UNIVERSITY OF WISCONSIN-MADISON
Computer Sciences Department
Ph.D. Qualifying Exam

Operating Systems **Fall 2014**

Instructions: There are six questions on this exam; you must answer *all* six questions

---

Question 1: Virtual Memory

Mach provided a machine-independent virtual memory implementation. One of the most important components was the **pmap** structure.

A. Explain the structure and use of Mach's pmap structure.

B. Do you need a pmap structure on processors with software-filled TLBs? Please explain your answer.

C. Disco also used a pmap data structure. Explain the differences between Mach's pmaps and Disco's pmaps.

Question 2: Processes, LWP's and Threads

A. Most modern operating systems support processes (sometimes called heavy-weight processes), lightweight processes (sometimes called kernel-level threads), and threads (sometimes called user-level threads). Describe each of these constructs and compare them, identifying the key ways in which they differ.

B. Give an example where you would want to use each of these constructs.

C. An early complaint about user-level threads was that if a thread blocked while it was doing a kernel call, the process that was running the thread would be blocked, preventing it from running any other threads. Describe a way that systems have overcome this limitation.

Question 3: Global Snapshots

Consider Chandy and Lamport's global snapshot algorithm.

A. The global snapshot algorithm is based on detecting stable properties. What is a stable property (and give an example) and why does the algorithm contain this restriction?

B. When does this algorithm terminate? What guarantees are given on the termination time of the global snapshot algorithm?

C. Chandy and Lamport assume that the distributed program being snapshotted has a certain amount of nondeterminism. What forms of nondeterminism do they consider?

Question 4: GFS

The Google File System (GFS) shows us how to build a scalable cluster-scale file system for certain types of workloads. In this question, we'll discuss some of the design decisions in GFS.

A. One of the biggest decisions made in building GFS is the single master. Describe the function of the single master, and why having a single master might be a bad choice when it comes to scale.

B. Reads from a client are generally quite efficient within GFS. Describe what happens when a client issues a read (in the common case).

C. Writes from a client are a bit more complex. Describe what happens when a client issues a write (again, in the common case). Why are writes generally less efficient than reads?

D. One major concern with a large-scale storage system built atop cheap machines and disks is data corruption. Describe how GFS protects against such corruption.

Question 5: RAID

This question is about redundant arrays of disk drives, commonly called RAID. In this question, we'll discuss different forms of RAID.

A. One commonly used RAID level is RAID level 1, sometimes referred to as "mirroring". Describe how mirroring works, touching on its performance, reliability, and effective capacity.

B. Another commonly used RAID level is RAID level 5, sometimes referred to as "rotated parity". Described how RAID-5 works, touching on its performance, reliability, and effective capacity.

C. When do users prefer RAID-5 over RAID-1? When do users prefer RAID-1 over RAID-5?

D. The AutoRAID system avoids the decision of RAID-1 vs. RAID-5; describe how this works. If the AutoRAID system is not very full, what is its behavior like?

Question 6: MapReduce

This question focuses on MapReduce, a large-scale data analysis system from Google.

A. The MapReduce programming model simplifies the task of parallel programming by allowing users to specify only a few functions (i.e., Map, Reduce) and then taking care of the rest. What aspects of parallel programming are made easier due to this paradigm?

B. MapReduce also greatly simplifies programming by removing the concern of failure handling. Why don't users have to worry about failure? How does MapReduce handle failures of machines during execution?

C. Slow tasks are particularly onerous for MapReduce. What happens when one task is slow during the map phase? Can the reduce phase begin before the map phase ends? What about during the reduce phase? Is slowness of a reducer also a problem? Describe how MapReduce handles such slowness, and what would happen without such machinery.

D. Between the map and reduce phases, a large amount of data is typically shuffled between machines. Can you quantify how much data is (roughly) moved in a typical execution? (make any necessary assumptions)