CS 444 GROUP 15

Writing assignment 2

Writer:

Changxu YAN

November 30, 2016

BLOCK I/O

Overview: Block I/O devices are used by random access with same size blocks of data. Each small unit on a block device is called sector. All device I/O will be done in units of sectors, which are then allocate to blocks in memory [1](pg. 289-290).

The sub-system performes these I/O operations is the block I/O scheduler. Block devices use request queues for keep their pending I/O requests [1](pg. 297). The I/O scheduler functions by managing this request queue, deciding the order of requests in the queue and when to dispatch each request to the block device. Seek time is the amount of time required to position the hard disk's head at the location of a specific block. One of the primary optimizations that can be made to block I/O performance is minimizing seek time. I/O schedulers sometimes do this by merging and sorting the requests in the queue. The entire request queue is kept sorted sectorwise so that all seeking along the queue moves sequentially [1](pg. 298-299).

Provided Functionality

Algorithms/Scheduler

There are several types of I/O schedulers built into the Linux kernel we worked with already and these schedulers are optimized for different purposes. The NOOP elevator is simple. It does not perform any data ordering. NOOP merges requests into queue. The Deadline I/O Scheduler is designed to prevent starvation of certain requests (such as write operations starving reads) [1](pg. 300). The Shortest Seek Time First sheduler is pretty self explainning. Lower sector numbers mean that the cylinder is closer to the spindle, while higher numbers indicate the cylinder is further away. The sstf algorithm check which request is closest to the current request position of the head, and then maniplate that request next[2].

In the version of the Linux kernel we were working with, the default scheduler was the Completely Fair Scheduler (CFS). The Completely Fair Scheduler (CFS) is a process scheduler. It was introduced into the 2.6.23 release of the Linux kernel. It handles CPU resource allocation for all running processes. It goal at maximizing overall CPU utilization and maximizing user interactive response performance. The idea behind CFS is to model scheduling as if the system had an ideal processor that could multitask perfectly. In this kind of system, each runnable process n would receive 1/n of the CPU time. CFS run processes round-robin style for very small tasks (in quantities called slices of time), so that in a given period of time it will appear as though many processes are running in parallel [1](pg. 48-50).

CFS is not the only type of scheduling available in the Linux kernel. The Linux scheduler provides soft real-time behavior, it tries to schedule processes within timing deadlines. Linux provides two real-time scheduling policies, FIFO and round-robin [1](pg. 64). sentially queues containing process descriptors for runnable processes. Round-robin scheduling cycles through processes, running each for a pre-specified amount of time known as a timeslice. On a more technical level, this means placing the process descriptor at the end of the runqueue after its timeslice, then running all subsequent tasks until it reaches the end of the queue, at which time it will start the cycle again. FIFO (First In, First Out) scheduling runs each FIFO process of the same priority until it is finished, then runs the next process in the runqueue. It does this in order of when processes were placed in the runqueue. On a technical level, FIFO scheduling behaves almost identically to round-robin scheduling but with infinite timeslices.

Cryptography

CHARACTER I/O

Overview: A key requirement of any realtime operating system is high-performance character I/O. Character devices can be described as devices to which I/O consists of a sequence of bytes transferred serially, as opposed to block-oriented devices (e.g. disk drives). Character devices are accessed one byte at a time.

Linux

As in the POSIX and UNIX tradition, these character devices are located in the OS pathname space under the /dev directory. For example, a serial port to which a modem or terminal could be connected might appear in the system as:

/dev/ser1

Typical character devices found on PC hardware include:

serial ports

parallel ports

text-mode consoles

pseudo terminals (ptys)

Windows

The Windows operating system has it's own system for I/O and utilizes it's own I/O manager. The I/O manager is the core of the I/O system on Windows because it defines the orderly framework within which I/O requests are delivered to device drivers. [3] The I/O system on Windows is packet driven and is represented by an I/O request packet called an IRP. These IRPs are designed to travel between one I/O system to another, allowing an individual application thread to manage multiple I/O request concurrently. An IRP is created in memory by the I/O manager representing an I/O operation. A pointer to the IRP is then passed to the correct driver until the completion of the I/O operation. Once the I/O operation has completed or needs to be passed on to another driver, the IRP is sent back to the I/O manager to either dispose of, or pass on to another driver. In addition to handling IRPs, the I/O manager provides flexible I/O services that allow environment subsystems, such as Windows and Posix to implement their respective I/O functions. The I/O manager has no knowledge of anything except for files, and passes the responsibility to translate file-oriented comments such as open, close and read to the driver. A typical I/O request starts with an application executing an I/O related function that is processed by the I/O manager, one or more device drivers, and the hardware abstraction layer (HAL). On Windows, threads perform I/O on virtual files which refer to any source or destination for I/O that is treated as if it were a file. This can be anything from files, directories pipes and mailslots.

FreeBSD

FreeBSD is a Unix-like operating system that descended from Research Unix via the Berkeley Software Distribution. Since FreeBSD is an operating system that is Unix-like, it contains a lot of features that are derived from UNIX. The basic model of the Unix I/O system is a sequence of bytes that can be accessed either randomly or sequentially, and there are no access methods and no control blocks in a typical Unix user process. FreeBSD categorizes hardware devices by being either structured or unstructured. Structured devices, also known as block devices, are typically disks, magnetic tapes and include most random access devices. On block oriented devices, the FreeBSD kernel supports read-modify-write-type buffering actions to allow reading and writting in a totally random byte-addressed fashion. On FreeBSD filesystems are created on block devices. Unstructured devices, also known as character devices, are devices which do not support a block structure. These can include communication lines, raster plotters and unbuffered magnetic tapes and disks. Character devices do typically support large block I/O transfers. The BSD kernel introduced an IPC mechanism which is more flexible than pipes and is based on sockets called the Socket IPC. A socket is an endpoint of communication referred to by a descriptor. Two processes can each create a socket, and then connect those two endpoints to produce a reliable byte stream. This new Socket IPC mechanism does essentially what was available previously with the use of pipes without the need for a common parent process to setup the communication channel. The transparency of sockets allows the kernel to redirect the output of one process to the input of another regardless if they have a common parent process. The socket IPC mechanism can also create a connection between sockets on two unrelated process and it can even connect them on different machines. This mechanism widely opens up the functionality of communication between processes however does require extensions to the traditional Unix I/O system calls. Previously, the read and write system calls were used for byte-stream type connections, but six new calls were added to allow sending and receiving addressed messages. The system calls for writing include send, sendto and sendmsg, and the system calls for reading messages include recv, recvfrom, and recvmesg. The FreeBSD kernel used to rely on the traditional read and write system calls, however 4.2BSD introduced the ability to do scatter/gather I/O. [4] Scatter/gather I/O is where a single procedure call sequentially reads data from multiple buffers and writes it to a single data stream, or reads data from a data stream and writes it to multiple buffers. Scatter input uses the readv system call to allow a single read to be placed in several different buffers while the writev system call allows several buffers to be written in a single atomic write. Instead of the traditional read and write which passes a single buffer and length parameter, the scatter/gather process passes in a pointer to an array of buffers and lengths, along with a count describing the size of the array.

References

[1] R. Love, *Linux Kernel Development*, 3rd ed. Pearson Education, Inc., 2010.

[2] I. I. of Technology, "Disk scheduling algorithms," http://www.cs.iit.edu/~cs561/cs450/disksched/disksched.html.

[3] M. Russinovich, *Windows Internals*, 6th ed. Microsoft Press, 2012.

[4] freebsd.org, "A look inside..." https://www.freebsd.org/doc/en_US.ISO8859-1/articles/linux-emulation/inside.html.