20183856 Lim jongseung

Atomic operation based (code)

Activities Terminal

limjs@limjs: ~/cau_linux_system/pxt4/atomic

```
#include <linux/delay.h>

//define my spin lock
struct lock {
        int ticket;
        int turn;
};

void init_lock(struct lock *lock_v){
        lock_v->ticket = 0;
        lock_v->turn =0;
}

void acquire(struct lock *lock_v){
        int myturn = __sync_fetch_and_add(&lock_v->ticket, 1);
        while(lock_v->turn != myturn);
};

void release(struct lock *lock_v){
        lock_v->turn = lock_v->turn+1;
}

int counter;
struct lock counter_lock;
struct task_struct *work_thread1, *work_thread2, *work_thread3, *work_thread4;

static int work_fn (void *data)
{
        int original;

        while(!kthread_should_stop()) {
                //critical section
                acquire(&counter_lock);
                counter++;
                original=counter;
                release(&counter_lock);
                //end of the critical section
                printk(KERN_INFO "pid[%u] %s: counter: %d\n", current->pid, __func__,original);
                msleep(500);
        }

        do_exit(0);
}

static int __init my_mod_init(void)
{
        printk("%s, Entering module\n", __func__);
        counter = 0;
        init_lock(&counter_lock);
        work_thread1 = kthread_run(work_fn, NULL, "work_fn");
        work_thread2 = kthread_run(work_fn, NULL, "work_fn");
        work_thread3 = kthread_run(work_fn, NULL, "work_fn");
        work_thread4 = kthread_run(work_fn, NULL, "work_fn");

        return 0;
}

static void __exit my_mod_exit(void)
{
        kthread_stop(work_thread1);
        kthread_stop(work_thread2);
        kthread_stop(work_thread3);
        kthread_stop(work_thread4);
        printk("%s, Exiting module\n",__func__);
}

module_init(my_mod_init);
module_exit(my_mod_exit);

MODULE_LICENSE("GPL");
```

Ticket_lock based

Used fetch_and_add

1. Acquire lock
2. Do critical section
3. Release lock

Atomic operation based (dmesg)

```
[16667.461640] pid[244607] work_fn: counter: 227
[16667.461642] pid[244605] work_fn: counter: 228
[16667.973592] pid[244607] work_fn: counter: 229
[16667.973595] pid[244605] work_fn: counter: 230
[16667.973609] pid[244606] work_fn: counter: 231
[16667.973610] pid[244608] work_fn: counter: 232
[16668.484657] pid[244606] work_fn: counter: 233
[16668.484675] pid[244605] work_fn: counter: 234
[16668.485036] pid[244608] work_fn: counter: 235
[16668.485047] pid[244607] work_fn: counter: 236
[16668.997191] pid[244605] work_fn: counter: 237
[16668.997193] pid[244607] work_fn: counter: 238
[16668.997210] pid[244608] work_fn: counter: 239
[16668.997211] pid[244606] work_fn: counter: 240
[16669.508459] pid[244608] work_fn: counter: 241
[16669.508464] pid[244607] work_fn: counter: 242
[16669.509040] pid[244606] work_fn: counter: 243
[16669.509043] pid[244605] work_fn: counter: 244
[16670.020705] pid[244607] work_fn: counter: 245
[16670.020706] pid[244606] work_fn: counter: 246
[16670.020716] pid[244605] work_fn: counter: 247
[16670.020721] pid[244608] work_fn: counter: 248
[16670.532722] pid[244608] work_fn: counter: 249
[16670.532741] pid[244607] work_fn: counter: 250
[16670.532928] pid[244605] work_fn: counter: 251
[16670.532942] pid[244606] work_fn: counter: 252
[16671.046530] pid[244607] work_fn: counter: 253
[16671.046532] pid[244606] work_fn: counter: 254
[16671.046555] pid[244608] work_fn: counter: 255
[16671.046557] pid[244605] work_fn: counter: 256
[16671.563541] pid[244608] work_fn: counter: 258
[16671.563544] pid[244605] work_fn: counter: 257
[16671.563560] pid[244606] work_fn: counter: 259
[16671.563561] pid[244607] work_fn: counter: 260
[16672.069531] pid[244607] work_fn: counter: 261
[16672.069555] pid[244608] work_fn: counter: 262
[16672.071305] pid[244606] work_fn: counter: 263
[16672.071316] pid[244605] work_fn: counter: 264
[16672.578029] my_mod_exit, Exiting module
limjs@limjs:~/cau_linux_system/pxt4/atomic$ 20183856 Lim jongseung
```

Linked list manipulation (spinlock – code(1))

Activities Terminal

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/kthread.h>
#include <linux/sched.h>
#include <linux/delay.h>
//lock header file import
#include <linux/spinlock.h>
#include <linux/mutex.h>
#include <linux/list.h>
#include "../calclock.c"

//define my spin lock
spinlock_t s_lock;

//initialize my linked list
struct my_node {
        struct list_head list;
        int data;
};

LIST_HEAD(my_list);

int counter;

int add_counter;
int srch_counter;
int del_counter;

KTDEF(add_to_list);
KTDEF(search_list);
KTDEF(delete_from_list);
```

Included spinlock.h

Continue;

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist | limjs@limjs: ~/cau_linux_system/pxt4/linked

```c
KTDEF(search_list);
KTDEF(delete_from_list);

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 my codes here
        spin_lock(&s_lock);
        int i;
        for(i=range_bound[0]; i<=range_bound[1]; i++) {
                struct my_node *new = kmalloc(sizeof(struct my_node),GFP_KERNEL);
                new->data = i;
                list_add(&new->list,&my_list);
        }
        spin_unlock(&s_lock);
        add_counter++;
        //first is seemed head of list
        return &my_list;
}

int search_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d search range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
        /* This will point on the actual data structure during the iteration */
        struct my_node *cur, *tmp;
        //put my codes here
        spin_lock(&s_lock);
        list_for_each_entry(cur, &my_list, list){
                //empty?
        }
        spin_unlock(&s_lock);
        srch_counter++;
        return 0;
}

int delete_from_list(int thread_id, int range_bound[])
{
```

Acquire s_lock

1. Generate node
2. It's data is it's generated index
3. Add behind to the head

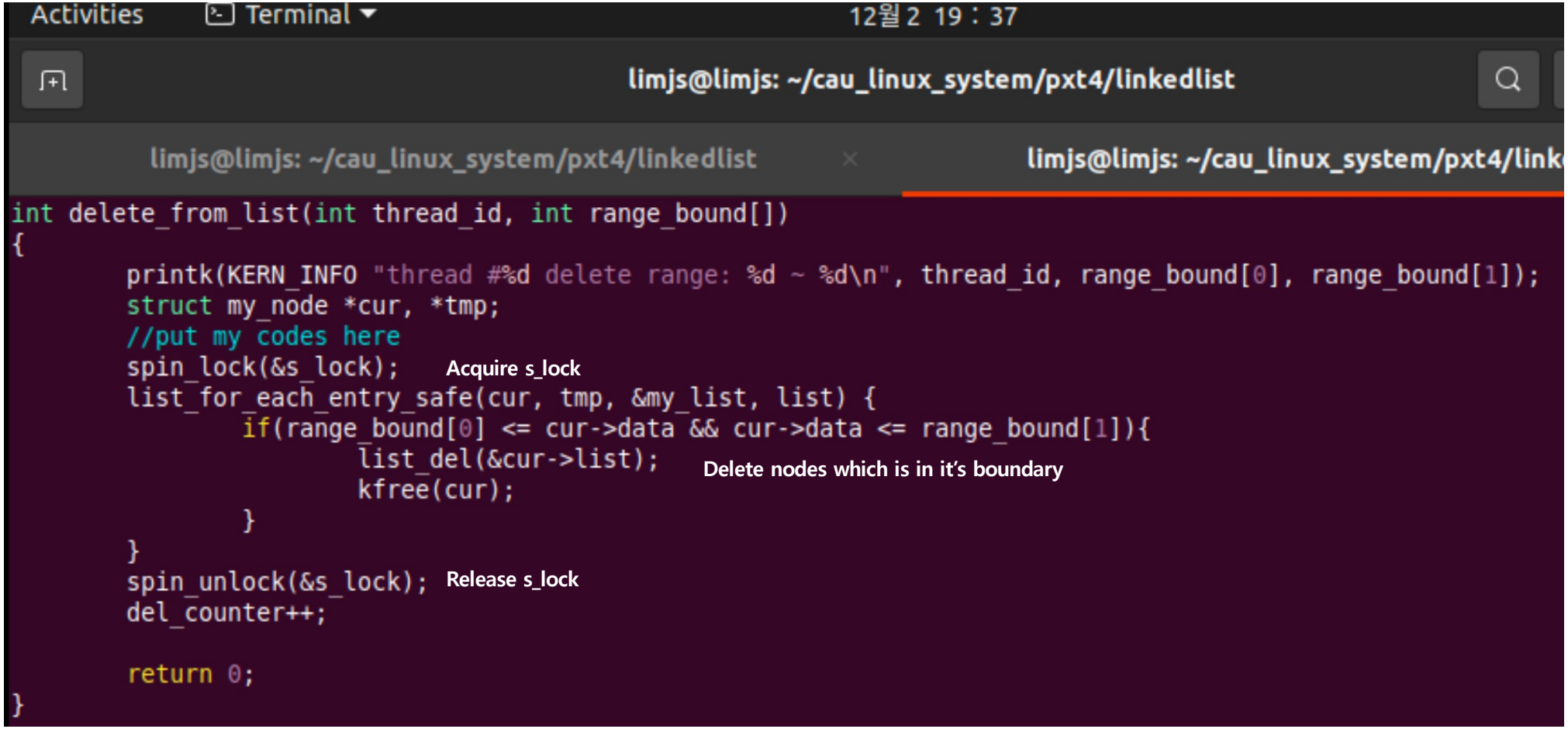
Release s_lock

Acquire s_lock

Itinerate list

Release s_lock

Linked list manipulation (spinlock – code(2))



Linked list manipulation (spinlock – code(3))

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

```
        return 0;
}

static int control_func(void * data){
        //call add, search, delete here (with critical section control)
        //and set other parameters for those functions
        int thread_id = counter++;
        int bound[2]={(thread_id-1)*250000,249999+(thread_id-1)*250000};
        //timespec
        int ret;
        ktime_t stopwatch[2];
        //add
        ktget(&stopwatch[0]);
        ret = add_to_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,add_to_list);
        //search
        ktget(&stopwatch[0]);
        ret=search_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,search_list);
        //delete
        ktget(&stopwatch[0]);
        ret=delete_from_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,delete_from_list);
        while(!kthread_should_stop()) {
                msleep(500);
        }
        printk(KERN_INFO "thread #%d stopped!\n", thread_id);
        return 0;
}

struct task_struct *work_thread1, *work_thread2, *work_thread3, *work_thread4;

static int __init my_mod_init(void)
```

limjs@limjs:

```
struct task_struct *work_thread1, *work_thread2, *work_thread3, *work_thread4;

static int __init my_mod_init(void)
{
        printk("%s, Entering module(spinlock)\n", __func__);
        counter=1;
        add_counter = 0;
        srch_counter = 0;
        del_counter = 0;
        INIT_LIST_HEAD(&my_list);

        spin_lock_init(&s_lock);

        work_thread1 = kthread_run(control_func, NULL, "kt1 work");
        work_thread2 = kthread_run(control_func, NULL, "kt2 work");
        work_thread3 = kthread_run(control_func, NULL, "kt3 work");
        work_thread4 = kthread_run(control_func, NULL, "kt4 work");

        return
}

KTDEC(add_to_list);
KTDEC(search_list);
KTDEC(delete_from_list);
static void __exit my_mod_exit(void)
{
        ktprint(1,add_to_list);
        ktprint(1,search_list);
        ktprint(1,delete_from_list);
        kthread_stop(work_thread1);
        kthread_stop(work_thread2);
        kthread_stop(work_thread3);
        kthread_stop(work_thread4);
        list_del_init(&my_list);
        printk("%s, Exiting module(spinlock)\n",__func__);
}
```

Each part call insert/search/delete function & measure timespec with calclock

Linked list manipulation (spinlock – dmesg)

```
[59638.981330] my_mod_init, Entering module(spinlock)
[59638.981585] thread #1 range: 0 ~ 249999
[59638.982094] thread #2 range: 250000 ~ 499999
[59638.982680] thread #3 range: 500000 ~ 749999
[59638.983026] thread #4 range: 750000 ~ 999999
[59638.999450] thread #1 search range: 0 ~ 249999
[59639.010314] thread #2 search range: 250000 ~ 499999
[59639.020315] thread #3 search range: 500000 ~ 749999
[59639.028048] thread #4 search range: 750000 ~ 999999
[59639.052466] thread #1 delete range: 0 ~ 249999
[59639.076516] thread #2 delete range: 250000 ~ 499999
[59639.099823] thread #3 delete range: 500000 ~ 749999
[59639.122852] thread #4 delete range: 750000 ~ 999999
[59648.620294]     add_to_list is called 4 times, and the time
 interval is 128,736,208ns (per thread is 32,184,052ns) (100.0
0%)
[59648.620301]     search_list is called 4 times, and the time
 interval is 293,530,042ns (per thread is 73,382,510ns) (100.0
0%)
[59648.620303]     delete_from_list is called 4 times, and the
 time interval is 337,832,416ns (per thread is 84,458,104ns) (
100.00%)
[59648.870147] thread #1 stopped!
[59648.901440] thread #2 stopped!
[59648.901941] thread #3 stopped!
[59648.933041] thread #4 stopped!
[59648.933096] my_mod_exit, Exiting module(spinlock)
```

Linked list manipulation (mutex – code(1))

Activities Terminal

limjs@limjs: ~/cau_linux_syster

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/kthread.h>
#include <linux/sched.h>
#include <linux/delay.h>
//lock header file import
#include <linux/spinlock.h>
#include <linux/mutex.h>
#include <linux/list.h>
#include "../calclock.c"

//define my spin lock
struct mutex my_mutex;

//initialize my linked list
struct my_node {
        struct list_head list;
        int data;
};

LIST_HEAD(my_list);

int counter;

int add_counter;
int srch_counter;
int del_counter;

KTDEF(add_to_list);
KTDEF(search_list);
KTDEF(delete_from_list);
```

Included mutex.h

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

limjs@limjs: ~/cau_linux_system/pxt4/link

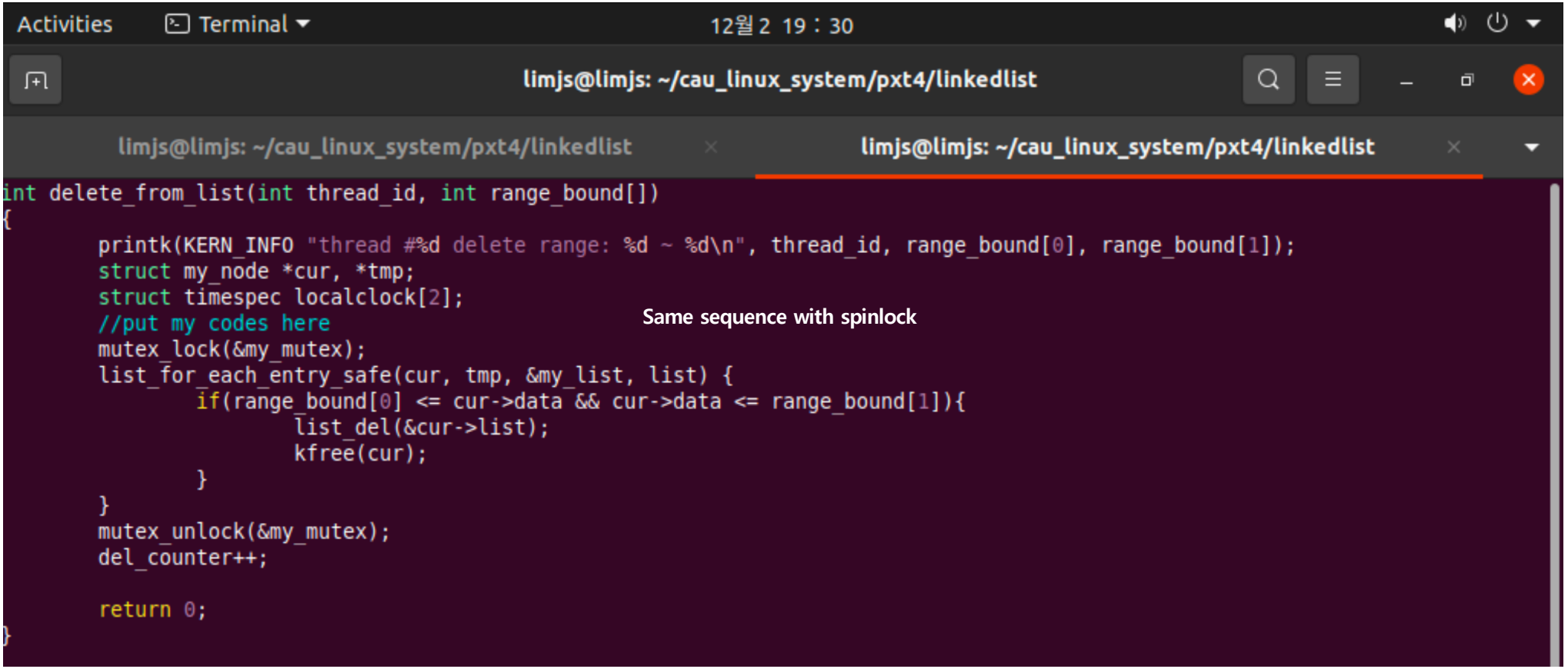
```c
void *add_to_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
        //put my codes here
        mutex_lock(&my_mutex);
        int i;
        for(i=range_bound[0]; i<=range_bound[1]; i++) {
                struct my_node *new = kmalloc(sizeof(struct my_node),GFP_KERNEL);
                new->data = i;
                list_add(&new->list,&my_list);
        }
        mutex_unlock(&my_mutex);
        add_counter++;
        //first is seemed head of list
        return &my_list;
}

int search_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d search range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
        struct timespec localclock[2];
        /* This will point on the actual data structure during the iteration */
        struct my_node *cur, *tmp;
        //put my codes here
        mutex_lock(&my_mutex);
        list_for_each_entry(cur, &my_list, list){
                //empty?
        }
        mutex_unlock(&my_mutex);
        srch_counter++;
        return 0;
}
```

Same sequence with spinlock

Same sequence with spinlock

Linked list manipulation (mutex – code(2))




```
int delete_from_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d delete range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
        struct my_node *cur, *tmp;
        struct timespec localclock[2];
        //put my codes here
        mutex_lock(&my_mutex);
        list_for_each_entry_safe(cur, tmp, &my_list, list) {
                if(range_bound[0] <= cur->data && cur->data <= range_bound[1]){
                        list_del(&cur->list);
                        kfree(cur);
                }
        }
        mutex_unlock(&my_mutex);
        del_counter++;

        return 0;
}
```

Linked list manipulation (mutex – code(3))

Activities Terminal

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

```
static int control_func(void * data){
        //call add, search, delete here (with critical section control)
        //and set other parameters for those functions
        int thread_id = counter++;
        int bound[2]={(thread_id-1)*250000,249999+(thread_id-1)*250000};
        //timespec
        int ret;
        ktime_t stopwatch[2];
        //add
        ktget(&stopwatch[0]);
        ret = add_to_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,add_to_list);
        //search
        ktget(&stopwatch[0]);
        ret=search_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,search_list);
        //delete
        ktget(&stopwatch[0]);
        ret=delete_from_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,delete_from_list);
        while(!kthread_should_stop()) {
                msleep(500);
        }
        printk(KERN_INFO "thread #%d stopped!\n", thread_id);
        return 0;
}

struct task_struct *work_thread1, *work_thread2, *work_thread3, *work_thread4;
```

Common parts (Only difference is initialization of lock)

Activities Terminal 12월 2 19 : 3

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

```
static int __init my_mod_init(void)
{
        printk("%s, Entering module(mutex)\n", __func__);
        counter=1;
        add_counter = 0;
        srch_counter = 0;
        del_counter = 0;
        INIT_LIST_HEAD(&my_list);

        mutex_init(&my_mutex);

        work_thread1 = kthread_run(control_func, NULL, "kt1 work");
        work_thread2 = kthread_run(control_func, NULL, "kt2 work");
        work_thread3 = kthread_run(control_func, NULL, "kt3 work");
        work_thread4 = kthread_run(control_func, NULL, "kt4 work");

        return 0;

KTDEC(add_to_list);
KTDEC(search_list);
KTDEC(delete_from_list);
static void __exit my_mod_exit(void)
{
        ktprint(1,add_to_list);
        ktprint(1,search_list);
        ktprint(1,delete_from_list);
        kthread_stop(work_thread1);
        kthread_stop(work_thread2);
        kthread_stop(work_thread3);
        kthread_stop(work_thread4);
        list_del_init(&my_list);
        printk("%s, Exiting module(mutex)\n",__func__);
}

module_init(my_mod_init);
module_exit(my_mod_exit);
```

Linked list manipulation (mutex – dmesg)

```
[59222.408982] my_mod_init, Entering module(mutex)
[59222.409116] thread #1 range: 0 ~ 249999
[59222.409219] thread #2 range: 250000 ~ 499999
[59222.409359] thread #3 range: 500000 ~ 749999
[59222.411074] thread #4 range: 750000 ~ 999999
[59222.440487] thread #1 search range: 0 ~ 249999
[59222.446389] thread #1 delete range: 0 ~ 249999
[59222.464859] thread #2 search range: 250000 ~ 499999
[59222.477046] thread #2 delete range: 250000 ~ 499999
[59222.496481] thread #3 search range: 500000 ~ 749999
[59222.513344] thread #3 delete range: 500000 ~ 749999
[59222.532847] thread #4 search range: 750000 ~ 999999
[59222.598950] thread #4 delete range: 750000 ~ 999999
[59224.550164]     add_to_list is called 4 times, and the time
 interval is 295,901,500ns (per thread is 73,975,375ns) (100.0
0%)
[59224.550173]     search_list is called 4 times, and the time
 interval is 101,052,959ns (per thread is 25,263,239ns) (34.15
%)
[59224.550175]     delete_from_list is called 4 times, and the
 time interval is 302,574,709ns (per thread is 75,643,677ns) (
100.00%)
[59224.610481] thread #1 stopped!
[59224.642233] thread #2 stopped!
[59224.642331] thread #3 stopped!
[59224.674292] thread #4 stopped!
[59224.674360] my_mod_exit, Exiting module(mutex)
```

Linked list manipulation (rw_semaphore – code(1))

Activities

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/slab.h>
#include <linux/kthread.h>
#include <linux/sched.h>
#include <linux/delay.h>
//lock header file import
#include <linux/semaphore.h>
#include <linux/spinlock.h>
#include <linux/mutex.h>
#include <linux/list.h>
#include "../calclock.c"

//define my spin lock
struct rw_semaphore rwsem;

//initialize my linked list
struct my_node {
        struct list_head list;
        int data;
};

LIST_HEAD(my_list);

int counter;

int add_counter;
int srch_counter;
int del_counter;

KTDEF(add_to_list);
KTDEF(search_list);
KTDEF(delete_from_list);
```

Included semaphore.h

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

limjs@limjs: ~/cau_linux_system/pxt4/linke

```
KTDEF(delete_from_list);

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 my codes here
        down_write(&rwsem);
        int i;
        for(i=range_bound[0]; i<=range_bound[1]; i++) {
                struct my_node *new = kmalloc(sizeof(struct my_node),GFP_KERNEL);
                new->data = i;
                list_add(&new->list,&my_list);
        }
        up_write(&rwsem);
        add_counter++;
        //first is seemed head of list
        return &my_list;
}

int search_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d search range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
        /* This will point on the actual data structure during the iteration */
        struct my_node *cur, *tmp;
        //put my codes here
        down_read(&rwsem);
        list_for_each_entry(cur, &my_list, list){
                //empty?
        }
        up_read(&rwsem);
        srch_counter++;
        return 0;
}

int delete_from_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d delete range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
```

Same sequence with spinlock
(Only difference is lock is separated with read/write.
In here, write semaphore is used for insert)

(In here, read semaphore is used for search)

Linked list manipulation (rw_semaphore – code(2))

```
int delete_from_list(int thread_id, int range_bound[])
{
        printk(KERN_INFO "thread #%d delete range: %d ~ %d\n", thread_id, range_bound[0], range_bound[1]);
        struct my_node *cur, *tmp;
        //put my codes here
        down_write(&rwsem);
        list_for_each_entry_safe(cur, tmp, &my_list, list) {
                if(range_bound[0] <= cur->data && cur->data <= range_bound[1]){
                        list_del(&cur->list);
                        kfree(cur);
                }
        }
        up_write(&rwsem);
        del_counter++;

        return 0;
}
```

(In here, write semaphore is used for delete)

Linked list manipulation (rw_semaphore – code(3))

Activities

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

limjs@limj

```
}

static int control_func(void * data){
        //call add, search, delete here (with critical section control)
        //and set other parameters for those functions
        int thread_id = counter++;
        int bound[2]={(thread_id-1)*250000,249999+(thread_id-1)*250000};
        //timespec
        int ret;
        ktime_t stopwatch[2];
        //add
        ktget(&stopwatch[0]);
        ret = add_to_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,add_to_list);
        //search
        ktget(&stopwatch[0]);
        ret=search_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,search_list);
        //delete
        ktget(&stopwatch[0]);
        ret=delete_from_list(thread_id,bound);
        ktget(&stopwatch[1]);
        ktput(stopwatch,delete_from_list);
        while(!kthread_should_stop()) {
                msleep(500);
        }
        printk(KERN_INFO "thread #%d stopped!\n", thread_id);
        return 0;
}

struct task_struct *work_thread1, *work_thread2, *work_thread3, *work_thread4;

static int __init my_mod_init(void)
{
        printk("%s, Entering module(rw_semaphore)\n", __func__);
```

Common parts (Only difference is initialization of lock)

n/pxt4/link

limjs@limjs: ~/cau_linux_system/pxt4/linkedlist

limjs@limj

```
struct task_struct *work_thread1, *work_thread2, *work_thread3, *work_thread4;

static int __init my_mod_init(void)
{
        printk("%s, Entering module(rw_semaphore)\n", __func__);
        counter=1;
        add_counter = 0;
        srch_counter = 0;
        del_counter = 0;
        INIT_LIST_HEAD(&my_list);

        init_rwsem(&rwsem);

        work_thread1 = kthread_run(control_func, NULL, "kt1 work");
        work_thread2 = kthread_run(control_func, NULL, "kt2 work");
        work_thread3 = kthread_run(control_func, NULL, "kt3 work");
        work_thread4 = kthread_run(control_func, NULL, "kt4 work");

        return 0;
}

KTDEC(add_to_list);
KTDEC(search_list);
KTDEC(delete_from_list);
static void __exit my_mod_exit(void)
{
        ktprint(1,add_to_list);
        ktprint(1,search_list);
        ktprint(1,delete_from_list);
        kthread_stop(work_thread1);
        kthread_stop(work_thread2);
        kthread_stop(work_thread3);
        kthread_stop(work_thread4);
        list_del_init(&my_list);
        printk("%s, Exiting module(rw_semaphore)\n",__func__);
}
```

Linked list manipulation (rw_semaphore – dmesg)

```
[60078.788933] my_mod_init, Entering module(rw_semaphore)
[60078.789531] thread #1 range: 0 ~ 249999
[60078.790377] thread #2 range: 250000 ~ 499999
[60078.791521] thread #3 range: 500000 ~ 749999
[60078.791689] thread #4 range: 750000 ~ 999999
[60078.807140] thread #1 search range: 0 ~ 249999
[60078.820018] thread #1 delete range: 0 ~ 249999
[60078.828574] thread #2 search range: 250000 ~ 499999
[60078.839994] thread #2 delete range: 250000 ~ 499999
[60078.847248] thread #3 search range: 500000 ~ 749999
[60078.865169] thread #3 delete range: 500000 ~ 749999
[60078.871621] thread #4 search range: 750000 ~ 999999
[60078.895265] thread #4 delete range: 750000 ~ 999999
[60079.885417]     add_to_list is called 4 times, and the time
 interval is 191,458,250ns (per thread is 47,864,562ns) (100.0
0%)
[60079.885423]     search_list is called 4 times, and the time
 interval is 65,862,874ns (per thread is 16,465,718ns) (34.40%
)
[60079.885424]     delete_from_list is called 4 times, and the
 time interval is 364,017,084ns (per thread is 91,004,271ns) (
100.00%)
[60079.951456] thread #1 stopped!
[60079.982470] thread #2 stopped!
[60079.982608] thread #3 stopped!
[60079.982644] thread #4 stopped!
[60079.982683] my_mod_exit, Exiting module(rw_semaphore)
```

Linked list manipulation (comparision)

4.5x to minimum

2.4x to minimum

	spinlock	mutex	RW semaphore
add_to_list	128,736,208 ns	295,901,500 ns	191,458,250 ns
search_list	293,530, 042 ns	101,052,959 ns	65,862,874 ns
delete_from_list	337,832,416 ns	302,574,709 ns	364,017,084 ns

Not that different