CS111 Midterm 05/03/2018

Name Aashita Patwani

Student ID # 004810708

Seat Row 6 Seat Col 10 Exam # 110

This is a closed-book, no-notes exam.

All questions are of equal value. Most questions have multiple parts. You must answer every part of every question. Read each question CAREFULLY; Make sure that

you understand EXACTLY what question is being asked

what type of answer is expected

your answer clearly and directly responds to the asked question

Many students lose many points for answering questions other than the one I asked. Misunderstanding a question may be evidence that you have not mastered the underlying concepts.

If you are unsure about what a question is asking for, raise your hand and ask.

Spend more time thinking and less time writing. Short and clear answers get more credit than long, rambling or vague ones.

Write carefully. I do not grade for penmanship, spelling or grammar, but if I cannot read or understand your answer, I can't give you credit for it.

1: (a) Define "Information-Hiding" (in the context of s/w design):

Information-Hiding is where interfaces are simplified to reduce side-effects.

(b) Briefly explain why Information-Hiding is a good thing:

Information-Hiding makes it easier for a user to use an interface if ~~for example~~ they do not ~~need to~~ know all the details of the implementation. For example, all the addresses visible to a user are virtual addresses and it appears to them that they have a contiguous block of addresses, when in reality, these might be mapped to different sections in physical memory.

(c) Define "Modularity" (without using the word "module"):

Modularity is when functionality is split into different ~~sections~~ components that can be combined to use for a certain implementation.

Heirarchical
general - [illegible]
Ordered

(d) Briefly describe a (covered in this course) characteristic of good modular decomposition:

A ~~characteris~~ characteristic is Heirarchial Decomposition where it makes it easier to understand as you only have to understand one-level at a time.

2: (a) What does the acronym "ABI" stand for?

Application Binary Interface.

(b) Define the term?

An ABI is what binds an API to a particular instruction set architecture.

(c) Why is ABI compatibility preferable to API compatibility?

An ABI compliant program is portable and can be run in all systems that support the same interface.

API compatibility is language specific.

(d) When might it be necessary or reasonable for two OSs that support the same APIs to not support the same ABIs?

When you want a program to run in different types of systems - greater generality.

3: (a) Give one advantage of static (non-shared) libraries over user-supplied object modules.

Common functions such as printf are included in the libraries so a ~~the~~ user does not have to worry about how to implement it.

- what mechanism does it on basis of what considerations

(b) What determines WHICH object modules FROM WHICH libraries become incorporated into a load module?

The mechanism that determines this is linkage editing. This is done on the basis of what unresolved references there are on the load module.

(c) Briefly list two advantages of shared libraries over ordinary libraries?

1. Reduces size of the load module and hence the memory it takes up and speeds up start up time

2. The version of the library is decided at load time and is not frozen at compile time so there is no need to recompile if the version changes.

4: (a) What fundamental problem (or truth about processes) motivates the use of multi-level feedback queues for process scheduling?

Processes have different levels of interactivity – for example, some require more I/O operations while some do more computation

(b) State TWO DISTINCT ways a process might find its way onto the right queue. program behaviors [illegible] may be changing

(1) If it uses / does not use up its time slice, its priority is lowered / increased.

(2) If a process uses up its time allocation (regardless of whether its used up its time slice) its priority is lowered.

(c) What would be the negative consequences of a process being on the wrong queue (provide an answer for wrong in each direction)?

(c1) If a low priority process is in a higher priority queue, it will slow down the response time of other processes in lower queues as it is run to completion (assuming single process in the queue)

(c2) If a high priority process is in a lower priority queue, it may take a long time before the scheduler reaches the process slowing its response time and it will take a longer time for it to complete as it will probably give up the CPU before the time slice ends and will have to wait again for the scheduler to reach the lower queues, even if it is ready to run.

5: (a) What is the primary problem associated with fixed-partition memory allocation (returning fixed sized regions that may be larger than the requested size)?

Internal fragmentation (not effective ~~memory~~ memory utilization)

(b) Briefly explain how special pools of fixed-size buffers affect this problem?

In special pools of fixed size buffers, this size corresponds to a 'popular size' ~~frequently~~ requested by a program. Hence, this improves the problem as it offers a perfect fit for majority of the requests, ~~avoid~~ reducing internal fragmentation.

(c) What new problem is likely to arise when we create such pools?

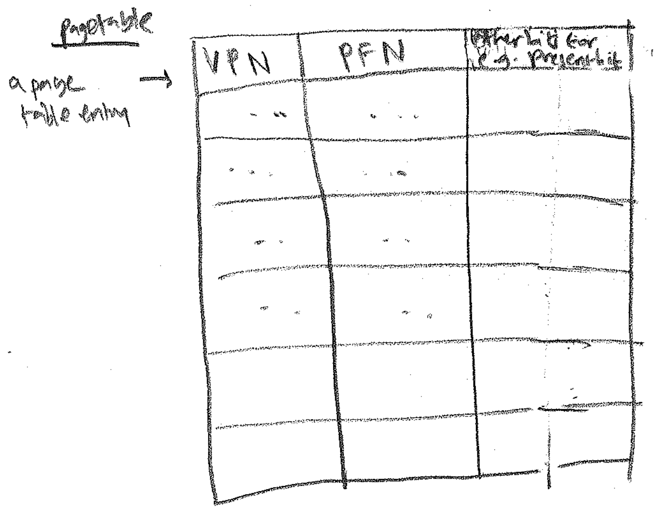
If the pools are too big, not all the buffers will be utilized and this will lead to waste of memory due to external fragmentation (for example all other requests require larger-sized buffers). But if they are too small, it will form a bottleneck and affect performance. Hence, the problem is deciding an optimal size.

(d) Briefly describe an approach for dealing with that problem?

~~An~~ An approach is using dynamic equilibrium where the opposing forces (increasing & decreasing the size) run, until a balance is formed where performance is optimal and performance is measured

No need TLB

6: (a) Draw a diagram of a paging MMU, and illustrating how it translates a virtual address into a physical address.



a virtual address consists of a virtual page number (VPN) and off-set. The VPN is used to index into the page table and get the corresponding physical frame number (if the page is valid/present)

The corresponding physical address = PFN + offset (same as that in virtual address)

(b) List (and briefly describe) two key pieces of information (other than the physical page frame number) that one might find in a page table entry

① Whether the page is present in memory. If it is not, a page fault occurs and the page will need to be swapped in from disk.

② Read/write access. If the page is read only/read and write, ~~so~~ so if changes can be made to the page.

(c) Given the (relative simplicity) of paged virtual address translation, why has it been necessary to create Translation Look-Aside Buffers?

~~As programs grow in size~~

Programs have become larger and creating a page table for every single page is alot of overhead as programs tend to exhibit temporal and spatial locality where some pages are referenced alot more often than others. TLBs help optimize this by storing popular pages in the cache and speeding performance.

7: Define (and distinguish the differences between) the following terms:

(a) "indeterminate"

There can be multiple possible outcomes.

(b) "non-deterministic" ... distinguish from "indeterminate"

The execution (process) can happen differently for every run. Indeterminate refers to the outcome, non-deterministic refers to the order in which everything is run.

(c) "race condition"

Operation where the result ~~of an operation~~ depends on timing.

(d) "critical section" ... distinguish from "race condition"

The correctness of the operation depends on timing.

Race condition is simply the result but critical section refers to whether it is correct or wrong.

(e) "atomicity"

An Operation that is run without interruption.

8: The text gave three criteria in terms of which lock mechanisms should be evaluated. In class this list was expanded to four criteria. List and briefly describe three of those criteria AND provide an example of a real locking mechanism that does poorly on each criteria ... briefly explaining why it does poorly.

performance
fairness
progress

(a) Mutual Exclusion / correctness. Interrupt disables does poorly on this as it is not effective in multiprocessor systems and multiple threads can access a critical section.

(whether the program runs correctly)

(b) ~~Fairness~~

Performance. If there is contention, spin locks do poorly in this as the resource holding thread might be preempted and since the other thread cannot access this ~~block~~ resource, it will simply spin and waste CPU time, not using it effectively.

(how effectively resources are utilized)

(c) Progress. Mutexes may not perform well for this criteria if not implemented correctly as there might be a deadlock where one thread has acquired the lock and is blocked but another thread needs to also acquire the lock to unblock it.

If there are resource convoys formed or deadlocks.

if event [illegible] wait semaphore

9: The text discussed both semaphores and condition variables as mechanisms that could be used to implement asynchronous event notification and waiting.
(a) Describe an important difference in what the waiter can assume after resuming after wait on a counting semaphore and on a condition variable.

~~[illegible]~~ With a counting semaphore, the waiter can assume that there is an event notification but with a condition variable, the state might have changed where for example the notification could have already been processed.

(b) Briefly explain why semaphores and condition variables are different in this respect.

Semaphores have a counting variable that can only be modified using wait or post so a new thread cannot enter, whereas in condition variables, the signal simply informs of the event notification, it does not control and change the state who is able to access it (ie. it does not mandate that the state will remain the same after the waiter resumes)

(c) Briefly describe a situation where this difference would make semaphores a better choice.

If for example there is one producer 'producing' the event notifications and multiple consumers that can process the notification, a semaphore would be a better choice as with condition variables, a consumer might attempt to process a notification that is no longer there. (similar to bounded buffer problem).

(d) Briefly describe a situation where this difference would make condition variables a better choice.

In a situation where it does not matter who gets to the resource first (e.g. one producer & consumer), condition variables are simpler to implement.

10: (a) What is the primary advantage of "enforced" (vs advisory) locking?

It ensures that that the shared resource is protected (does not leave room for mistakes due to user implementation) by incorporating locking within the mechanism.

(b) Describe a problem characteristic that would make "advisory" preferable?

When the user wants flexibility on what is locked or unlocked if for example a shared resource is being updated the user can decide who is able to access it when.

(c) What is required to make it possible to "enforce" locking?

When resources are persistent for example file systems.

XC: (a) What otherwise difficult/expensive problems (be very specific) do "clock algorithms" address?

Clock algorithms address the problem of long expensive searches to find what was least recently reduced.

(b) What are the key elements of a "clock algorithm"?

① A bit that indicates whether ~~to~~ a ~~process~~ page was recently referenced
→ set to 1 when a process references a page
→ when searching for a page to evict, the bit is reseted after checking that it is equal to 1

② Pages are arranged in a circular manner where · to find a page to evict you find the first zero from where the 'clock' is pointing to as that signals that it hasn't be referenced recently

(c) Briefly describe an LRU Clock Algorithm for deciding what old thing to remove from your refrigerator to make room for a new thing. I specifically want to understand how you implement your progressive scan, and what your "recently used" test is.

① Place items in the fridge from front to back in the order in which it was bought (newest back, oldest front).

② when deciding what to remove examine the first item, check if it is expired, If it is that ~~is~~ throw it away. → approximating least recently used as the items are examined in order of when it was bought from oldest to newest
If not ~~w~~ check the item behind it.

③ Place new item at the back, push all other items ahead.

④ If none ~~the~~ of the items are thrown and you reach back to the first item (after examining the last one) → throw it away. (the 1st one)

(d) Explain why your progressive clock-scan yields a reasonable approximation of Least Recently Used.

It examines from oldest item to newest item to determine what can be evicted (not referenced is seen as meaning expired in this case). If it seems like none of them can, you simply throw away the oldest item (the 1st item checked. Similar to ~~where~~ in clock where the bit is reseted after checking, so if you go through the entire clock without finding a 0 bit and arrive at the first you checked that is the LRU page)

(e) Not unlike Global LRU, this seems a clumsy mechanism, in that it imposes a more-or-less constant replace-by age on all items, even though some expire in days while others are good for years. Describe changes to your scan algorithm and "recently used" test to implement "Working Set Clock" replacement. I specifically want to understand your progressive scan, what your "recently used" test is, and how you set the key comparison parameters.

The 'working set' is analagous to the amount of space in the fridge each category of items take up.

Separate items into categories and give them allocated spaces ~~when searching~~ based on how ~~so~~ durable they are (short or long expiries) (frontmost, backmost). When searching for an item to remove, examine the space using the approach mentioned in (d) from the shortest expiry date category to the longest. So if for example, a new item belongs to category B and that and item ~~far~~ from category A is removed, that space now belongs to category B and that [illegible] working set (~~[illegible]~~ ~~the order of~~