/ 51

Midterm Exam
CS 111, Principles of Operating Systems
Fall 2016

Name: _____

Student _____

This is a closed book, closed note test. Answer all questions.

Each question should be answered in 1-3 paragraphs. DO NOT simply write everything you remember about the topic of the question. Answer the question that was asked. Extraneous information not related to the answer to the question will not improve your grade and may make it difficult to determine if the pertinent part of your answer is correct.

1. What is the difference between the invalid bit in a page table entry and the invalid bit in a translation lookaside buffer entry? What happens in each case if an address translation attempts to use that entry? Is it possible for both to be set? Why?

The invalid bit in a page table entry shows weather the page is in the TLB or not. When address translation attempts to use the entry, a page fault is generated, and the page is fetched from disk. page table

The valid bit in the TLB indicates if the page is clean or dirty. Dirty pages have been modified and must be updated on the disk before being used.

It is not possible for both to be set because pages must be cleaned before they are removed from the TLB.

(3)

2. There are many difficult issues that arise due to uncontrolled concurrent executions. Why do we not simply turn off interrupts to prevent such problems from arising? Why not always? Why not just for all critical sections of code?

Turning off interrupts is not an effective way to make code mutually exclusive. First, turning off interrupts are not supported in user mode because it is a priviliged instruction.

Turning off interrupts prevents a thread from being descheduled, but it may also cause deadlock if there is a memory request. The thread will block infinitely while waiting for a memory request return.

Turning off interrupts is also not supported for multicore programming.

preemptive, i/o

(5)

3. What issues arise in management of thread stacks that do not arise in management of process stacks? Why do these issues arise and what is typically done to handle them? What are the disadvantages of this approach and why is the approach used despite those disadvantages?

Thread utilize shared data segments, while they have their own PC, registers, and stack. Processes do not share data unless an explicit inter-process connection is established. Threads must monitor their shared data so that there will not be any race conditions and the data will not enter an indeterminate state. To eliminate race conditions based on shared data between threads, locks are used to prevent more than one thread manipulating shared data at once.

fixed sized stacks?

2

Threads are still useful because they take less overhead to create than new processes and they allow for faster parallel programing on multi core machines. It is also easier to share data between threads vs. processes.

4. In MLFQ scheduling, processes are moved from one scheduler queue to another based on their behavior. Each such queue has a particular length of time slice for all processes in that queue. Why might a process be moved from a queue with a short time slice to a queue with a long time slice? How can the operating system tell that the process should be moved?

A process will be moved from a queue with a short time slice to a longer time slice queue to improve overall performance of the system. The OS can tell when the process should be moved by if the process consistently does not block during its time slice.

4

Moving it to a larger time slice will increase efficiency because the process will have a longer time to complete its task (or hit a point where it needs to block) before being descheduled. The process will be able to finish its job with a lower number of time slices, and therefore less context switches. Minimizing context switches minimizes overhead and increases efficiency.

less i/o

5. Will binary buddy allocation suffer from internal fragmentation, external fragmentation, both, or neither? If it does suffer from a form of fragmentation, how badly and why? If it does not suffer from a form of fragmentation, why not?

The binary buddy allocation recursively divides available memory into 2 until it finds the smallest block that will satisfy the request. Buddy allocation suffers from both internal and external fragmentation. Internal fragmentation occurs because only blocks of 2^n can be allocated, and not all of the memory may be utilized. → explain External fragmentation occurs because small blocks between allocated memory may appear, and they will be too small to be useful. Internal fragmentation almost always occurs unless the memory request is exactly of size 2^n. External fragmentation is easily combatible with the buddy system because the starting address of buddy blocks differ by 1 bit. This makes combining unused memory easy.

6

6. What is the purpose of a trap table in an operating system? What does it contain? When is it consulted? When is it loaded?

The trap table is a table that directs the OS to what code the execute once a certain trap is called. It contains an ID unique to the trap or system call and information of where to find the code relating to the specific trap. The trap table is consulted when a trap is called (system call, memory request, etc). It is loaded during the OS boot time.

8

7. Assuming a correct implementation, do spin locks provide correct mutual exclusion? Are they fair? Do they have good performance characteristics? Explain why for all three of these evaluation criteria (correctness, fairness, performance).

If implemented correctly, spin locks will provide correct mutual exclusion (but correct implementation may be tricky). Spin locks are only as fair as the scheduler, since they rely on the scheduler to interrupt a spinning thread at certain intervals in order to update conditions that may depend on a different thread.

one exception → If a thread has top priority and the scheduler doesn't allow preemption, a spin lock will spin forever because other threads won't be able to run.

5

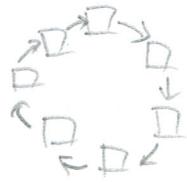
Spin locks generally have poor performance because waiting threads waste their entire time slice doing nothing. Additionally, the scheduler does not know which threads are waiting or not, so it may schedule multiple waiting threads back-to-back, amounting to a whole lot of nothing.

8. What is the purpose of using a clock algorithm to handle page replacement in a virtual memory system? How does it solve the problem it is intended to address?

The clock algorithm is an approximation of the least recently used (LRU) algorithm. In LRU, a time stamp is generated when a page is referenced, and when a page must be replaced, the page with the oldest time stamp is removed. This naive implementation would be costly because it requires a structure to keep the time stamps, and requires searching through an entire list of time stamps to find the oldest.

The clock algorithm bypasses these limitations by creating a circular list of pages and using one bit to indicate if the page has been referenced recently or not. The clock algorithm can use the position on the circular list as an approximation of time. As it runs through the list, it will eject the first page that has not been referenced (no bit set), while unsetting bits of pages that have been recently used.

7



9. What are two fundamental problems faced by a user level thread implementation that are not faced by a kernel level thread implementation? Why do these problems arise in user level implementations? Why don't they arise in kernel level implementations?

User level thread implementation cannot effectively make use of the scheduler because the scheduler is run by the OS. The OS is not aware of user level threads, but is aware of kernel level threads.

Also, if the scheduler deschedules a kernel level thread that has multiple user level threads, all of the user level threads will to halted.

(4)

multiprocessor & thread blocking issues

10. What is the difference for a virtual memory system between segmentation and paging? Why might both be used in a single system?

Segmentation is used to move larger blocks of code related to the process, such as shared libraries, the code segment, and stack. Segmentation allows for more space efficiency within physical memory because these blocks can be moved within physical memory and then referenced with a base and limit registers.

Paging is used for more efficient memory allocation and timely memory referencing. Paging allows faster memory access by utilizing temporal and spacial locality within code and data.

(7)

Both may be used in a single system. Segmentation can be used to relocate blocks of allocated memory (pages) within physical memory & to more efficiently use physical memory.
↳ more compact