Advanced Operating Systems
Spring 2013

Mid-term

Roll no: ____________________

Instructions. The exam consists of 30 multiple choice questions, each worth one mark. You have 2 hours and 30 minutes to solve this. If you think some question has no correct answer or multiple correct answers, you can write your assumptions/objections and argue with me later. If you are right, I'll drop that question for everyone or accept multiple answers as correct. The solution will be available on LMS after the exam.

How (and How Not) to Write a Good Systems Paper

1. What paper the authors' would not approve regarding novelty and references?

□ A new system with references of venerable work from the 90's since the authors' claim they built directly upon the old work.

□ A novel improvement that combines ideas from multiple recent systems from last few years and the author's argues that the work does not directly derive from any old work.

□ A survey of many old and new systems in a field. The authors argue that this is useful because it helps researchers in the field read only one paper instead of the many from different decades.

□ A survey of only two recent techniques but adding a comparitive analysis. The authors refer old systems, new systems, and other comparitive analyses and show that no one else compared these two systems against each other.

The Rise of "Worse is Better"

2. The following is an example of worse is better principle applied to WAPDA.

□ Using a single generic electricity meter that caters the needs of a small house up to a large enterprise.

□ Using lower grade copper wires that allow wider distribution.

□ Using a proof-of-concept prototype of a billing system for production use.

□ Using a well-designed transformer which is simple to build and simple to use for most consumers but large enterprise have to use complex aggregation of multiple transformers

Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism

3. When a thread is to be pre-empted, its state is changing and an UPCALL should inform it of the state change. If the process never returns from the UPCALL, it can take ultimate priority as it will never be pre-empted out. How do the authors solve this problem?

□ They do not solve it. In fact there is no problem. After all its a user thread library and delegates authority to the user level thread scheduler which will be well-designed.

□ The authors do not address it because its trivial to solve. The kernel can easily preempt the UPCALL after a given timeout. Since the user-space library knows the timeout, a well designed library will finish the work quickly.

□ The authors account for the extra time and deduct it from the next time slice given to the process. Thus the average time taken by the processes will be fair. This also leaves no motivation for the user-space thread scheduler to steal extra time.

□ The authors do not send a timely UPCALL when the last thread of a process is to be preempted.

4. Scheduler activations are kernel-to-user calls on every state change. Then why is their performance better than kernel threads which also need to go in the kernel for context switches?

□ A call from kernel to user space is much more efficient that a call from user space to kernel space because the kernel can prepare the context according to the user program.

□ They are not better and this is the main reason why scheduler activations have not been deployed in any real system. They are a research concept and the key contribution is in defining a novel interface, not in performance.

□ Scheduler activations are light-weight data structures containing a thread id whereas the detailed information is contained by the user-space thread scheduler which performs the actual context switch. Thus the UPCALL with the light-weight structure is much more efficient than a context switch performed completely in the kernel.

□ There are fewer schdeuler activations than context switches in kernel threads because of a different design. Thus the overall address space switches are less.

5. When a user-level thread blocks in the kernel, the kernel should inform the user-level thread scheduler but the kernel does not have the list of user-level threads. How can it communicate to the user-level thread scheduler?

□ Ofcourse it cannot. It just informs that *some* user-thread has blocked in the kernel and the user-space thread scheduler can query the threads to find one which one.

□ The kernel does have the list. Only the scheduling is controlled by the user-space scheduler.

□ The kernel simply uses an UPCALL to get information about the thread running on a virtual processor and can then pass that *id* to the user-space thread scheduler.

□ The kernel and user-space scheduler are both aware of scheduler activations. The kernel maps them to virtual processors while user-space scheduler to threads.

6. When one of three virtual processors allocated to a process is preempted, how many user-threads are pre-empted as a result and why?

□ None. Why would any user threads be preempted. The kernel does not know about user threads. We are not using kernel threads.

□ Ofcourse the user thread running on the preempted processor is preempted, nothing else.

□ All three are preempted so that the user-level thread scheduler can decide which two to continue and which one to stop.

□ Two user threads are preempted to use one for UPCALL and the third can continue to run whatever user thread it is running.

Threads cannot be implemented as a library

7. The following turn based synchronization solution does not work if:

```
P1 { ... if(turn==0) { foo(shared_var); turn=1; } ... }
P2 { ... if(turn==1) { bar(shared_var); turn=0; } ... }
```

□ It always works. It is a correct solution of the mutual exclusion problem. The only problem is *progress* which means that even if one process wants to enter its critical section when the other is in the remainder section, it may not be able to if this is not its *turn*.

□ A sequentially consistent memory model is being used.

□ `pthread_mutex_lock` and `pthread_mutex_unlock` are used before and after `foo` and `bar` calls.

□ Memory barriers are used before the `foo` and `bar` calls.

8. When reading variable X to later modify it, if the compiler reads more than necessary (e.g. to use 64-bit operations), how can that introduce a race condition?

- □ It cannot. Since we know the compiler will always read and write variable X atomically in one operation (memory operations are always atomic) so reading extra never hurts.
- □ Due to adjacent shared variables, we may introduce a race for variable X by overwriting two shared variables in one atomic operation.
- □ Due to the variable X being read in two parts e.g. when it does not lie on a 64-bit boundary. Thus a read which was atomic is no longer atomic.
- □ Due to adjacent shared variables, we may introduce a race for some variable other than X.

9. Whats a memory barrier and why is it necessary to introduce one when taking a lock? Is a one way acquire barrier good enough?

- □ A memory barrier pauses memory operations until the lock statement is completely executed. A one way barrier works for lock but not for an unlock operation.
- □ A memory barrier stops hardware reordering of memory operations across the barrier instruction. A one way barrier is good enough for both lock and unlock operations.
- □ A memory barrier is a barrier between other processors and the memory. This ensures that during the lock operation, other cores cannot access the same memory locations introducing the possibility of a race. A one way barrier is the recommended approach for a lock operation.
- □ A memory barrier guarantees an order between memory operations initiated by instructions before and after the barrier. A one way barrier is good enough for a lock operation.

10. What is the author's takeaway from Fig 1 and Fig 2?

- □ The *unsafe* approach is the best performing and the recommended approach.
- □ There is no point in adding more processors when using locks. Its better to not use locks and just a single processor.
- □ Applications using locks are expected to be take 3-5X the time taken by unsafe code.
- □ The are just arguing in favor of algorithms built around atomic instructions.

Experiences with Processes and Monitors in Mesa

11. How can a non-preemptive scheduler be used for mutual exclusion?

- □ We need semaphores, mutexes, monitors etcfor mutual exclusion. A scheduler is for scheduling not mutual exclusion.
- □ Inside a `yield` call, a non-preemptive scheduler can easily choose to switch to threads which will not access a shared variable being used by the current thread.
- □ Mutual exclusion is ensured because every source code instruction is now implicitly atomic, as we can only put `yield` calls before or after a line of source code.
- □ We can avoid putting any `yield` calls in a critical section.

12. Why does the race possible with naked notify does not happen with a normal NOTIFY?

- □ Interrupts are special as they can occur at arbitrary times. Threads, on the other hand, are only preempted at specific times.
- □ Well, the race condition can still occur but the authors suggest to be use the familiar *wakeup-waiting switch* to handle it.
- □ A thread cannot be preempted when it has checked a condition and not yet called WAIT.
- □ Two threads cannot be at the state when they have just checked the condition before the WAIT.

13. When a method in monitor A invokes a method in monitor B which calls WAIT, can other processes enter the monitors (in Mesa)? can this lead to deadlock?

□ Ofcourse. WAIT releases the monitor locks. Thats how Mesa prevents deadlocks.

□ No. The monitor locks have to be explcitly released. Thats exactly how deadlocks occur. The locks have to be acquired and released in proper order to avoid deadlock.

□ Only the monitor lock of A is released since it is the one directly called from external code. Indirect monitor locks cannot be released but they can cause deadlock that the programmer should prevent.

□ Only the monitor lock of B is released, otherwise it would be difficult to guarantee correctness of A. The held lock of A can cause a deadlock.

14. What does it mean to assign a priority to the monitor to solve priority inversion?

□ Monitor is not a running entity. A thread enters (i.e. uses) a monitor. The monitor does not execute itself. So priorities are only assigned to threads and not monitors.

□ Monitors are assigned a priority when they are declared (often the highest possible priority) and any thread entering the monitor assumes that priority. Thus every thread inside the monitor has the same priority and priority inversion cannot happen.

□ Monitors assume the priority of the highest priority thread waiting in the monitor. All threads waiting in the monitor assume this priority until they leave the monitor.

□ It means to raise (but never reduce) the priority of a thread executing inside a monitor method to $max(P_i)$ where P_i are all the threads waiting in monitor queues.

Eraser: A Dynamic Data Race Detector for Multithreaded Programs

15. In what aspect is happens-before superior to Eraser?

□ Its not. Eraser is a much advanced scheme that clearly improves upon previous work.

□ Happens-before provides guarantees that the code is race-condition free. Eraser cannot provide these guarantees but works much faster.

□ Happens-before finds every unprotected shared variable on the schedule it observes.

□ Happens-before finds races that happen in at least one schedule.

16. How does the lockset algorithm know that a variable is not protected by a lock?

□ The system is initialized with a mapping of variables to locks and Eraser makes sure the mapping is not violated.

□ The system assumes every variable is unprotected and accesses can violate that assumption. Any assumptions that still hold in the end are reported as potential race conditions.

□ The system builds a lockset index containing all the locks every held when accessing this variable. The lockset index can be easily analyzed to find violations.

□ The system assumes a variable is protected by every lock in the system and refines the assumption on every access.

17. With ref to Fig 5, how many lockset index entries will be there given that we have 5 shared variables and 10 locks in the system.

□ Simply 5. Each shared variable points to a lockset index entry.

□ Its 10. We need a lockset index entry for every lock.

□ All possible subset of locks. 2^{10} in this case.

□ Funny. How do I know.

18. What does it mean to say that effectiveness of tools based on happens-before depends on interleaving produced by the scheduler?

□ It does not. Happens-before can find races that will also happen in other interleavings.

□ Its true and the same is equally true for Eraser. Any dynamic testing tool will depend on the exact schedule followed.

□ When the interleaving produced by the scheduler takes a lock on one thread and then switches to another thread, any violations on the second thread are not reported as errors as the tool thinks that we are already holding a lock.

□ Because of a given schedule, locks on one shared variable can force a happens-before relationship on access of another unprotected shared variable and thus hide the error.

Design and Implementation of a log structured file system

19. When everything is in the log, why keep the checkpoint region and superblock fixed (Table 1)? Doesn't that hurt performance by introducing a long seek?

□ The Sprite LFS used a fixed checkpoint region for implementation simplicity but propose that it can be made dynamic in the future.

□ Its fixed but is replicated frequently (every 30s in Sprite LFS) in the log. Thus long seeks are not required.

□ Its fixed so you know where to find different blocks containing the inode map. It hurts performance and is exactly the reason LFS doesn't perform for a near-full file system. The seek time from outer cylinders to the checkpoint hurts performance.

□ Need to fix some meta data to find the rest. Performance is handled by reducing checkpoint frequency.

20. According to Fig 3, a log-structured file system can perform worse than a normal file system for writes. How is that possible when a log-structured file system needs no disk seek?

□ This is due to the long seeks for reading the checkpoint region because of a near full disk.

□ This is due to seeks from one segment to another. In an empty disk, we can find contiguous segments but with a near full disk, few segments are free here and there. If Sprite LFS used larger segments, the situation would improve.

□ This is due to the overhead of looking up inode maps and inode tables spread over the whole disk due to a near full disk. A normal file system gives consistent performance since meta deta is always placed at the same place.

□ This is due to time wasted on extra reads and writes not directly initiated by the user.

21. When new files are written, they are supposedly a combination of cold and hot files. How is age calculated and how does it help in separating cold and hot data.

□ Age of a segment is the time since the least recently modified block in the segment. It helps by splitting the distribution in a bi-modal graph.

□ Age of a segment is the time since the least recently modified block in the segment. Its used in the cost-benefit formula which results in the least write cost as shown in the paper empirically.

□ Age of a segment is the time since the most recently modified block in the segment. Segments with hot files will find their age often reduced whenever these hot files are modified.

□ Age of a segment is the time since the most recently modified block in the segment. A segment filled once cannot reduce in age. Hot files move out.

22. How many I/O operations (read and write) are required to increment the first byte of a file and checkpoint in Sprite LFS with a given i-node when nothing is in memory? Remember the two checkpoint regions.

- □ 7 operations.
- □ 8 operations.
- □ 10 operations.
- □ 9 operations.

Solaris ZFS and Red Hat Enterprise Linux Ext3 File System Performance

23. What does it mean to say that ext3 file system mount options require making a trade off between data integrity and performance?

- □ The ext3 file system introduces data integrity and better performance than ext2. The trade-off discussed is between compatibility offered by ext2 and these advanced features offered by ext3.
- □ To perform better, the ext3 file system increases the duration between checkpoints that make the file system consistent. However, this also increases the amount of data that can be lost (i.e. everything since the last checkpoint).
- □ The ext3 file system calculates checksums for data which can be disabled for better performance. However the trade-off of disabling checksums is reduced data integrity.
- □ The ext3 file system performs better wehn the journaling option is disabled.

24. In Fig 1.1 why the buffered read operations are significantly faster than ZFS?

- □ ZFS buffering was not enabled in these experiments. The authors were emphasizing that ZFS can perform well-enough even without buffering.
- □ ZFS allocation algorithm tries to spread file data on the disk to provide better data protection. However, the cost is paid in a slightly reduced read bandwidth.
- □ Buffered reads are more efficient for any file system based on in-place modification because the a particular block of a file is always on the same place on the disk and this information can be cached.
- □ A copy-on-write file system finds it difficult to keep data contiguous.

25. Why should we keep checksums in intermediate nodes only vs. checksum in every block?

- □ Checksums in every block check data integrity of all the data. It is good enough and used in many file systems.
- □ Checksums in intermediate nodes verify data integrity of all the meta-deta but not the data blocks themselves. This improves performance.
- □ Checksums in children are stored in intermediate nodes. A corrupted block results in the checksum stored in the parent block not validating. Thus it provides the same goals as checksums in every block, but faster.
- □ Checksums in intermediate nodes ensure that we understand the tree correctly. If a block becomes part of two files (due to a bug), then writing it and updating the checksum in the intermediate node will violate the checksum in the other intermediate node and the inconsistency cannot silently linger on.

26. What does it mean to say ZFS does not depend on utilities like fsck to maintain on-disk consistency?

- □ ZFS has its own file system consistency builder thats run when you mount the file system.
- □ ZFS can work well even with inconsistencies. Each directory and file is quickly scanned for inconsistencies when accessed.
- □ It takes frequent snapshots and auto-reverts when an inconsistency is found.
- □ You cannot find a point in time when it is inconsistent.

A Fast File System for UNIX

27. How are advisory locks different from normal locks?

- □ They are dictated by the program and forced by the system.
- □ Advisory locks are handled by a user-space lock manager like user-space threads. The lock manager can allow a program to access the file or not.
- □ Advisory locks are automatic locks that the system enforces itself by observing the pattern of file access.
- □ A program can easily access a file with advisory lock without acquiring the lock.

28. How are seek times reduced in the FFS?

- □ By spreading data over disk blocks.
- □ By keeping all directories and file inodes in one place for quick access.
- □ By allowing two block sizes where the half of the disk has one block size and the other half has the other block size.
- □ By spreading inodes over disk blocks.

29. Why does FFS opt for two block sizes?

- □ To reduce external fragmentation and balance it with internal fragmentation.
- □ To allow the user to choose which block size is best on a per-file basis.
- □ To significantly increase the performance for small files.
- □ To avoid making a trade-off between wasted space and performance.

30. How is locality of reference in FFS different from LFS?

- □ LFS does not use the concept of locality of reference.
- □ FFS does not use the concept of locality of reference.
- □ Its not different. The idea of keeping files contiguous is fundamental in the design of every file system.
- □ Its logical locality (based on access pattern) vs temporal locality (based on access time).