Instructions: Answer all six questions. **Computer Sciences @ Wisconsin – OS Qualifier: Spring 2011**

1. Distributed Systems

Google's GFS and Amazon's Dynamo systems both provide storage for applications running in a data center. However, their designers made different choices regarding the tradeoffs between availability and consistency.

(a) What is the availability and consistency guarantee offered by Dynamo, and why? For example, under what circumstances will Dynamo not be available or not offer consistent results?

(b) What is the availability and consistency guarantee offered by GFS, and why? For example, under what circumstances will GFS not be available or not offer consistent results?

(c) The end-to-end argument asks the ultimate consumer of data to be responsible for its reliability. Both Dynamo and GFS rely on this argument in providing service. For each service, what can applications do to handle inconsistent data?

2. Reliability

Fault-tolerant systems often include an isolation mechanism, a failure-detection mechanism, and a recovery mechanism. In Nooks, isolation was provided largely by virtual-memory protection, failure detection by the processor raising exceptions for illegal behavior, and recovery by restarting the device driver.

(a) What is the value of isolation in a fault-tolerant system? What bad things can happen without isolation?

(b) Suppose Nooks used Wahbe's Software Fault Isolation instead of virtual-memory protection. What are the benefits of using virtual memory protection instead of software fault isolation? What are the drawbacks?

(c) Recovery often relies on redundancy of some form. Give an example of how Nooks relies on redundancy to recover from a driver failure.

3. Synchronizations and Race Detection

Synchronization is a classic topic in operating systems. In this question, we examine basic synchronization primitives including semaphores and monitors.

(a) In the THE paper, Dijkstra mentioned that semaphores can be used in two different ways. Please describe the two different types of synchronization that semaphores can achieve.

(b) Monitors can be used as an alternative to semaphores. Can monitors also conduct the above two types of synchronization? If yes, explain how you can use monitors to do that. If no, explain the reason.

(c) Please briefly describe the "lock-set" algorithm that Eraser uses to detect data races.

(d) Is lock-set algorithm useful for programs that use semaphores for synchronization? If yes, how would you adapt lock-set to work with semaphores? If no, explain the reason.

(e) Is lock-set algorithm useful for programs that use monitors for synchronization? If yes, how would you adapt lock-set to work with monitors? If no, explain the reason.

4. **Replay**

InstantReplay can deterministically replay the execution of parallel programs by recording the "partial order" among all accesses to shared objects (e.g., shared memory objects, semaphores, locks, etc.).

(a) Why is "total order" unnecessary for deterministic replay?

(b) Consider a scenario where a multi-threaded program executes on a multi-core shared-memory machine. How can you optimize InstantReplay to record less information, if the multi-threaded program is known to be data-race free? (Please use the data-race definition used in the Eraser paper)

(c) Rx is a system that uses replay to survive software failures; describe how Rx does so.

(d) Could InstantReplay be useful in implementing Rx? If so, explain how. If not, explain why not.

5. **Scheduling**

The Borrowed-Virtual-Time (BVT) scheduler was developed in the late 1990s as an alternative to "general-purpose operating system schedulers" and "specialized real-time schedulers".

To review, the basic idea behind of BVT scheduling is that it chooses to dispatch the runnable thread with the earliest effective virtual time (EVT); latency-sensitive threads are allowed to warp back in virtual time and obtain preference for being dispatched. The calculation for the EVT of a thread i, E_i, is computed as follows:

$E_i = A_i - (warp\ ?\ W_i\ :\ 0)$

where A_i is the actual virtual time of the thread, $warp$ is whether or not warp is enable for this thread, and W_i is the thread's virtual time warp. The BVT scheduler increments A_i, the actual virtual time, of a running thread by the number of timer ticks it consumed divided by its weight, w_i (note that this is different than W_i).

(a) What are the primary goals of BVT? What types of workloads is a BVT scheduler intended to support?

(b) A BVT scheduler needs to take care of some details in order to handle some important cases. For each of the following "details", briefly describe its purpose and give an example for which the modification matters.

i. The BVT scheduler doesn't necessarily dispatch the thread with the earliest EVT: it allows the time of the currently dispatched thread to advance beyond another runnable thread by a fixed amount C. Why? Example?

ii. When a thread becomes runnable after sleeping, it is assigned a new actual virtual time (A_i) equal to the "scheduler virtual time" (SVT), which is the minimum actual virtual time of any runnable thread currently in the system. Why? Example?

iii. A newly created thread is created with a non-zero warp, W_i. Why? Example?

iv. On shared-memory multiprocessors, each processor runs the earliest EVT thread of all the runnable threads, calculated as follows: $E_i = A_i - (warp\ ?\ W_i\ :\ 0) + M$. Why? Example?

6. **File Systems**

This question examines classic file systems and how they might need to change in order to adapt to new technologies.

(a) Classic file systems contain many policies that are tuned for hard disk drives. Consider the Berkeley Fast File System (FFS); what policies does FFS contain that are specific to hard drives?

(b) New computers are now being sold with flash-based drives instead of conventional rotating disks. Flash has this following performance property: excellent random I/O performance for *reads* (nearly the same as sequential). Assume that you run FFS on top of this raw flash device. What will the impact on read performance be? Just considering read performance, is FFS well-suited to flash?

(c) Flash also has this performance property: when writing to a *page* (assume a page size of 4KB), one must first *erase* an entire *block* (assume a block size of 128KB). Assume that you run FFS on top of this raw flash device. What will the impact on write performance be? Just considering write performance, is FFS well-suited to flash?

(d) Flash also has an unusual reliability property: when a particular block is erased “too many” (i.e., 10,000 or 100,000) times, it will “wear out” and thus become unreadable. Assume you run FFS on top of this raw flash. What will the impact on reliability be? Just considering reliability, is FFS well-suited to flash?