# Assignment-Bonus

## Linux System and its Applications

**Systems and Storage Laboratory**
**Department of Computer Science and Engineering**
**Chung-Ang University**

# Assignment-Bonus: lock-free linked list

1. **Implement a lock-free linked list in lecture slide [8. Linux Kernel Data Structure (new)]**
2. **Compare the execution time with the spin-locked linked list with the same condition**

❖ **What to submit**

- Screenshot of source code
  - Lock-free linked list
  - Spin-locked linked list
- Screenshot of each result screen

❖ **Submit within pdf format**

- Make sure to include your name and student id

# Data structure

- **Container of the list which contains every cat inside**

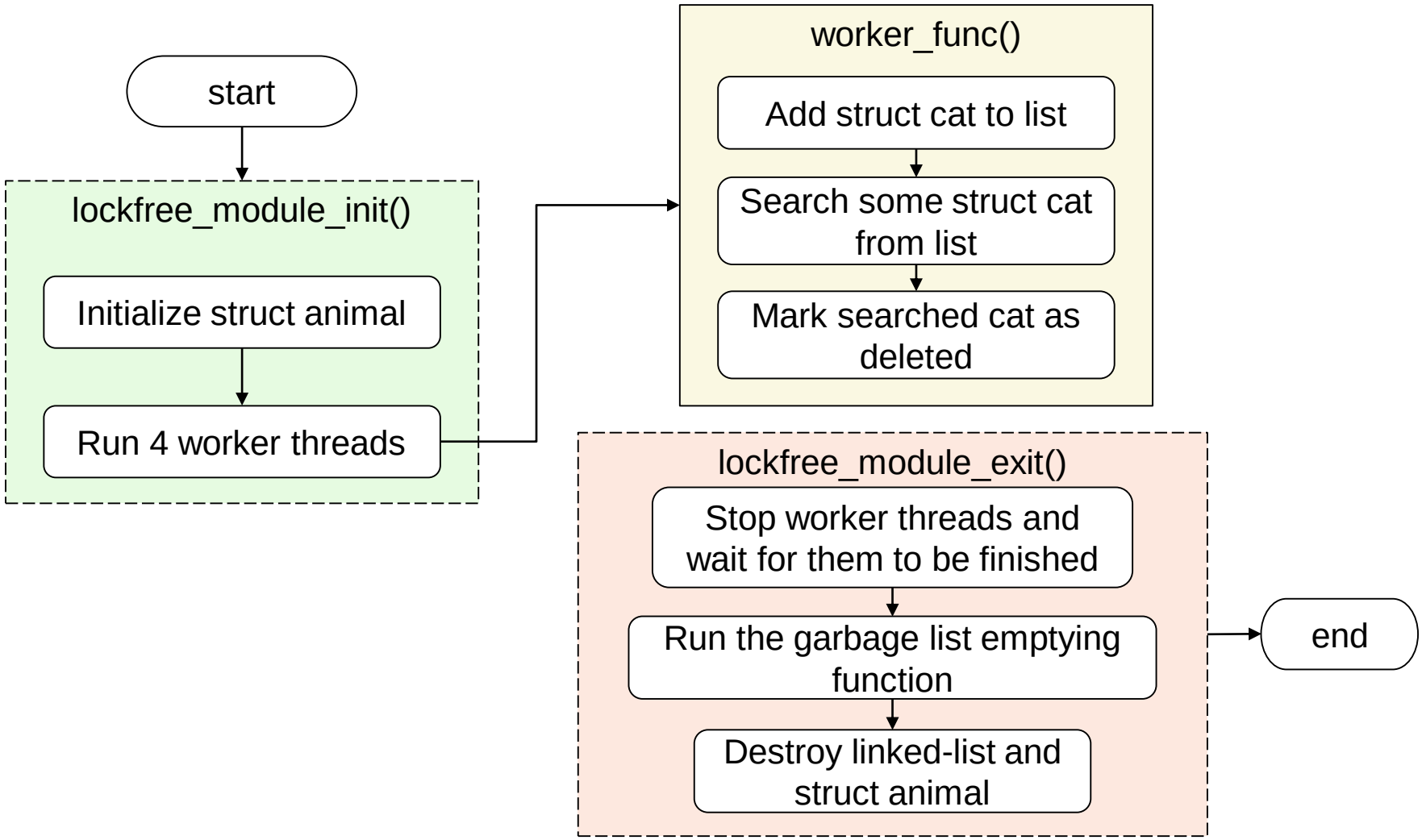
```
struct animal {
    int total;   /* number of entries in the list */
    struct list_head entry;   /* head of the list */
    atomic_t removed;   /* boolean for marking as deleted */
    struct list_head gc_entry;   /* head of the garbage list */
};
```

- **List element**

```
struct cat {
    int var;   /* id of this cat */
    struct list_head entry;   /* list entry to be inserted */
    atomic_t removed;   /* boolean for marking as deleted */
    struct list_head gc_entry;   /* garbage list entry */
};
```

# Flowchart for lock-free linked list module

- **Perform tasks by following order**



# Initializing module

1. **Initialize struct animal**
2. **Run 4 worker threads**

```
int __init lockfree_module_init(void)
{
    printk("%s: Entering Lock-free Module!\n", __func__);

    /*  initialize struct animal here  */

    /* initialize animal's list head and gc_list head here */

    /* start each thread here */

    return 0;
}
```

# Thread work

- **Work function for each worker thread should perform**

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

    set_iter_range(thread_id, range_bound);
    add_to_list(thread_id, range_bound);
    err = search_list(thread_id, range_bound);
    if (!err)
        delete_from_list(thread_id, range_bound);

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

    return 0;
}
```

*Initialize range boundary according to the thread id*

# Insert

## ❖ add new cat into animal list

```
void add_to_list(int thread_id, int range_bound[])
{
    struct timespec localclock[2];
    struct cat *new, *first = NULL;
    int i;

    for (i = range_bound[0]; i < range_bound[1] + 1; i++) {
        getrawmonotonic(&localclock[0]);
        /* initialize new cat here */


        if (!first)
            first = new;
        /* initialize cat's list head and gc_list head here */


        /* add new cat into animal's list */


        getrawmonotonic(&localclock[1]);
        calclock(localclock, &add_to_list_time, &add_to_list_count);
    }

    printk(KERN_INFO "thread #%d: inserted cat #%d-%d to the list, total: %u cats\n",
           thread_id, first->var, new->var, head->total);
}
```

*" i " is id of new cat*

# Search

- **Find cat with target_idx and print it out**
  - Return: 0 on success, ENODATA when no matching entry in the list

```
int search_list(int thread_id, int range_bound[])
{
    struct list_head *entry, *iter = &head->entry;
    /* This will point on the actual data structures during the iteration */
    struct cat *cur;
    struct timespec localclock[2];
    int target_idx = select_target_index(range_bound);

    /* iterate over the list, skip removed entry,
        search cat with target index */


    return -ENODATA;
}
```

*Use this cursor for every iteration*

# Delete

- **Search for the list, add cats within the range_bound into garbage list**
  - Delete target: target_idx ~ range_bound[1]

*target_idx cat's cur->var*

*range_bound[1] cat's cur->var*

```
void delete_from_list(int thread_id, int range_bound[])
{
    int start = -1, end = -1;
    struct timespec localclock[2];
    struct list_head *entry, *iter = &head->entry;
    /* This will point on the actual data structures during the iteration */
    struct cat *cur;
    int target_idx = select_target_index(range_bound);

    /* iterate over the list, skip removed entry */
    while (...) {


        /* add cat into garbage list if its id is within target range */


    }
    printk(KERN_INFO "thread #%d: marked cat #%d-%d as deleted, total: %d cats\n",
            thread_id, start, end, head->total);
}
```

# Adding target entry into garbage list

- Mark entry in the list as deleted and add to garbage list

```
static void add_to_garbage_list(int thread_id, void *data)
{
    struct cat *target = (struct cat *) data;

    /* set cat as deleted */

    /* add to garbage list */

    /* decrease total cats in struct animal */

}
```

*Cat to be marked as delete*

# Cleaning up the module

1. Stop worker threads and wait for them to be finished
2. Run the garbage list emptying function
3. Destroy linked-list and struct animal

```
void __exit lockfree_module_cleanup(void)
{
    /*  print calclock result here  */

    /* stop every thread here */

    empty_garbage_list();
    printk("After gc: total %d cats\n", head->total);

    destroy_list();
    printk("After destroyed list: total %d cats\n", head->total);
    kfree(head);

    printk("%s: Exiting Lock-free Module!\n", __func__);
}
```

# Deallocate garbages

- Garbage list emptying function

```
int empty_garbage_list(void)
{
    struct cat *cur;
    struct list_head *entry, *iter = &head->gc_entry;
    unsigned int counter = 0;

    /* iterate garbage list,
       remove each entry from every list and free it */


    printk(KERN_INFO "%s: freed %u cats\n", __func__, counter);

    return 0;
}
```

# Destroy list

```c
void destroy_list(void)
{
    struct cat *cur;
    struct list_head *entry, *iter = &head->entry;

    /* iterate the list,
        remove each entry from every list and free it */

}
```

# Output Template

## ❖ Linked list with lock-free method

- Please follow the below template when printing out

- Insert, Search, Delete template

```
[  719.595065] lockfree_module_init: Entering Lock-free Module!
[  719.658092] thread #2: inserted cat #250000-499999 to the list, total: 789498 cats
[  719.670912] thread #4: inserted cat #750000-999999 to the list, total: 926906 cats
[  719.679475] thread #3: inserted cat #500000-749999 to the list, total: 987400 cats
[  719.682840] thread #1: inserted cat #0-249999 to the list, total: 1000000 cats
[  719.702335] thread #2: found cat #375000
[  719.744580] thread #4: found cat #875000
[  719.770243] thread #3: found cat #625000
[  719.781604] thread #1: found cat #125000
[  719.834387] thread #2: marked cat #375000-499999 as deleted, total: 839327 cats
[  719.885076] thread #4: marked cat #875000-999999 as deleted, total: 627365 cats
[  719.904800] thread #3: marked cat #625000-749999 as deleted, total: 512601 cats
[  719.906565] thread #1: marked cat #125000-249999 as deleted, total: 500000 cats
```

- When removing module

```
[  722.581721] lockfree_module_cleanup: Lock-free linked list insert time: 215863027 ns, count: 1000000
[  722.581722] lockfree_module_cleanup: Lock-free linked list search time: 74219819 ns, count: 2014374
[  722.581722] lockfree_module_cleanup: Lock-free linked list delete time: 187962042 ns, count: 2957463
[  722.956388] thread #1 stopped!
[  723.207121] thread #2 stopped!
[  723.468431] thread #3 stopped!
[  723.979155] thread #4 stopped!
[  724.005899] empty_garbage_list: freed 500000 cats
[  724.005900] After gc: total 500000 cats
[  724.027160] After destroyed list: total 0 cats
[  724.027161] lockfree_module_cleanup: Exiting Lock-free Module!
```

# Another Bonus: Lock-free XArray

- **Implement Lock-free XArray**
  - Lock-free insert node/items, update node/items, delete items
    - Use Compare-And-Swap in xas_store()
  - Lock-free search
  - Lock-free delete node
    - Use logically removing technique and GC list