Submitted by:
Harshvardhan Mehta(1MS22CS062)
Kartik Hegde(1MS22CS076)

Linux Kernel Module code (To update and write the hardware time to shared memory)

```c
// rtc_shm_sync.c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>
#include <linux/slab.h>
#include <linux/time.h>
#include <linux/uaccess.h>
#include <linux/mm.h>
#include <linux/gfp.h>
#include <linux/version.h>
#include <asm/io.h>          // for page_to_pfn()

#define DEVICE_NAME "rtc_shm"
#define CLASS_NAME  "rtc_class"
#define SHM_SIZE    PAGE_SIZE

struct rtc_shared_time {
   u64 seconds;
   u32 nanoseconds;
};

static struct page *shared_page;
static void        *shared_time_page;
static dev_t        dev_number;
static struct cdev  rtc_cdev;
static struct class *rtc_class;
static struct device *rtc_device;

/* open/release no-ops */
static int rtc_open(struct inode *inode, struct file *file) { return 0; }
static int rtc_release(struct inode *inode, struct file *file) { return 0; }

/* mmap: map the single shared page into user space */
static int rtc_mmap(struct file *filp, struct vm_area_struct *vma)
{
   unsigned long pfn = page_to_pfn(shared_page);
   size_t size = vma->vm_end - vma->vm_start;

   if (size > SHM_SIZE)
       return -EINVAL;

   if (remap_pfn_range(vma,
                       vma->vm_start,
                       pfn,
                       size,
                       vma->vm_page_prot)) {
       pr_err("rtc_shm: remap_pfn_range failed\n");
       return -EAGAIN;
   }

   return 0;
}

/* master writes current time into shared memory */
static ssize_t rtc_write(struct file *file, const char __user *buf,
                        size_t count, loff_t *ppos)
{
   struct timespec64 now;
   struct rtc_shared_time *rtc_data = shared_time_page;

   ktime_get_real_ts64(&now);
   rtc_data->seconds     = now.tv_sec;
   rtc_data->nanoseconds = now.tv_nsec;

   pr_info("rtc_shm: [WRITE] Master wrote time: %llu.%09u\n",
           rtc_data->seconds, rtc_data->nanoseconds);

   return count;
```

```
}

/* slave reads time, sets system clock, and returns a small message */
static ssize_t rtc_read(struct file *file, char __user *buf,
                       size_t count, loff_t *ppos)
{
   struct rtc_shared_time *rtc_data = shared_time_page;
   char output[80];
   int len;

   /* only allow one read */
   if (*ppos > 0)
       return 0;

   /* sync system clock */
   {
       struct timespec64 ts = {
           .tv_sec  = rtc_data->seconds,
           .tv_nsec = rtc_data->nanoseconds
       };
       do_settimeofday64(&ts);
   }

   pr_info("rtc_shm: [READ] Synced system time to: %llu.%09u\n",
           rtc_data->seconds, rtc_data->nanoseconds);

   /* send confirmation to user-space */
   len = snprintf(output, sizeof(output),
                  "Synced to %llu.%09u\n",
                  rtc_data->seconds, rtc_data->nanoseconds);

   if (count < len)
       return -EINVAL;
   if (copy_to_user(buf, output, len))
       return -EFAULT;

   *ppos += len;
   return len;
}

static const struct file_operations rtc_fops = {
   .owner    = THIS_MODULE,
```

```c
    .open    = rtc_open,
    .release = rtc_release,
    .write   = rtc_write,
    .read    = rtc_read,
    .mmap    = rtc_mmap,
};

static int __init rtc_shm_init(void)
{
    int ret;

    /* allocate one contiguous page */
    shared_page = alloc_page(GFP_KERNEL);
    if (!shared_page)
        return -ENOMEM;
    shared_time_page = page_address(shared_page);

    /* register char device */
    ret = alloc_chrdev_region(&dev_number, 0, 1, DEVICE_NAME);
    if (ret)
        goto free_page;

    cdev_init(&rtc_cdev, &rtc_fops);
    ret = cdev_add(&rtc_cdev, dev_number, 1);
    if (ret)
        goto unregister_chrdev;

#if LINUX_VERSION_CODE >= KERNEL_VERSION(6, 4, 0)
    rtc_class = class_create(DEVICE_NAME);
#else
    rtc_class = class_create(THIS_MODULE, CLASS_NAME);
#endif
    if (IS_ERR(rtc_class)) {
        ret = PTR_ERR(rtc_class);
        goto del_cdev;
    }

    rtc_device = device_create(rtc_class, NULL, dev_number, NULL, DEVICE_NAME);
    if (IS_ERR(rtc_device)) {
        ret = PTR_ERR(rtc_device);
        goto destroy_class;
    }
    pr_info("rtc_shm: Module loaded, /dev/%s ready\n", DEVICE_NAME);
    return 0;

destroy_class:
    class_destroy(rtc_class);
del_cdev:
    cdev_del(&rtc_cdev);
unregister_chrdev:
    unregister_chrdev_region(dev_number, 1);
free_page:
    __free_page(shared_page);
    return ret;
}

static void __exit rtc_shm_exit(void)
{
    device_destroy(rtc_class, dev_number);
    class_destroy(rtc_class);
    cdev_del(&rtc_cdev);
    unregister_chrdev_region(dev_number, 1);
    __free_page(shared_page);

    pr_info("rtc_shm: Module unloaded\n");
}

module_init(rtc_shm_init);
module_exit(rtc_shm_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Kartik Hegde");
MODULE_DESCRIPTION("RTC synchronization over shared memory across partitions");
```

Program to read from the shared memory

```c
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <stdint.h>

struct rtc_shared_time {
   uint64_t seconds;
   uint32_t nanoseconds;
};

int main() {
   int fd = open("/dev/rtc_shm", O_RDWR);
   if (fd < 0) {
       perror("open");
       return 1;
   }

   struct rtc_shared_time *rtc_time = mmap(NULL, getpagesize(),
                                                PROT_READ | PROT_WRITE,
                                                MAP_SHARED, fd, 0);
   if (rtc_time == MAP_FAILED) {
       perror("mmap");
       close(fd);
       return 1;
   }

   printf("Shared Memory RTC Time:\n");
```

```c
    printf("  Seconds     : %llu\n", rtc_time->seconds);
    printf("  Nanoseconds : %u\n", rtc_time->nanoseconds);

    // Optional: manually set system time (for testing)
    /*
    struct timespec ts = {
        .tv_sec = rtc_time->seconds,
        .tv_nsec = rtc_time->nanoseconds
    };
    clock_settime(CLOCK_REALTIME, &ts);
    */

    munmap(rtc_time, getpagesize());
    close(fd);
    return 0;
}
```