DATA STORAGE TECHNOLOGIES & NETWORKS

(CS C446, CS F446 & IS C446)

LECTURE 15– STORAGE

Anticipatory scheduling

- Used for scheduling harddisk input/output.
- It seeks to increase the efficiency of disk utilization by "anticipating" synchronous read operations.
- "Deceptive idleness"
  - is a situation where a process appears to be finished reading from the disk when it is actually processing data in preparation of the next read operation.
  - I/O scheduler may switch to servicing I/O from an unrelated process assuming the I/O of the process is done.
  - Results in performance degradation because of seeking workload
- Anticipatory scheduling overcomes deceptive idleness by pausing for a short time (a few milliseconds) after a read operation in *anticipation* of another close-by read requests.

Anticipatory scheduling

- Anticipatory scheduling yields significant improvements in disk utilization for some workloads. [In some situations the Apache web server may achieve up to 71% more throughput from using anticipatory scheduling]
- The Linux anticipatory scheduler may reduce performance on disks using TCQ (Tagged Command Queuing – allows OS to send multiple read & write requests to hard disks), high performance disks, and hardware RAID arrays.
- An anticipatory scheduler (AS) was the default Linux kernel scheduler between 2.6.0 and 2.6.18, by which time it was replaced by the CFQ scheduler.
- As of kernel version 2.6.33, the Anticipatory scheduler (AS) has been removed from the Linux kernel

Completely Fair Queuing [CFQ]

- Places synchronous requests into per process queues
- Allocates time slice for each of the queues to access the disk
- The length of the time slice and the number of requests a queue is allowed to submit depends on the IO priority of the given process.
- Asynchronous requests for all processes are batched together in fewer queues, one per priority.
- While CFQ does not do explicit anticipatory IO scheduling, it achieves the same effect of having good aggregate throughput for the system as a whole, by allowing a process queue to idle at the end of synchronous IO thereby "anticipating" further close IO from that process.

Implementation in Linux

- The various algorithms employed in the kernel for scheduling and reordering I/O operations are known as *I/O schedulers*
- Traditionally, I/O schedulers are also called *elevators*. They are represented by a collection of functions grouped together in the data structure
- Will be available in elevator.h header file as struct elevator_ops.
- The I/O scheduler is responsible for request reordering and management of request queue

```
struct elevator_ops
{       elevator_merge_fn *elevator_merge_fn;
        elevator_merged_fn *elevator_merged_fn;
        elevator_merge_req_fn *elevator_merge_req_fn;
        elevator_dispatch_fn *elevator_dispatch_fn;
        elevator_add_req_fn *elevator_add_req_fn;
        elevator_activate_req_fn *elevator_activate_req_fn;
        elevator_deactivate_req_fn *elevator_deactivate_req_fn;
        elevator_queue_empty_fn *elevator_queue_empty_fn;
        elevator_completed_req_fn *elevator_completed_req_fn;
        elevator_request_list_fn *elevator_former_req_fn;
        elevator_request_list_fn *elevator_latter_req_fn;
        elevator_set_req_fn *elevator_set_req_fn;
        elevator_put_req_fn *elevator_put_req_fn;
        elevator_may_queue_fn *elevator_may_queue_fn;
        elevator_init_fn *elevator_init_fn;
        elevator_exit_fn *elevator_exit_fn;
};
```

- elevator_merge_fn
  - checks whether a new request can be coalesced with an existing request.
  - It also specifies the position at which a request is inserted in the request queue.
- elevator_merge_req_fn
  - coalesces two requests into a single request
  - elevator_merged_fn is invoked *after* two requests have been merged
    - it performs clean-up work and returns management data of the I/O scheduler that are no longer needed because of the merge to the system

- elevator_dispatch_fn
  - selects which request from a given request queue should be dispatched next.
- elevator_add_req_fn and elevator_remove_req_fn
  - add and remove a request to/from the request queue.
- elevator_queue_empty_fn
  - checks whether the queue contains requests ready for processing.
- elevator_former_req_fn and elevator_latter_req_fn
  - find the predecessor and successor request of a given request; this is useful when performing merging.

- elevator_init_fn and elevator_exit_fn
  - are invoked when a queue is initialized and returned;
  - their effect is the same as that of a constructor or destructor.
- elevator_set_req_fn and elevator_put_req_fn
  - are invoked when a new request is instantiated and returned to memory management
  - (at this point in time the requests are not yet or no longer associated with any queue or have been satisfied).
  - The functions give the I/O scheduler the opportunity to allocate, initialize, and return management data structures.

Each elevator is encapsulated in the data structure that holds further management information for the kernel:

<elevator.h>

```
struct elevator_type
{
        struct list_head list;
        struct elevator_ops ops;
        struct elv_fs_entry *elevator_attrs;
        char elevator_name[ELV_NAME_MAX];
        struct module *elevator_owner;
};
```