CMU 18–746 **Storage Systems**
Spring 2010 **Solutions to Homework 0**

Problem 1 : Locality of reference. [10 points]

Spatial locality is the concept that objects logically "close" in location to an object in the cache are more likely to be accessed than objects logically "far." For example, if disk block #3 is read and placed in the cache, it is more likely that disk block #4 will be referenced soon than block #31,415,926 because blocks #3 and #4 are close to each other on the disk.

Temporal locality is the concept that items more recently accessed in the cache are more likely to be accessed than older, "stale" blocks. For example, if disk blocks are read and placed in the cache in the order #271, #828, #182, then it is more likely that disk block #182 will be referenced again soon than block #271. Another way of thinking of this is that a block is likely to be accessed multiple times in a short span of time.

Problem 2 : Virtual memory. [20 points]

(a) *Swapping* is the process of evicting a virtual memory page from physical memory (writing the contents of the page to a backing store [for example, a disk]) and replacing the freed memory with another page (either a newly created page, or an existing page read from the backing store). In essence, one virtual memory page is "swapped" with another. Swapping is initiated whenever all physical page frames are allocated and a virtual memory reference is made to a page that's not in memory. The granularity of swapping is a single page frame; it need not be an entire process that gets swapped out.

(b) A *page fault* occurs when a virtual memory reference is made to a virtual memory page that is not currently in memory. This may or may not cause swapping: if there are free physical page frames, and the virtual memory page is "new" (it was not previously created and written to the backing store) then swapping is not necessary. These are the steps involved in handling a page fault:

- An exception is raised by the memory management unit
- The faulting address is checked for validity (if invalid, the process receives a segmentation violation signal)
- A clean physical page frame is allocated to make space for the new virtual memory page (this may require swapping)
- If the virtual memory page was previously written to disk, a disk I/O is initiated to read the contents into the physical page frame
- The new physical page frame is mapped into the process' address space
- The user process is re-started at the faulting instruction

Problem 3 : Polling and interrupts. [15 points]

Polling and interrupt-driven notification are methods that the operating systems uses to notify a processor when an I/O device requires attention; e.g., when requested data are available or an operation completes. In *polling* the processor reads the device's status register ("polls") and checks whether a "finished" flag has been set by the device. If the flag is set, the processor initiates a data transfer between the device and main memory. If the flag is not set, the processor either tries again later or busy-waits (continuously reading the register until the flag is set).

In *interrupt-driven notification* the device asserts a special I/O line that triggers an "interrupt" on the processor. This causes the processor to stop what it's doing and jump to a special operating-system routine that determines which device caused the interrupt and what needs to be done next (e.g., initiating a data transfer between the device and main memory).

Two advantages of polling are (1) its simpler hardware requirements and software implementation, and (2) greater efficiency when the I/O device generates a lot of traffic. Two advantages of interrupt-driven notification are (1) the processor does not need to expend cycles polling the device, and (2) there can be less latency between when the device is ready and the data are transferred.

Problem 4 : Application I/O. [10 points]

(a) Application run time can usually be improved both by improving the processor (faster clock speed, greater IPC, etc.) and improving the I/O subsystem (reducing bus load, mirroring disks, etc.) However, a process is most "bound" by the resource that is the major factor preventing that process from completing faster. An application that spends over half its time stalled waiting for I/O requests to complete (an *I/O-bound process*) would be helped more by improving I/O latency and/or bandwidth than it would be by overclocking the processor (think of this in terms of Amdahl's Law). The reverse analogy holds for *CPU-bound processes*.

(b) If an application is unable to continue execution until an outstanding I/O request completes, then it is a *blocking I/O request*. This can be either necessary (a scientific application needs more data over which to calculate) or application-directed (an event-logging application that requests confirmation that data have been written to stable storage before permitting further events to occur). *Nonblocking I/O* allows the application to continue execution regardless of the availability of the data or the I/O device. This can be by design (an application-directed cache preloading request) or by circumstance (a "select" system call when no clients have sent data to the network interface).

Problem 5 : Programmed I/O and DMA. [15 points]

(a) These are methods the OS programmer can use to control the actual transfer of data between system memory and I/O devices.

In *Programmed I/O (PIO)* the processor reads or writes one byte of data at a time through the data register on the I/O device controller. Special flags in the control register are used to coordinate some communication between the processor and I/O device controller ("new data are available in the data register," "I have finished transferring the data," etc.) The processor may use either polling or interrupt-driven I/O to learn when the I/O device has finished a task. PIO may be inefficient when transferring large amounts of data to or from a storage device, because the processor spends a large amount of time dealing with one-byte reads and writes between the data register and main memory.

In *Direct Memory Access (DMA)*, the processor offloads the entire transfer to the DMA controller (a special-purpose processor attached to the system bus). The processor writes the source and destination memory locations and length to the DMA controller once, then the DMA controller directs the transfer of data between the I/O controller and memory while the processor does other work. The processor is eventually notified via interrupt when the DMA transfer completes.

(b) Devices that transfer "large" amounts of data typically use DMA. For example, all PCI-bus-based I/O controllers that support PCI bus mastering use DMA. This includes SCSI host adapters, some ATA disk controllers, PCI-based network cards, video cards, etc. PIO is used for devices that transfer small amounts of data infrequently, such as keyboards and mice.

Problem 6 : Communications networks. [20 points]

(a) Both terms have slightly loose definitions. *Latency* is the amount of time you wait to get the data your requested—so it is the time between when you click on a web page link and either (1) that web page appears in its entirity, or (2) you first start receiving data for that web page. The unit of latency is usually seconds.

Bandwidth is the rate at which your get the data you requested—so it is the size of the web page divided by either (1) the total time between when you clicked on the link and you received the last byte of data, or (2) the total time between when you received the first and last bytes of data. The unit of latency is usually bytes (or kB, mB, bits, etc.) per second.

(b) A *Hamming code* is a mechanism that enables error detection (for up to N bit errors) and error correction (for fewer than N bit errors) on an encoded string of bits. It works by calculating parity information across different groups of bits inside the string—when errors occurs during transmission, the location of errors can be pinpointed by recalculating the "correct" parity values and comparing them with those that were received.

Parity-based error *detection* only detects than an error occured during transmission (and then only for an odd number of bit errors). The use of Hamming codes allows for error *correction*, and for the detection of either an even or odd number of bit errors (up to a limit).