Real-Time Operating Systems

S7784976 - Mamoru Ota

1 Introduction

This report describes the design and implementation of a simple kernel driver and a user-space application with multiple threads based on the given assignment requirements.

2 Assignment Requirements

The assignment required the following:

- Design an application with 3 periodic threads (J1, J2, J3) and 1 aperiodic thread (J4).
- Implement periodic threads with periods of 300ms, 500ms, and 800ms.
- The aperiodic thread (J4) is triggered by J2.
- Threads "waste time" by executing dummy loops.
- Design a simple driver that supports open, close, and write system calls.
- Each thread writes its identifier with square brackets into the driver.
- The driver logs the received messages into the kernel log.

3 Kernel Driver Design

The kernel driver provides minimal functionality with three operations: open, close, and write. Messages written by threads are logged into the kernel log.

3.1 Driver Operations

- **Open:** Initializes the driver when accessed.
- **Close:** Releases resources when the driver is closed.
- **Write:** Receives input from threads and logs it.

3.2 Algorithm for Driver Behavior

The following algorithm describes the behavior of the kernel driver.

Algorithm 1 Kernel Driver Operations

Allocate device number.
Initialize and register the character device.
Create a device file at /dev/mydriver.
Open: Print log message indicating file access.
Write: Copy input string, log to kernel.
Close: Print log message indicating file release.
Unregister device during exit.

4 User-Space Application Design

The application defines four threads, each executing specific tasks and interacting with the kernel driver.

4.1 Threads

- Periodic Threads
  - J1: Period = 300ms
  - J2: Period = 500ms
  - J3: Period = 800ms
- Aperiodic Thread
  - J4: Triggered by J2 at specific intervals.

4.2 Thread Execution Algorithm

The following algorithm describes the execution of thread.

Algorithm 2 Thread Behavior

Open driver file.
Write identifier with open brackets, e.g., ”[1”.
Close driver file.
Perform dummy workload.
Open driver file.
Write identifier with close brackets, e.g., ”1]”.
Close driver file.
Sleep until the next period (for periodic threads).

5 Execution Steps

To execute the kernel driver and user-space application, follow these steps:

5.1 Kernel Driver

1. Compile the kernel driver:

   ```
   make
   ```

2. Insert the compiled kernel module:

   ```
   sudo insmod mydriver.ko
   ```

3. Verify the creation of the device file:

   ```
   ls /dev/mydriver
   ```

4. Check kernel logs for initialization messages:

   ```
   sudo dmesg
   ```

5.2 User-Space Application

1. Compile the application:

   ```
   gcc -pthread main.c -o main
   ```

2. Ensure proper permissions for accessing `/dev/mydriver`:

   ```
   sudo chmod 666 /dev/mydriver
   ```

3. Run the application:

   ```
   ./main
   ```

4. Monitor the kernel logs to observe task execution and preemption in another terminal:

   ```
   sudo dmesg
   ```

5.3 Cleanup

1. Terminate the application if needed:

```
Ctrl+C
```

2. Remove the kernel module:

```
sudo rmmod mydriver
```

3. Clean up build files:

```
make clean
```

6 Results

This section presents the results obtained from executing the user-space application with the kernel driver under different configurations of the `waste_time()` function. Specifically, the loop count multiplier was adjusted to analyze the impact of computation time on thread preemption and scheduling.

6.1 Observations

Two experiments were conducted to analyze thread behavior under different computational loads:

6.1.1 Experiment 1: High Computational Load

In this experiment, the computation time in the `waste_time()` function was set to:

```
loops * 100000
```

Output:

```
[11][2[12]1][33][11][22][4[11]4]
```

Analysis:

- Preemption is clearly observed.
- The aperiodic thread J4 is triggered by J2 and executed.

6.1.2 Experiment 2: Low Computational Load

In this experiment, the computation time in the `waste_time()` function was reduced to:

```
loops * 1000
```

Output:

```
[11][33][11][22][44][11][33][11]
```

Analysis:

- Preemption behavior is no longer visible. Tasks execute sequentially without interruptions.
- This result highlights that lower computation times reduce contention for CPU resources, leading to cooperative-like scheduling.

6.2 Discussion

The results demonstrate that the behavior of real-time threads in the system depends heavily on computational load. With higher computational times, preemptions become apparent, allowing higher-priority tasks to interrupt lower-priority ones. Conversely, lower computational times reduce preemption, resulting in tasks running in sequence. These findings validate the importance of tuning computational load when designing real-time systems to test preemptive behavior effectively.

7 Conclusion

This project demonstrates the integration of kernel drivers and user-space applications using multithreading and preemption. The kernel logs effectively capture the sequence of thread operations, validating correct behavior under scheduling constraints.