Writing Topic 2: I/O and Provided Functionality Spring 2017 CS444

Hannah Solorzano

***Abstract*—This paper discusses the I/O, how its implemented and the forms it comes in. The data structure, algorithms, and cryptology will be analyzed as well as an in depth look at how the Windows, FreeBSD, and Linux operating system utilize I/O.**

I. INTRODUCTION

I/O, defined as the communication process between a computer and the outside world [**?**], is a way of receiving results from the computer. There are different types of devices that possess different speeds that are measured in Bps where a keyboard is about 10 BPS and a compact flash card is around 40 MBps.

II. TYPES OF I/O

A. BLOCK I/O

A fixed-size chunk of data that can be either physical or virtual which receives requests and sends them to the block devices is called a Block I/O.

Block devices get their name from the fact that software runs on units called blocks that represent a block size in the file system.

B. CHARACTER I/O

Character I/O differs from Block devices as they are simplistic in that they have basic interfaces and coding. They resemble streams of data where the data contains small irreplaceable bytes. These devices are also incapable of random access.

III. DATA STRUCTURES

The Block I/O operates on memory blocks called buffers which correlates to an actual chunk of memory that contains a buffer head as a descriptor. The Block I/O struct is able to grab memory in a scatter-gather method which is allows it to get memory that is not a continuous block.

Listing 1. BIO VEC function that takes in page, length, and offset parameters

```
struct bio_vec {
        struct page     *bv_page;
        unsigned int    bv_len;
        unsigned int    bv_offset;
};
```

A. SCHEDULING

The main role of the I/O scheduler is to control the order of the queues that are sent to the block device. A poor scheduler, such as the Linux Elevator, can cause slow reaction times resulting from a poor request dispatch order.

IV. ELEVATOR ALGORITHMS

The algorithms behind the schedulers that are created to reduce latency in servicing each request are referred to as elevators. Though elevators come in different forms, they all perform the same base functions of merging, either from the back or the front, aging of requests, and sorting. Examples of elevators include the Linus Elevator and the Deadline Elevator.

A. LINUS ELEVATOR

The Linus elevator is not a particularly efficient elevator and is mostly depreciated in most operating systems. It implements both front and back end merging with a sorted single queue, and ages its requests in a strict first-in-first-out order.

Unfortunately, this aging process causes great performance issues because if one request in the queue is older than a predetermined time, then every request is stuffed into the back of the queue.

B. DEADLINE ELEVATOR

The Deadline elevator implements sorting and merging in the same way that the Linus elevator does, but it's method of aging the requests is significantly better. Rather than having one queue, the Deadline elevator utilizes three queues: a sorted queue, a read queue, and a write queue.

The requests are stored in a first-in-first-out method but each request also possesses a timeout. If the head of a queue has an expired timeout, the requests from that queue are processed.

V. I/O SPECIFICITIES

A. DEVICE DRIVERS

The kernel uses device drivers for the 'abstract machine' interface that the users are presented with. Device drivers are capable of working in different ways such as using Direct Memory Access, Processor I/O, and Memory Mapped I/O. Each of these methods utilize memory and registers differently with some being more efficient than others. Processor I/O is one of the least effective as it lets the CPU handle every aspect of I/O which slows processor speed significantly.

VI. OPERATING SYSTEM COMPARISON

Each operating system implements a different kind of elevator to suit their specific needs.

A. WINDOWS

Windows has the Synchronous I/O as it's default, which makes the thread wait for the CPU to finish its operations, only then will the thread receive its status code and continues it's job [**?**].

B. FREEBSD

The I/O featured in FreeBSD possesses a simple algorithm for scheduling called ULE. This scheduler is the successor to the BSD scheduler and is the default for FreeBSD [**?**]. It's key feature is it's ability to support the execution of multiple threads at once by using several queues.

Threads awaiting execution have a priority level given based on whether or not they are time sharing threads, the others merely receive a base priority level. This allows the interactive threads to run first when located in the same queue as a non-interactive thread.

C. LINUX

The Linux I/O scheduler implements the Completely Fair Scheduler(CFS) to preserve a balance of threads and the time slices alloted to them. All threads have the same priority levels and their executable time is determined by the virtual runtime assigned to them [**?**].

A unique feature of the CFS is that it maintains a red black tree rather than a queue. These trees are ordered by time, are self balancing, and allows for the easy addition or deletion of a node.

D. DIFFERENCES IN SCHEDULERS

Each operating system can prefer to use a specific type of I/O scheduler as different ones are specialized. For example, the ULE use by FreeBSD specialized in quick responses and as a result, has a special method of determining which thread runs first. Other operating systems like Linux don't concentrate as much on which is why they use a stricter first-in-first-out thread execution.

E. SIMILARITIES IN SCHEDULERS

Though schedulers perform the same job, their methods of executing threads are quite different. The biggest similarity between these I/O schedulers is that they have queues that maintain the threads and the order in which they are executed. They also implement merging of some sort whether is front, back, or both.

VII. CONCLUSION

In total, I/O comes in many forms with different capabilities and uses. While the schedulers seem to be performing the same job, they use different methods of handling threads and queues to achieve optimal performance while simultaneously reducing latency.