

# Systems Comprehensive Exam Solutions, Spring 2005

January 11, 2005

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

*Briefly* answer 3 of the following 4 questions. Your answer should be no longer than *one* paragraph.

1. **Caching.** Write-through caches are less complex and less expensive to implement than write-back caches because they require less state in general and because, in multi-processor systems, they need only snoop the memory bus to detect cache updates. There is no need to use more complex directory or ownership cache coherence protocols. On the other hand, write-through caches use significantly more bus bandwidth than write-back caches, and so generally do not scale beyond a handful of processors.
2. **File Systems.** Log-structured file systems are designed to optimize writes and recovery after crashes instead of reads, with the assumption being that the buffer cache will handle most reads. Because writes are bundled together at the end of the log based on when they happened, as opposed to across the disk based on the directory the associated file is in, log-structured file systems work best on workloads with temporal locality, while traditional UNIX file systems work best on workloads with logical locality.
3. **Consistency.** Weakened consistency models associate all consistency with synchronization operations such as barriers and entry or release of critical sections, as opposed to with reads and writes.
4. **TCP/IP.** This state is used to make sure that there aren't packets floating around in the network for a socket that has been closed. The process that closes the socket first enters the TIME WAIT state, historically this was expected to be the client. Modern Web servers tend to close connections very quickly (as soon as the request has been handled) and are likely to close the connection before the client. This means that Web servers will have lots of sockets in the TIME WAIT state which may lead to resource exhaustion.

## 2 Medium Answer - Answer 2 of 3 (40 points)

Provide detailed answers to two of the following three questions. Be sure to state any assumptions you make and to fully justify your answers. Limit your answers to approximately one to two pages in length.

1. **Addressing Modes. Basic tradeoff:** Page tables increase processor and OS complexity for managing and switching between processes; reducing this cost with other translation or protection mechanisms reduces processor cost and improves the predictability of instruction execution times, specifically in memory accesses. Such mechanisms are generally less flexible, however, can require application programmer involvement, increasing development time and program complexity, and may only work if there are a limited number of programs running.

**Details** Costs of page translation include the TLB hardware and either page table hardware or OS complexity for software-managed TLBs. Also problematic is the uncertainty of memory access times associated with TLB misses. Embedded systems, and particularly real-time systems, cannot afford this cost and its unpredictable nature.

In base and bound addressing, for example, a base register is used to specify the bottom of section of the application's physical address space and the application addresses using offsets from this base register. This only requires that the operating system verify the base address when it is set and a comparison with the bound of the address space on each access, which is a simple and predictable operation. Such addressing does not easily support sparse address spaces or large numbers of applications without complex programmer involvement, making it generally unsuitable for more general systems.

2. **Locking. Spin-locks** work well with short waiting times because they immediately detect the release of the lock. For long waits, however, spin-locks waste large amounts of processor time that could be used to run other processes.

**Blocking locks** work well with long waiting times because they release the processor immediately for other processes to run. However, the overhead associated with queueing and dequeuing processes from a waiting list and rescheduling is much greater than the simple test-and-set primitives used by spin-locks, so for short waits, the overhead of blocking locks is undesirable.

**A hybrid approach** that assumes a bi-model distribution of waiting times would start by spin-locking for a certain amount of time prior to blocking. In this way, short waits would be satisfied with very low overhead, while long waits would not waste too much CPU. The amount of time to spin prior to waiting would have to be determined by examining the distribution of waiting times - too long of a spin would waste inordinate amounts of CPU in cases where the process should block, while too short of a spin would incur the overhead of blocking when a marginally longer spin would have acquired the lock.

3. **Operating System Structure.** Compare and contrast the following general approaches to OS design: monolithic systems, monolithic systems with loadable modules, microkernels and exokernels (library OS).

**Monolithic systems** are generally large and difficult to reason about, but are usually more efficient and easier to develop because artificial protection boundaries do not impede design or implementation, or limit optimization opportunities.

**Support for modules** makes it easier to manage the runtime complexity of a monolithic system, but requires that interfaces within the kernel be designed more carefully.

**Microkernels** move all but the required trusted computing base out of the OS and into servers. This makes it easier to reason about the code that constitutes the kernel as so is more appropriate in systems where reliability and security are the overriding concern. Moving device management out of the kernel is particularly difficult as it requires mechanisms to allow servers to access and control devices- the efficient delivery of interrupts from devices is particularly difficult. Authentication and authorization may be repeated by multiple servers on a single request, protection boundary crossing between servers have inherent overheads, and the hard division of services into separate servers can limit optimization opportunities. As a result, they are not as easily efficient as monolithic systems.

**Exokernels and "library operating systems"** are extreme versions of microkernels that move as much functionality as possible into libraries linked with applications instead of into servers. This can reenables some of the optimization opportunities lost by pure microkernel systems. However, shared resources (e.g. the TCP port space) must still generally reside in servers, imposing some additional overheads compares to monolithic systems.

### 3 Design - Answer 1 of 2 (30 points)

#### 3.1 Multiprocessor Scheduling

Consider the scheduler for an operating system for a shared memory multi-processor. In such systems, the scheduler functionality can be distributed in a variety of ways. Describe in detail the advantages and disadvantages of:

- A single global scheduler
- Individual per-processor schedulers that interact with a global scheduler that assigns jobs to processors.
- A hierarchy of interacting processor schedulers.

In writing your answer, be sure to consider:

- The size of the machine (which can range from two processors up to several hundred processors)
- Application characteristics (e.g. job lengths, job-level parallelism, and synchronization behavior)
- Operating system parameters (e.g. length of scheduling quantum)

The key issues to address in this problem, roughly in order of importance, are:

**Scheduler Scalability.** How well does the scheduling strategy scale as the number of processors in the system increases, and what causes potential scalability problems in each approach? Centralized approaches with a global scheduling queue will occur lock contention/synchronization problems as the number of processors in the system increases. Hierarchical and multi-level approaches ameliorate this by calling the upper-level (or global) schedulers less often (when a load imbalance between processors is detected, a new processor becomes available, or a new process is created).

**Fairness.** Does the scheduler strategy divide CPU time fairly between multiple competing processes? Global scheduling strategies are generally fair; because they have a global view of the state of the system, they can guarantee that each process gets a fair share of the processor. Distributed approaches based on per-processor schedulers, however, may become unfair due to load imbalance. If too many jobs are assigned to a local processor, jobs on that processor may get less processor time than other jobs at the same priority on less-loaded processors.

**Workload dependencies** How does the viability of this scheduler strategy, for example in terms of fairness, depend on application workload? Fairness in distributed schedulers will primarily be an issue when large numbers of processes are being created, destroyed, or blocking, because this will increase the demands on the scheduler and opportunities for unfairness. This will also, however, increase potential lock contention problems on global scheduling strategies. Fairness versus scalability is still the key tradeoff in scalable scheduling.

**Scheduler Parameters** How does changing various scheduler parameters, for example scheduler quantum, effect the efficiency and fairness of each scheduler? Longer scheduler quanta will decrease the frequency at which the scheduler is called, allowing global strategies with potential lock contention issues to scale somewhat more. However, this also increases the time between reschedules, potentially decreasing fairness and interactive response. Discussing an adaptive strategy to setting the scheduler quantum here might be an interesting addition worth extra consideration.

**Application Performance.** How well will the scheduler support optimizations that improve the performance of multi-threaded applications? Global schedulers make implementation of scheduling optimizations such as gang scheduling easy; such scheduling optimizations are very difficult to implement in distributed schedulers without some form of global coordination. Addressing this issue, which is not specifically called out in the problem, is worth extra consideration.

Satisfactory answers should address at least the first two issues and at least one of the others.

## 3.2 Intelligent Networks

Suppose that you have a small router (10-12 port pairs) that is capable of actively processing every packet that comes through the router. That is, the router can execute on the order of 1000 instructions per packet (in addition to routing the packet). In addition, the router has a small amount of memory (on the order of a couple of megabytes) that can be used to store state information. As an example, the router could examine an incoming packet and, based on a value in the packet, decide that the original packet should be dropped, a counter should be incremented and that several new packets should be constructed and sent on different output ports.

How might you use such a router in the implementation of a parallel file system where one of the port pairs will be used to connect the parallel file system to the outside world and the remaining port pairs would be used to connect systems that provide the processing and storage space needed for the file system.

Pay particular attention to how you would partition functionality between the switch and the systems connected to the switch.