Assignment-10

Linux System and its Applications

Systems and Storage Laboratory

Department of Computer Science and Engineering

Chung-Ang University

Assignment-10: Synchronization atomic instruction

Atomic operations (atomic instructions)

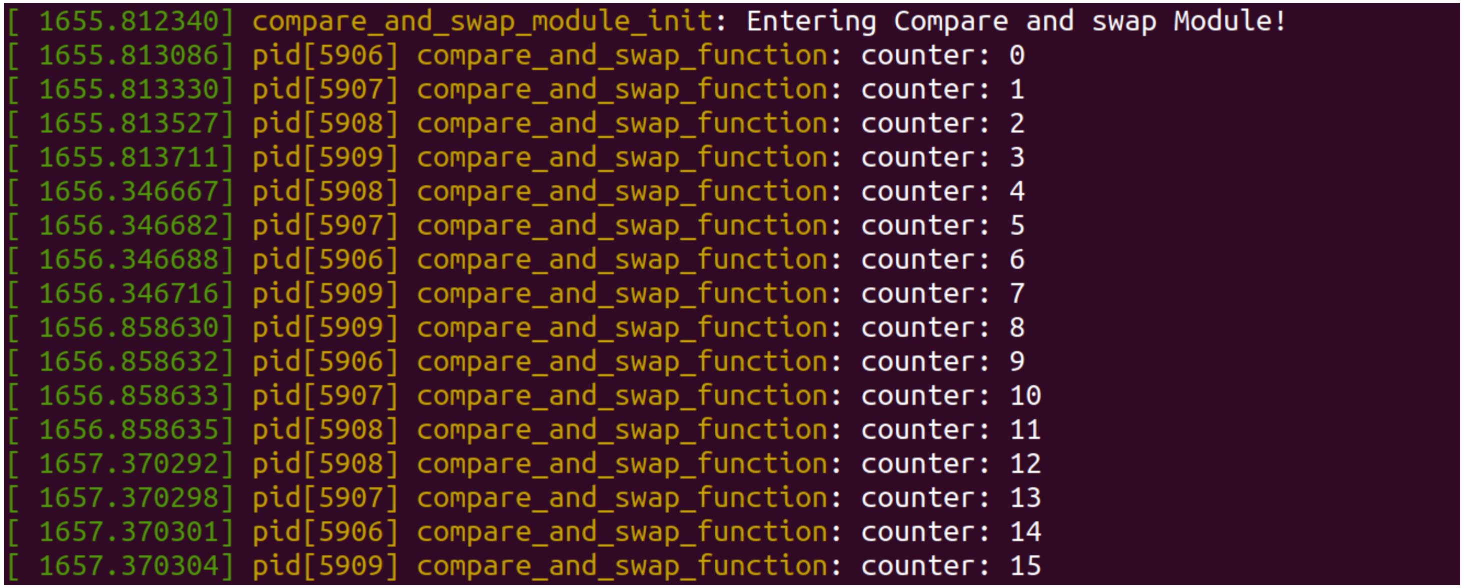
- Perform atomic operations using atomic instructions in a simple kernel module with four kernel threads.
  - Fetch-and-add
  - Test-and-set
  - Compare-and-swap
- Each thread should increase the shared resource "`counter`" by 1 and print it.

- Reference:
  - Lecture slide **6. synchronization (2)**
  - https://www.ibm.com/docs/en/xcfbg/121.141?topic=functions-gcc-atomic-memory-access-built-in

Example screenshot 1 atomic instruction

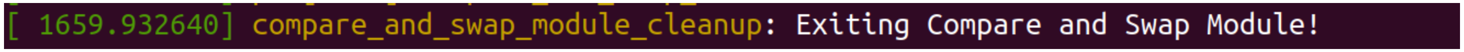
Atomic operations

- Compare-and-Swap example after inserting module



```
[ 1655.812340] compare_and_swap_module_init: Entering Compare and swap Module!
[ 1655.813086] pid[5906] compare_and_swap_function: counter: 0
[ 1655.813330] pid[5907] compare_and_swap_function: counter: 1
[ 1655.813527] pid[5908] compare_and_swap_function: counter: 2
[ 1655.813711] pid[5909] compare_and_swap_function: counter: 3
[ 1656.346667] pid[5908] compare_and_swap_function: counter: 4
[ 1656.346682] pid[5907] compare_and_swap_function: counter: 5
[ 1656.346688] pid[5906] compare_and_swap_function: counter: 6
[ 1656.346716] pid[5909] compare_and_swap_function: counter: 7
[ 1656.858630] pid[5909] compare_and_swap_function: counter: 8
[ 1656.858632] pid[5906] compare_and_swap_function: counter: 9
[ 1656.858633] pid[5907] compare_and_swap_function: counter: 10
[ 1656.858635] pid[5908] compare_and_swap_function: counter: 11
[ 1657.370292] pid[5908] compare_and_swap_function: counter: 12
[ 1657.370298] pid[5907] compare_and_swap_function: counter: 13
[ 1657.370301] pid[5906] compare_and_swap_function: counter: 14
[ 1657.370304] pid[5909] compare_and_swap_function: counter: 15
```

- When removing module



```
[ 1659.932640] compare_and_swap_module_cleanup: Exiting Compare and Swap Module!
```

Assignment-10: Synchronization basic locking

- **Linked list with synchronization**
  - Protect linked list operations (such as insert, search, and delete) by using three different locking mechanisms in your kernel module with four kernel threads.
    - Spinlock
    - Mutex
    - RW semaphore

Assignment-10: Synchronization basic locking

- **Linked list with synchronization scenario**
  - Four kernel thread runs simultaneously.
  - Each kernel thread perform Insert, Search, Delete operations to the global linked list, which is shared by all threads.
  - To protect global linked list from corruption by concurrent access of the threads, single global lock is used.
  - Each thread perform Insert, Search, Delete for its own data range bound.

Assignment-10: Synchronization basic locking

Linked list with synchronization

- **Tips.**
  - Create four threads in the module initalization procedure by using **work_fn()** on the next slide.
  - Stop four threads in the module cleanup procedure by using kthread_stop().
  - You might need **delay.h** header file provided by Linux kernel.

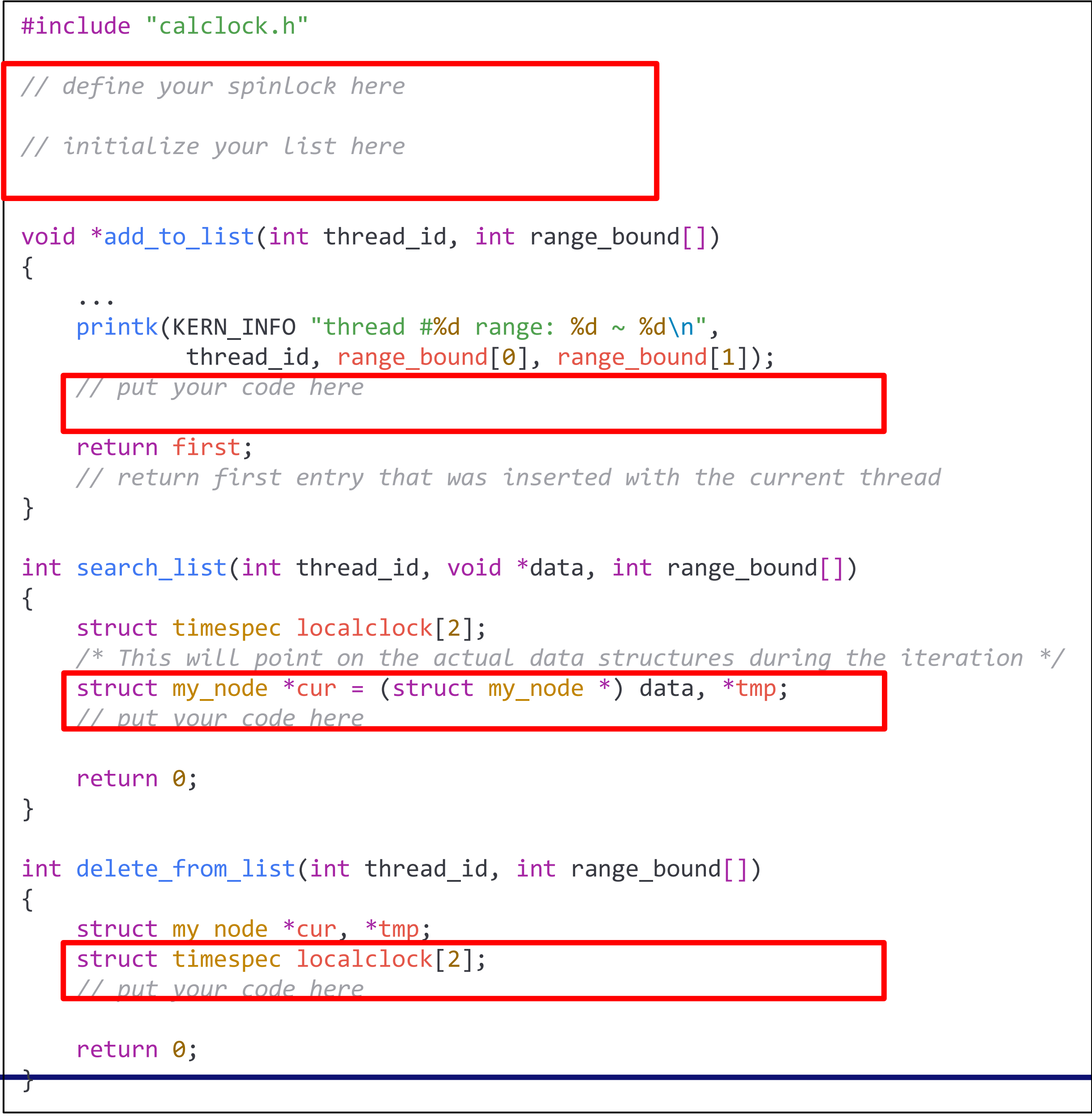
Assignment-10: Synchronization basic locking

❖ Worker function example

```c
static int work_fn(void *data)
{
    int range_bound[2];
    int thread_id = *(int*) data;

    set_iter_range(thread_id, range_bound);
    void *ret = add_to_list(thread_id, range_bound);
    search_list(thread_id, ret, range_bound);
    delete_from_list(thread_id, range_bound);

    while(!kthread_should_stop()) {
        msleep(500);
    }
    printk(KERN_INFO "thread #%d stopped!\n", thread_id);

    return 0;
}
```

Assignment-10: Synchronization basic locking

- **Linked list with synchronization example**
  - You have to fill in the blank and complete the code **linked_list_impl.c**
  - Different implementation is okay for the submission.



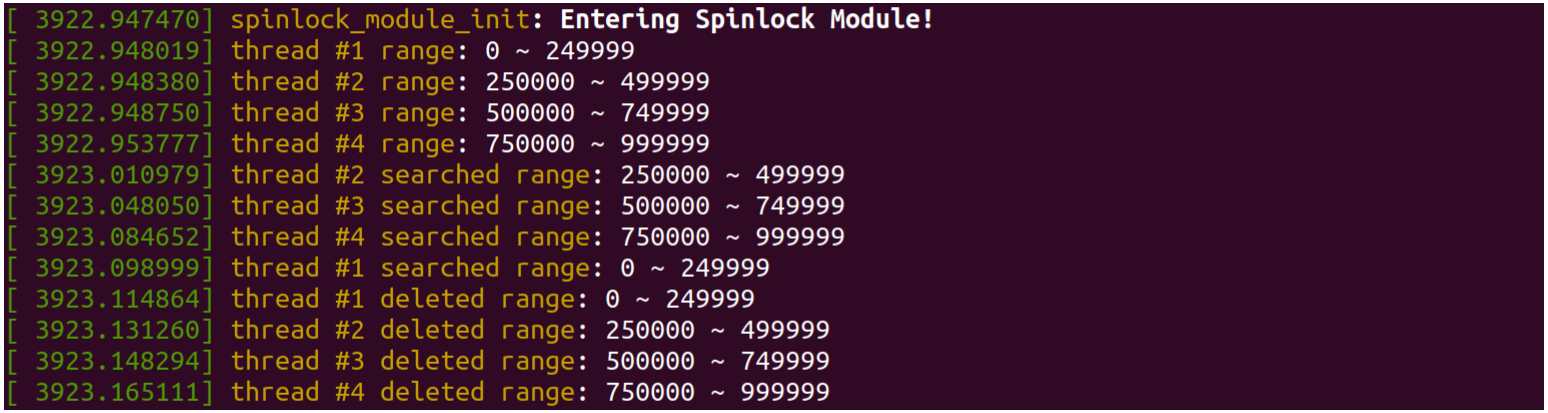
```
#include "calclock.h"

// define your spinlock here

// initialize your list here

void *add_to_list(int thread_id, int range_bound[])
{
    ...
    printk(KERN_INFO "thread #%d range: %d ~ %d\n",
            thread_id, range_bound[0], range_bound[1]);
    // put your code here

    return first;
    // return first entry that was inserted with the current thread
}

int search_list(int thread_id, void *data, int range_bound[])
{
    struct timespec localclock[2];
    /* This will point on the actual data structures during the iteration */
    struct my_node *cur = (struct my_node *) data, *tmp;
    // put your code here

    return 0;
}

int delete_from_list(int thread_id, int range_bound[])
{
    struct my_node *cur, *tmp;
    struct timespec localclock[2];
    // put your code here

    return 0;
}
```

Example screenshot 1 basic locking

❖ Linked list with synchronization

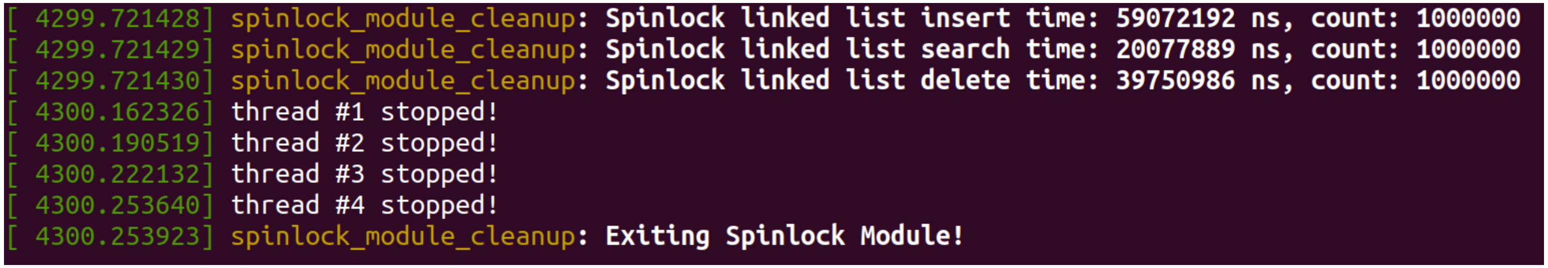
- Please follow the below template when printing out

- Insert, Search, Delete example after inserting module



```
[ 3922.947470] spinlock_module_init: Entering Spinlock Module!
[ 3922.948019] thread #1 range: 0 ~ 249999
[ 3922.948380] thread #2 range: 250000 ~ 499999
[ 3922.948750] thread #3 range: 500000 ~ 749999
[ 3922.953777] thread #4 range: 750000 ~ 999999
[ 3923.010979] thread #2 searched range: 250000 ~ 499999
[ 3923.048050] thread #3 searched range: 500000 ~ 749999
[ 3923.084652] thread #4 searched range: 750000 ~ 999999
[ 3923.098999] thread #1 searched range: 0 ~ 249999
[ 3923.114864] thread #1 deleted range: 0 ~ 249999
[ 3923.131260] thread #2 deleted range: 250000 ~ 499999
[ 3923.148294] thread #3 deleted range: 500000 ~ 749999
[ 3923.165111] thread #4 deleted range: 750000 ~ 999999
```

- When removing module



```
[ 4299.721428] spinlock_module_cleanup: Spinlock linked list insert time: 59072192 ns, count: 1000000
[ 4299.721429] spinlock_module_cleanup: Spinlock linked list search time: 20077889 ns, count: 1000000
[ 4299.721430] spinlock_module_cleanup: Spinlock linked list delete time: 39750986 ns, count: 1000000
[ 4300.162326] thread #1 stopped!
[ 4300.190519] thread #2 stopped!
[ 4300.222132] thread #3 stopped!
[ 4300.253640] thread #4 stopped!
[ 4300.253923] spinlock_module_cleanup: Exiting Spinlock Module!
```

What to submit

Atomic operations

- Short summary of each atomic operation.
- Code screenshot of each atomic operation module file.
- Screenshot of dmesg after successful insertion & removal of module for each atomic operation.

Linked list with synchronization

- Code screenshot of each locking mechanism module's `linked_list_impl.c` file.
- Screenshot of time measure result of each operation (insert, search, delete) at 1,000,000 nodes (250,000 per thread) while using spinlock, mutex, RW semaphore each.

What to submit

- **Submission format should be pdf.**
  - Make sure to include `linked_list_impl.c` screenshot and **dmesg** screenshots in submitted PDF.
  - Make sure to include your name and student id.