create a Sysfs File**

   ```
   static struct kobj_attribute my_attr = __ATTR(my_file, 0664, show_file,
   store_file);
   sysfs_create_file(my_kobject, &my_attr.attr);
   ```

3. **Create a Sysfs Group (Optional)**

   ```
   sysfs_create_group(my_kobject, &my_attr_group);
   ```

4. **Remove Sysfs Entries on Module Unload**

   ```
   sysfs_remove_group(my_kobject, &my_attr_group);
   kobject_put(my_kobject);
   ```

This tutorial provides an overview of how to create and manage **Sysfs** files in the Linux kernel for communication between userspace and kernel space.