```c
/*
 * elevator noop
 */
#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 noop_data {
        struct list_head queue;
};

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

static int noop_dispatch(struct request_queue *q, int force)
{
        struct noop_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);
                return 1;
        }
        return 0;
}

static void noop_add_request(struct request_queue *q, struct request *rq)
{
        struct noop_data *nd = q->elevator->elevator_data;

        list_add_tail(&rq->queuelist, &nd->queue);
}

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

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

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

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

static int noop_init_queue(struct request_queue *q, struct elevator_type *e)
{
        struct noop_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);
        return 0;
}

static void noop_exit_queue(struct elevator_queue *e)
{
        struct noop_data *nd = e->elevator_data;

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

static struct elevator_type elevator_noop = {
        .ops = {
                .elevator_merge_req_fn          = noop_merged_requests,
                .elevator_dispatch_fn           = noop_dispatch,
                .elevator_add_req_fn            = noop_add_request,
                .elevator_former_req_fn         = noop_former_request,
                .elevator_latter_req_fn         = noop_latter_request,
                .elevator_init_fn               = noop_init_queue,
                .elevator_exit_fn               = noop_exit_queue,
        },
        .elevator_name = "noop",
        .elevator_owner = THIS_MODULE,
};

static int __init noop_init(void)
{
        return elv_register(&elevator_noop);
}

static void __exit noop_exit(void)
{
        elv_unregister(&elevator_noop);
}

module_init(noop_init);
module_exit(noop_exit);


MODULE_AUTHOR("Jens Axboe");
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("No-op IO scheduler");
```