CS 444 Group 15

Project 2 Writeup

Member:

Xiaomei WANG

Changxu YAN

Xilun GUO

supervised by

D. Kevin McGrath Brown

October 24, 2016

What do you think the main point of this assignment is?

In this assignment, we need to edit the Linux source code directly. It is a lot to consider and a lot to learn because if we want to change anything, we have to take into consideration of the impact on the whole. First, we need to know how to write and modify the I/O scheduler under this circumstances, then make it to work. We need to think more about the code and understand the mechanism to make it work. So, I think, the main point of this assignment is to get us working with the Linux source code, then know how the LOOK/C_LOOK I/O scheduler work.

How did you personally approach the problem? Design decisions, algorithm, etc.

Personally, we just look at the code and try to understand it first. We have to understand how it works before we try to make changes, which is, implement a scheduler. To understand the various structs took us a bit of time. Then we noticed that the Noop is the default scheduler, so we build our C_LOOK scheduler based on the Noop.

How did you ensure your solution was correct? Testing details, for instance.

We use printk to print out the debugging information and change our solution according to it.

What did you learn?

We learned how to make an I/O scheduler with simple FIFO based algorithm. Also, after go through the Linux source code, we learned quite a bit about how different parts connects.

The design you plan to use to implement the SSTF algorithms.

C_LOOK is a disk scheduling algorithm. It begins to scan from the starting point to the nearest end; after reaching the lowest/highest, it jumps to the other end and moves in the same direction, until it reaches all the element in the queue.

Version control log

Date	Person	Message
Mon Oct 24 22:45:32	Changxu	Final Adjustment
Mon Oct 24 22:31:34	Xiaomei	Update Writeup
Mon Oct 24 20:09:51	Changyu	Upload the main file
Sun Oct 23 01:18:29	Xiaomei	Cleanup dir
Fri Oct 21 16:49:09	Changxu	Looed in to noop
Fri Oct 21 16:23:15	Xilun	Added tex file; noop
Thu Oct 13 20:21:21	Changxu	Initial setup, update README.md

Work Log

Date	Person	Task
Fri Oct 14	Changxu	Locate/find noop-iosched.c
Fri Oct 14	Xilun	Read and analyze the structure in noop-iosched.c
Fri Oct 14	Xiaomei	Prepare the writeup tex file
Fri Oct 21	Xilun	Learned how to perform C-LOOK disk scheduling algorithm
Fri Oct 21	Xiaoemi	Learned how to perform C-LOOK disk schefuling algorithm
Fri Oct 21	Changxu	Modify noop-iosched.c to fit C-LOOK algorithm
Mon Oct 24	Xilun	Debugging
Mon Oct 24	Xiaomei	Debugging
Mon Oct 24	Xiaomei	Finish the writeup
Mon Oct 24	Chnagxu	Debugging

Appendix 1: Look elevator code

```c
/*
 *elevator c-look
 */
#include <linux/blkdev.h>
#include <linux/elevator.h>
#include <linux/bio.h>
#include <linux/module.h>
#include <linux/slab.h>
#include <linux/init.h>

sector_t currSector = -1;

struct look_data{
        struct list_head queue;
};

static void look_merged_requests(struct request_queue *q, struct request *rq, struct request *next)
{
        list_del_init(&next->queuelist);
        printk(KERN_DEBUG "Request merged.\n");
}

static int look_dispatch(struct request_queue *q, int force)
{
        struct look_data *nd = q->elevator->elevator_data;

        if(!list_empty(&nd->queue)){
                struct request *rq;
                rq = list_entry(nd->queue.next, struct request, queuelist);
                list_del_init(&rq->queuelist);
                elv_dispatch_sort(q, rq);
                currSector = blk_rq_pos(rq);
                return 1;
        }
        printk(KERN_DEBUG "Queue dispatched.\n");
        return 0;
}

static int look_init_queue(struct request_queue *q, struct elevator_type *e)
{
        struct look_data *nd;
        struct elevator_queue *eq;

        eq = elevator_alloc(q, e);
        if(!eq){
                return -ENOMEM;
        }
        nd = kmalloc_node(sizeof(*nd), GFP_KERNEL, q->node);
        if(!nd){
                kobject_put(&eq->kobj);
                return -ENOMEM;
        }
        eq->elevator_data = nd;

        INIT_LIST_HEAD(&nd->queue);

        spin_lock_irq(q->queue_lock);
        q->elevator = eq;
        spin_unlock_irq(q->queue_lock);
        printk(KERN_DEBUG "SSTF INIT.\n");
        return 0;
}

static void look_add_request(struct request_queue *q, struct request *rq)
{

    struct look_data *nd = q->elevator->elevator_data;
        struct list_head *curr = NULL;
        struct list_head *reqOnQueue = NULL;

        if(list_empty(&nd->queue)){
                printk(KERN_DEBUG "EM: The queue is empty.\n");
        }

        list_for_each(curr, &nd->queue){
                struct request *c = list_entry(curr, struct request, queuelist);
                printk(KERN_DEBUG "Now going through the queue.\n");

        if(blk_rq_pos(rq)<currSector){
                        if(blk_rq_pos(rq)>blk_rq_pos(c)){
                                break;
                        }
                        if(blk_rq_pos(c)>currSector){
                                break;
                        }
                }else{
                        if(blk_rq_pos(c)>currSector && blk_rq_pos(rq)>blk_rq_pos(c)){
                                break;
                        }
                }
        }

        if(curr != NULL){
                printk(KERN_DEBUG "Current is not NULL!!");
        }

        list_add_tail(&rq->queuelist, curr);

        printk(KERN_DEBUG "This is current sector: %llu.\n", (unsigned long long)currSector);//orig
        printk(KERN_DEBUG "This is the sector of the new request: %llu.\n", (unsigned long long)blk_
        list_for_each(reqOnQueue, &nd->queue){
                struct request *secNum = list_entry(reqOnQueue, struct request, queuelist);
                printk(KERN_DEBUG "%llu.\n", (unsigned long long)blk_rq_pos(secNum));
        }
        printk(KERN_DEBUG "Request Added.\n");
}

static struct request* look_latter_request(struct request_queue *q, struct request *rq)
{
        struct look_data *nd = q->elevator->elevator_data;

        if(rq->queuelist.next == &nd->queue){
                return NULL;
        }
        return list_entry(rq->queuelist.next, struct request, queuelist);
}

static struct request* look_former_request(struct request_queue *q, struct request *rq)
{
        struct look_data *nd = q->elevator->elevator_data;

        if(rq->queuelist.prev == &nd->queue){
                return NULL;
        }
        return list_entry(rq->queuelist.prev, struct request, queuelist);
}

static void look_exit_queue(struct elevator_queue *e)
{
        struct look_data *nd = e->elevator_data;

        BUG_ON(!list_empty(&nd->queue));
        kfree(nd);
}

static struct elevator_type elevator_look = {
        .ops = {
                .elevator_merge_req_fn                = look_merged_requests,
                .elevator_dispatch_fn                = look_dispatch,
                .elevator_add_req_fn                = look_add_request,
                .elevator_former_req_fn                = look_former_request,
                .elevator_latter_req_fn                = look_latter_request,
                .elevator_init_fn                = look_init_queue,
                .elevator_exit_fn                = look_exit_queue,
        },
        .elevator_name = "look",
        .elevator_owner = THIS_MODULE,
};

static int __init look_init(void)
```

```
{
        return elv_register(&elevator_look);
}

static void __exit look_exit(void)
{
        elv_unregister(&elevator_look);
}

module_init(look_init);
module_exit(look_exit);

MODULE_AUTHOR("Tao Chen");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("C-Look IO Scheduler");
```