CPSC 8220 – PROJECT 3

SSTF (GREEDY) DISK SCHEDULER

Chinmay Joshi

Prantit Lokre

Shreya Deodhar

Chirantan Sharma

SCHEDULER CODE

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

struct greedy_data {
	sector_t head;
	struct list_head upper;
	struct list_head lower;
};

static void greedy_merged_requests(struct request_queue *q, struct request *rq,
				 struct request *next)
{
	list_del_init(&next->queuelist);
}

static int greedy_dispatch(struct request_queue *q, int force)
{
	struct greedy_data *gd = q->elevator->elevator_data;
	struct request *rq, *rq1;
	sector_t ld,hd;

	if(list_empty(&gd->lower) && list_empty(&gd->upper))
	{
		return 0;
	}
	if(list_empty(&gd->lower))
	{
		rq = list_entry(gd->upper.next,struct request,queuelist);

	}
	else if(list_empty(&gd->upper))
	{
		rq = list_entry(gd->lower.next,struct request,queuelist);

	}
	else
	{
		rq = list_entry(gd->lower.next, struct request, queuelist);
		rq1 = list_entry(gd->upper.next, struct request, queuelist);
		ld = blk_rq_pos(rq);
		hd = blk_rq_pos(rq1);
		if(ld < gd->head) ld = gd->head - ld;
		else ld= ld - gd->head;
		if(hd < gd->head) hd = gd->head - hd;
		else hd= hd - gd->head;
```

```c
                if (ld < hd) //lower is near than upper
                {
                        ;
                }
                else   // upper is near than lower
                {
                        rq=rq1;
                }
        }
        list_del_init(&rq->queuelist);
        elv_dispatch_add_tail(q, rq);
        gd->head = rq_end_sector(rq);
        return 1;
}

static void greedy_add_request(struct request_queue *q, struct request *rq)
{
        struct greedy_data *gd = q->elevator->elevator_data;

        struct request *point;
        struct list_head *pos;
        sector_t curr;

        curr = blk_rq_pos(rq);
        //now check to decide which queue to add request
        if(curr < gd->head)//add to lower
        {
                list_for_each(pos,&gd->lower)
                {
                        point = list_entry(pos,struct request, queuelist);
                        if (curr > blk_rq_pos(point))
                                break;
                }
        }
        else
        {
                list_for_each(pos,&gd->upper)
                {
                        point = list_entry(pos,struct request, queuelist);
                        if (curr < blk_rq_pos(point))
                                break;
                }
        }
        __list_add(&rq->queuelist,pos->prev,pos);
}

static struct request *
greedy_former_request(struct request_queue *q, struct request *rq)
{
        struct greedy_data *gd = q->elevator->elevator_data;

        if( (rq->queuelist.prev == &gd->upper) || (rq->queuelist.prev == &gd->lower) )
                return NULL;
        return list_entry(rq->queuelist.prev, struct request, queuelist);
}

static struct request *
greedy_latter_request(struct request_queue *q, struct request *rq)
```

```
{
	struct greedy_data *gd = q->elevator->elevator_data;

	if( (rq->queuelist.next == &gd->upper) || (rq->queuelist.next == &gd->lower) )
		return NULL;
	return list_entry(rq->queuelist.next, struct request, queuelist);
}

static int greedy_init_queue(struct request_queue *q, struct elevator_type *e)
{
	struct greedy_data *gd;
	struct elevator_queue *eq;

	eq = elevator_alloc(q, e);
	if (!eq)
		return -ENOMEM;

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

	INIT_LIST_HEAD(&gd->upper);
	INIT_LIST_HEAD(&gd->lower);
	gd->head = 0;

	spin_lock_irq(q->queue_lock);
	q->elevator = eq;
	spin_unlock_irq(q->queue_lock);
	return 0;
}

static void greedy_exit_queue(struct elevator_queue *e)
{
	struct greedy_data *gd = e->elevator_data;

	BUG_ON(!list_empty(&gd->lower));
	BUG_ON(!list_empty(&gd->upper));
	kfree(gd);
}

static struct elevator_type elevator_greedy = {
	.ops = {
		.elevator_merge_req_fn		= greedy_merged_requests,
		.elevator_dispatch_fn		= greedy_dispatch,
		.elevator_add_req_fn	= greedy_add_request,
		.elevator_former_req_fn		= greedy_former_request,
		.elevator_latter_req_fn		= greedy_latter_request,
		.elevator_init_fn		= greedy_init_queue,
		.elevator_exit_fn		= greedy_exit_queue,
	},
	.elevator_name = "greedy",
	.elevator_owner = THIS_MODULE,
};

static int __init greedy_init(void)
```

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

static void __exit greedy_exit(void)
{
        elv_unregister(&elevator_greedy);
}

module_init(greedy_init);
module_exit(greedy_exit);

MODULE_AUTHOR("BlueRibbon");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greedy IO scheduler");
```

RESULTS

CFQ

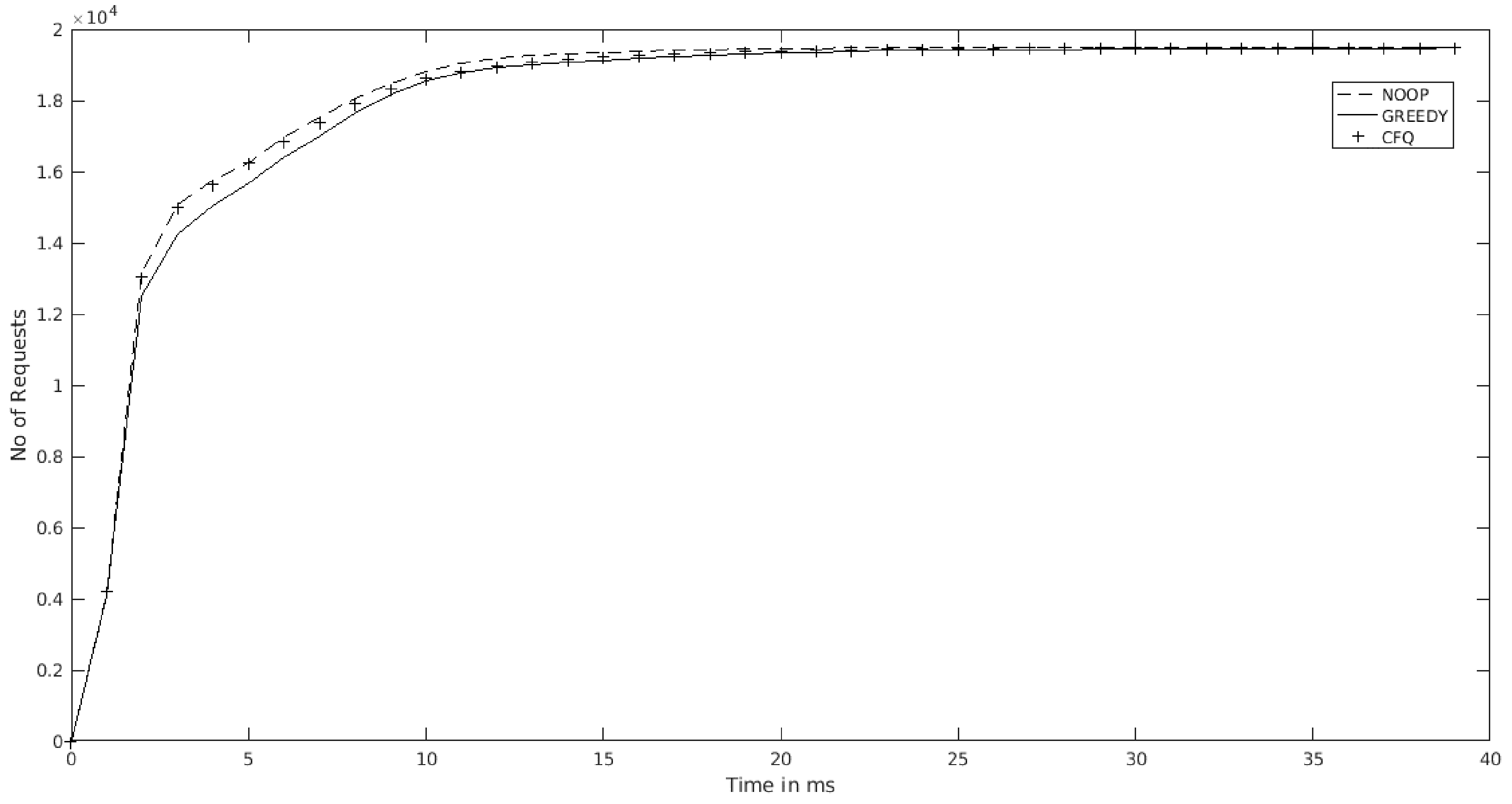
- Mean Service Time: 2.3321 ms
- Mean Response Time: 76.377 ms

NOOP

- Mean Service Time: 2.2515 ms
- Mean Response Time: 11.252 ms

GREEDY

- Mean Service Time: 2.6395 ms
- Mean Response Time: 10.328 ms



×10^4
No of Requests
Time in ms
NOOP
GREEDY
CFQ