Assignment Requirements for Character Driver Enhancements

Use IOCTL reference code given in Day2 from Linux Device Drivers

Function: my_read()

1. Partial Read and Write

Requirement:

- Implement **partial read and write functionality** to allow reading and writing data in **smaller chunks** rather than the entire buffer at once.
- Ensure that **read/write pointers** are correctly managed to track how much data has been read or written.

Details:

- Use a **read pointer (read_ptr)** to keep track of the position in the buffer from where the next read will start.
- Use a **write pointer (write_ptr)** to keep track of where new data is written.
- Handle **wrap-around cases** when the end of the buffer is reached (circular buffer behavior).

Example Scenario:

- If the buffer size is 1000 bytes and the user reads 500 bytes, the next read should start from the 501st byte.
- If the buffer reaches its end, the read should **wrap around to the beginning** of the buffer.

2. Reverse Read Enable/Disable (via IOCTL)

Requirement:

- Implement a **feature to enable or disable reverse reading** using an **IOCTL command**.
- When **reverse read is enabled**, the my_read() function should return data starting from the **end of the buffer** instead of the beginning.

Details:

- Add a **global flag (reverse_read_enabled)** that can be set or cleared via an **IOCTL command**.
- When **enabled**, the my_read() function should read the data **in reverse order** (from end to start).

```
#define IOCTL_ENABLE_REVERSE_READ _IO('a', 4)

#define IOCTL_DISABLE_REVERSE_READ _IO('a', 5)
```

Example Scenario:

- If the buffer contains the string **"HelloWorld"**, a normal read would return **"HelloWorld"**, but a reverse read would return **"dlroWolleH"**.

3. Return System Information on Read (Enable via IOCTL)

Requirement:

- Add an **IOCTL command** to toggle the **return of system information** (like **CPU usage**, **free memory**, etc.) when the my_read() function is called.

Details:

- When the **system info read mode is enabled**, the my_read() function should **ignore the buffer content** and instead return **system information**.
- Use **Linux kernel APIs** like ***si_meminfo()*** to fetch memory usage and other details.

```
#define IOCTL_ENABLE_SYSINFO_READ _IO('a', 6)

#define IOCTL_DISABLE_SYSINFO_READ _IO('a', 7)
```

Example Scenario:

- If the system info mode is enabled, calling my_read() might return:

```
CPU Usage: 25%

Free Memory: 512 MB
```

Function: my_write()

1. Data Size Limit Handling

Requirement:

- Implement a feature to **limit the maximum size of data** that can be written to the buffer in a single write operation.

Details:

- Set a **maximum data size limit** (e.g., 512 bytes) for each write operation.
- If the user tries to write more than the allowed size, **reject the operation** and return an **appropriate error message**.

Example Scenario:

- If the maximum write size is set to **512 bytes**, and the user tries to write **600 bytes**, the driver should return an error like **"Data size limit exceeded"**.

2. Log Write Operations (with Timestamps)

Requirement:

- Maintain a **log of all write operations**, including:
    - **Number of bytes written**.
    - **Timestamp of the write operation**.

Details:

- Use a **log buffer** to store the write operation details.

Example Scenario:

- After a write operation, the log might look like:

```
[12:30:45] 100 bytes written

[12:31:10] 200 bytes written
```

Function: my_ioctl()

1. Dynamic Buffer Allocation

Requirement:

- Add an **IOCTL command** to **dynamically allocate or resize the buffer** used by the driver.

Details:

- Allow the user to **increase or decrease** the size of the buffer at runtime.
- Ensure that **existing data is preserved** when resizing the buffer.

```
#define IOCTL_SET_BUFFER_SIZE_IOW('a', 8, int)
```

Example Scenario:

- The user can call an IOCTL command to **increase the buffer size from 1000 bytes to 2000 bytes**.

2. Device Statistics

Requirement:

- Add an **IOCTL command** to return **device statistics**, including:
    - **Number of read operations**.
    - **Number of write operations**.
    - **Total bytes read**.
    - **Total bytes written**.

Details:

- Maintain a **statistics structure** in the driver to track these values.

```
#define IOCTL_GET_STATS _IOR('a', 8, int)
```

Example Scenario:

- Calling the IOCTL command might return:

```
Reads: 10

Writes: 5

Bytes Read: 1000

Bytes Written: 500
```

4. Reset Device (Factory Reset)

Requirement:

- Add an **IOCTL command** to **reset the device to factory settings**, which includes:
  - **Clearing all buffers**.
  - **Resetting all statistics**.

Details:

- Ensure that the **device is fully reset** to its initial state.

```
#define IOCTL_RESET_DEVICE _IOW('a', 8, int)
```

Example Scenario:

- After a factory reset, the buffer should be empty, and all statistics should be set to **0**.

Requirements to Update test_app.c for Communicating with the Enhanced Driver

To test the enhanced functionalities of the **character driver**, the **user-space application (test_app.c)** needs to be modified to handle **IOCTL commands** and test various features like **partial reads/writes**, **reverse reads**, **system info retrieval**, **encryption**, and **log retrieval**.

Here are the **requirements for modifying test_app.c** to support these new features.

General Requirements for test_app.c

1. **Provide a Menu-Driven Interface**
   - Add a **menu** in test_app.c that lists all the features available in the driver.
   - The user should be able to select options to:
     - Enable/disable reverse read.
     - Enable/disable system info read mode.
     - Retrieve device statistics.

2. **Add IOCTL Command Support**
    - Implement the necessary **ioctl() calls** to communicate with the driver for:
        - Enabling/disabling **reverse read**.
        - Enabling/disabling **system info mode**.
        - Retrieving **device statistics**.
        - Set **buffer size**
        - Reset **buffer**

Specific Changes to test_app.c

Menu Option for Partial Read/Write

- Implement **partial read and write functionality** in the application.
- Allow the user to specify **how many bytes to read/write**.

Menu Example:

```
Menu:
1. Open Device
2. Write to Device (Partial Write)
3. Read from Device (Partial Read)
4. Enable Reverse Read
5. Disable Reverse Read
6. Enable System Info Read Mode
7. Disable System Info Read Mode
8. Get Device Statistics
9. Reset Device
10. Close Device
11. Exit
```