

Systems Comprehensive Exam Solutions, Fall 2003

Patrick G. Bridges

Tuesday, August 19, 2003

1 Short Answer - Answer 3 of 4 (30 points)

each question should be no longer than *one* paragraph.

1. Consider a machine with a standard 5-stage pipeline (perhaps the DLX pipeline described by Hennessy and Patterson—Fetch, Decode, Execute, Memory, and Writeback). Suppose you split every stage into 2 stages which took half as long to run and as a result doubled the clock speed of the processor. Explain why programs would not necessarily run twice as fast on this processor despite the doubled clock frequency.

Answer: Because of pipeline hazards between stages, more pipeline stalls and bubbles would probably occur. This would increase the average CPI of most instructions, preventing the performance of the machine from doubling.

2. Consider the inner-most backward branch in a set of nested loops, where the program exits the loop when that branch is not taken. Why will a two-bit branch predictor generally perform better in this case than a one-bit predictor?

Answer: Assumption: The inner branch is taken when the loop continues and not taken when the loop finishes a set of iterations. A one-bit predictor will always mispredict the inner backwards branch on the first iteration as not taken because the branch was not taken when the previous incarnation of the loop finished. In contrast, a two-bit saturating predictor requires two successive not-takens to flip the predictor; If the inner loop generally processes more than two iterations, a two-bit predictor will correctly predict the inner branch as taken the first time it is reached.

3. In the context of RPC, what is marshalling and how is it done?

Answer: Marshalling is the process of flattening RPC arguments into a buffer for transmission over the network; it may also convert the arguments to a processor-independent representation to avoid problems such as processor byte order differences. Marshalling is generally done by copying integral datatypes into the buffer and walking linked data structures in a predetermined way to copy them into the buffer prior to transmission.

4. In a transport protocol, what is the difference between congestion control and flow control?

Answer: In congestion control, the sender limits the rate at which it sends data based on information from the network on its available capacity. In flow control, the sender throttles data transmission based on feedback from the receiver to avoid overrunning its capacity.

2 Medium Answer - Answer 2 of 3 (20 points)

2.1 Priority Inversion

Priority inversion is an important problem in scheduling and synchronization. Define what priority inversion is, and carefully illustrate how it could occur in a specific situation. How is priority inversion usually prevented?

Answer: Priority inversion is when a low-priority process prevents a higher-priority process from running² in a strict priority scheduling system even though the low priority process does not explicitly share resources (e.g. locks) with the higher-priority process.

As an example, consider three processes A , B , and C , where $Prio(A) > Prio(B) > Prio(C)$ and one lock L shared between processes A and C . If a situation can occur where process C prevents process B from running, process C prevents process A from running *without holding the lock*, or process B prevents process A from running, then priority inversion has occurred.

Now consider the following sequence of events.

1. Processes C become runnable and processes A and B do not yet exist.
2. Process C acquires lock L .
3. Process A becomes runnable, preempts process C (according to strict priority scheduling) and runs until it attempts to acquire lock L .
4. Process C is scheduled because process A is no longer runnable and continues to execute while holding the lock.
5. Process B becomes runnable and preempts process C (according to strict priority scheduling).

At this point, process B is running and indirectly preventing process A from ever running even though the two processes do not share a lock. This is priority inversion.

Priority inversion is generally solved using *priority inheritance locks*. In priority inheritance locks, the process that holds a lock is temporarily given the priority of the highest priority job waiting for a lock. In the case above, process C would be given the same priority as process A while A was waiting to acquire the lock. This would prevent B from running until C released the lock, at which point A would be runnable and would acquire the lock and continue executing without interference from process C .

2.2 Consistency

A variety of different consistency models have been defined for use in distributed systems, the most important of which are *sequential consistency* and various forms of *weak consistency*.

- (a) Define sequential consistency. Pick and describe one specific weak consistency model.
- (b) Construct and show an example in which sequential consistency and your chosen form of weak consistency will return different results.

Answer: In sequential consistency, operations (reads and writes) from multiple machines appear to happen in the same sequential order on every machine, and that order obeys program order. In the various weak consistency models, synchronization variables on whom operations are kept relatively strongly consistent, operations on non-synchronization variables can happen in a variety of orders in relation, but operations on non-synchronization variables happen in a strongly-consistent (sequentially, causally, or FIFO consistent) order in relation to operations on synchronization variables. In other words, operations on synchronization variables impose a partial order on operations on non-synchronization variables.

Canonical weak consistency requires that **sync** operations between processors happen in a sequentially consistent order, that every outstanding write must complete on every processor (in arbitrary order) before a **sync** completes, and that no read or write is allowed to be performed on any processor until all outstanding **sync** operations have completed.

Consider the following case with two processors, where the variable A starts with the initial value of -1 .

Processor 1	Processor 2
Write(A , 0)	$X = \text{Read}(A)$
Write(A , 1)	$Y = \text{Read}(A)$

In sequential consistency, Processor 2 cannot end up with $X = 1$ and $Y = 0$ because it must obey program order. With weak consistency and no synchronization variables in place, however, $X = 1$ and $Y = 0$ is a valid weakly consistent order, because only accesses to synchronization variables and writes explicitly ordered by these accesses are specifically ordered.

2.3 Caching in Distributed File Systems

Different distributed file systems cache data at different granularities. For example, the Andrew File System (AFS) caches whole files, while the Sprite file system and most versions of NFS cache individual file blocks.

(a) What are the advantages and disadvantages of each approach?

(b) How does the choice of a caching mechanism relate to options for maintaining consistency and coherency in the file system?

(c) How suitable is of each approach to dealing with disconnected operation (i.e., the access of cached data when disconnected from the network)?

Answer: *Per-file caching* caches whole files on hosts both when the is opened and potentially longer-term. *Per-block caching*, on the other hand, retrieves and caches file information on a block-by-block basis. This means that per-file caching is potentially more aggressive than per-block caching since it caches larger amounts of data for large files. If large amounts of local space and sufficient network bandwidth are available for initial loading of the cache, per-file caching will achieve higher hit-rates. If the amount of memory or local disk available for caching is low, on the other hand, or the number of files expected to be simultaneously open is very large, per-block caching is less resource-intensive.

In terms of consistency and coherency, per-file caching guarantees that information read from a file is consistent across the cached version of the file; the file data will not change out from under the reader. This is not necessarily the case with per-block caching, where reads to successive blocks of a file may return different versions of the block of the file unless extra measures are taken to ensure consistency.

In terms of dealing with disconnected operation, per-file caching is superior as the entire contents of cached files are available. In a per-block caching scheme, only portions of cached files are typically available. It may not be wise to successfully open a file when only a fraction of its contents are available at the time.

3 Design - Answer 1 of 2 (25 points)

3.1 System I/O Hardware and Software

The company you work for is designing a new computer system design and heavily customizing the operating system to take advantage of this system. Two types of non-volatile media appropriate for secondary storage are being considered for use in the system's file system: a high-latency, high-bandwidth medium (e.g. modern disk drives) and a low-latency, low bandwidth medium (similar to e.g., compact flash memory). Both formats have the same price/megabyte. How would you customize the operating system to best take advantage of these types of storage, and approximately how much each type of memory would you suggest be included in low-end and high-end versions system?

Answer: The most obvious way to take advantage of this storage is to partition data based on usage pattern to the different media types, with large numbers of small, frequently accessed items on the low bandwidth medium and large, less-frequently accessed items on the high bandwidth medium. This partitioning could be conducted either statically based on file time or user preference, or dynamically based on information about file size and usage.

In standard file systems, the easiest way to partition data would be to put metadata (which is small and frequently accessed) on the low latency storage and normal file blocks on the high-bandwidth medium. In addition, simple file assignment policies could be used to, for example, default all created files in a given directory to default to one storage or another. Migration between storage media could also be done using high and low water mark sizes for files, with files over certain sizes (perhaps for a certain length of time) automatically moved between storage devices by an overnight maintenance task.

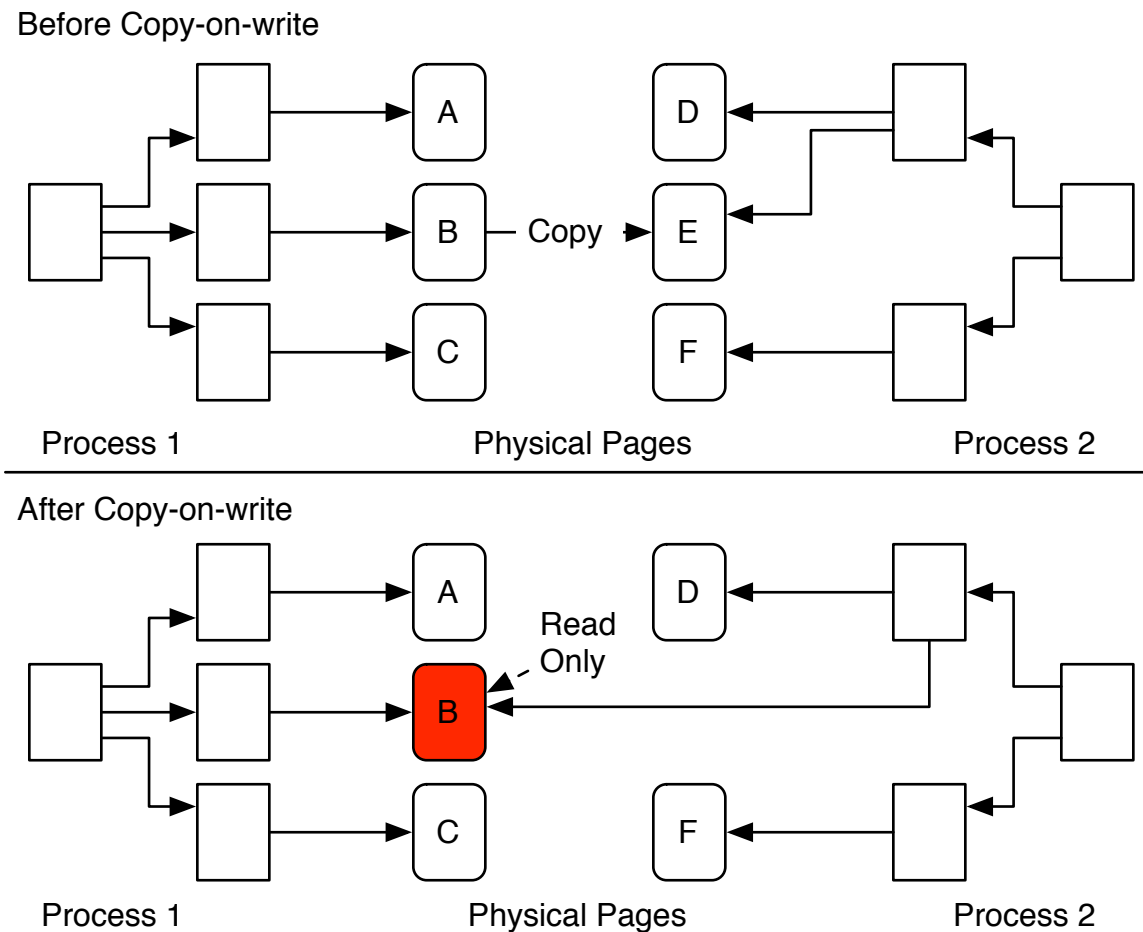
Reliability is going to be one possible concern with this approach; by splitting a file system across two different devices, the chances of the file system failing are increased because the failure of either device would corrupt large portions of the entire file system. In high-end systems, it might be desirable to mirror the file system between the two devices so that the failure of one could be tolerated and the performance advantages of both types of storage optimized. Maintaining consistency between the two devices on writes would then be the primary concern, but this could be dealt with using either logging or a small amount of nonvolatile RAM to store data until it is committed to both devices.

3.2 Stupid Paging Tricks

Copy-on-write is frequently used to optimize performance of data passing and sharing between multiple processes. (Recall that in a copy-on-write system, “copied” data is actually mapped read-only into multiple processes, and an actual copy is made for a process only when it attempts to write to the data.)

(a) Assuming a two-level page table, show the actions necessary for providing copy-on-write sharing for one page of data between two processes.

Answer:



(b) In some cases, immediately copying the data may be quicker than remapping the data. Which factors should you consider in determining which approach is better, and how do each of these factors effect the performance of each approach?

Answer: If a copy-on-write page is eventually written to, then the page remapping costs associated with performing copy-on-write are unnecessary because the page would be copied anyway. To determine whether copy-on-write is appropriate, a system needs to know F_W , the frequency of copied pages that are written to ($F_R = 1 - F_W$ is the percentage of pages *not* copied), C_{COW} , the cost of doing copy-on-write page table manipulations, and C_{COPY} , the cost of doing the copy. The overall cost of a copy is then $C_{COW} + C_{COPY} * F_W$ and copy on write is on average cheaper than a real copy when:

$$\begin{aligned}
 C_{COW} + C_{COPY} * F_W &< C_{COPY} \\
 C_{COW} &< C_{COPY} * (1 - F_W) \\
 C_{COW}/C_{COPY} &< (1 - F_W) \\
 C_{COW}/C_{COPY} &< (1 - F_W) \\
 C_{COW}/C_{COPY} &< F_R
 \end{aligned}$$

When the of copy-on-write to copy cost ratio is greater than the percentage of pages that COW is useful on, we should just copy instead.