CS111 Midterm Exam

Justin Ma

TOTAL POINTS

78.5 / 110

QUESTION 1

Principles 10 pts

1.1 Define Info Hiding 2 / 2

✓ **- 0 pts** Correct

1.2 Value of Info Hiding 3 / 3

✓ **- 0 pts** Correct

1.3 Define Modularity 2 / 2

✓ **- 0 pts** a system is composed of distinct sub-components

1.4 Good Modularity 0 / 3

✓ **- 3 pts** wrong

QUESTION 2

ABIs and APIs 10 pts

2.1 ABI acronym 2 / 2

✓ **- 0 pts** Application Binary Interface

2.2 ABI definition 2 / 3

✓ **- 1 pts** partial answer

2.3 ABI vs API 1 / 2

✓ **- 1 pts** missing the key word "binary program" or "recompile"

2.4 When API over ABI 3 / 3

✓ **- 0 pts** They are using different instruction set architectures.

QUESTION 3

Libraries 10 pts

3.1 Static library - advantages 0 / 3

✓ **- 3 pts** no need to implement or explicitly include the module in the compilation or linkage edit.

This is not different from explicitly included object modules

3.2 Which modules loaded 0 / 3

✓ **- 3 pts** the linkage editor deals with unresolved external references by pulling in the first module (in the specified library search order) that can satisfy it..

3.3 Shared library - advantages 4 / 4

✓ **- 0 pts** Correct

QUESTION 4

Multi-Level Queues 10 pts

4.1 What problem they solve 1 / 2

✓ **- 1 pts** Not enough. Should clearly mention that different processes have different behavior or we may want to give different processes different time slices.

4.2 What drives queue changes 4 / 4

✓ **- 0 pts** Correct

4.3 Consequences of wrong queue 4 / 4

✓ **- 0 pts** Correct

QUESTION 5

Fixed Paritition Allocation 10 pts

5.1 problem with it 3 / 3

✓ **- 0 pts** Internal Fragmentation

5.2 effect of special sub-pools 0 / 3

✓ **- 3 pts** incorrect

5.3 problem with special sub-pools 0 / 2

✓ **- 2 pts** incorrect

5.4 preventing that problem 1 / 2

✓ **- 1 pts** Incomplete

This should be done dynamically

QUESTION 6

Paging MMU 10 pts

6.1 diagram MMU, translation 5 / 5

✓ **- 0 pts** Correct

6.2 info in page table entry **2 / 3**

✓ **- 1 pts** Only one information was correctly described.

6.3 motivation for TLA buffers **2 / 2**

✓ **- 0 pts** Correct

QUESTION 7

Synchronization Terminology 10 pts

7.1 indeterminate **2 / 2**

✓ **- 0 pts** Correct

7.2 non-deterministic **2 / 2**

✓ **- 0 pts** Correct

7.3 race condition **0 / 2**

✓ **- 2 pts** incorrect

7.4 critical secstion **0.5 / 2**

✓ **- 0.5 pts** not mentioning 'correctness'

✓ **- 1 pts** too close to race condition

7.5 atomicity **2 / 2**

✓ **- 0 pts** Correct

QUESTION 8

Correct locking criteria 10 pts

8.1 criteria and mechanisms that fails **4 / 4**

✓ **- 0 pts** Correct

8.2 criteria and mechanisms that fails **2 / 3**

✓ **- 1 pts** did not describe how/why a real locking mechanism fails

8.3 criteria and mechanisms that fails **2 / 3**

- 1 Point adjustment

deadlock is not a fundamental problem with the locking mechanism, but with how it is used.

QUESTION 9

Asynchronous Completion mechanisms 10 pts

9.1 semaphores vs condition variables **3 / 3**

✓ **- 0 pts** Correct

9.2 why they differ **2 / 3**

✓ **- 1 pts** Condition variables are only wakeups, and make no guarantees about the state of the resource.

9.3 when choose semaphores **1 / 2**

- 1 Point adjustment

read write locks are much easier with condition variables because of the mandatory queuing in semaphores.

9.4 when choose condition variables **2 / 2**

✓ **- 0 pts** Correct

QUESTION 10

Enforced locking 10 pts

10.1 advantage of enforced **4 / 4**

✓ **- 0 pts** Correct

10.2 when choose advisory **2 / 4**

✓ **- 2 pts** When the access control we need is more nuanced than simple SHARED/EXCLUSIVE (e.g. different operations have different serialization rules).

if the file is read only, no locking is required.

10.3 requirement for enforced **2 / 2**

✓ **- 0 pts** Correct

QUESTION 11

Clock Algorithms 10 pts

11.1 what problem they solve **1 / 2**

✓ **- 1 pts** 1. finding the absolutely LRU element in a very large list involves a very long and expensive search. 2. updating a time on every reference would greatly slow down the system.

11.2 elements of clock algorithm **1 / 2**

✓ **- 1 pts** (1) progressive scan (2) through circular list (3) consulting a referenced bit to finditems unreferenced since last scan

you are talking about working set clock, not clock algorithms in general

11.3 refrigerator LRU algorithm **1 / 2**

✓ **- 1 pts** A clock algorithm is a progressive scan that stops with the first non-used item it encounters.

11.4 approximation of LRU **2 / 2**

✓ **- 0 pts** **Correct**

11.5 regrigerator working set algorithm **2 / 2**

✓ **- 0 pts** **Correct**

CS111 Midterm 05/03/2018

Name Justin Ma

Student ID # 604805167

Seat Row 3 Seat Col 10 Exam # 44

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 hiding the implementation from the user since the implementation is complex and unnecessary to understand to actually interact with the progam.

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

We don't want the user to worry about the implementation side of modules, since they are complex and may change with updates. The user should only be exposed to the interface, which remains mostly the same.

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

Breaking down a big, complex structure into smaller ones that have specifically-defined functions. These smaller pieces are easier to understand and work with.

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

Instead of having a single function that takes care of both locking and unlocking depending on the parameters, we seperate them into distinct functions. This makes each function simpler to understand and use.

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

Application Binary Interface

(b) Define the term?

It defines how a program is compiled in low-level binary level. It determines things such as endianess, linkage, etc.

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

ABI is much more fundamental and determines compliation. In fact, API is based on ABI. All programs that are not ABI compatible will not be able to run correctly.

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

Some APIs are necessary to work accoss different platforms, so they must be able to support different ABI's

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

Faster runtime

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

The linkage choice (static, shared, dynamically-loaded). Static chooses to have the object modules incorporated in the load module, but shared and dynamically-loaded do not.

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

shared libraries don't duplicate the same object modules if there are multiple instances to it. Thus, it sakes memory. Also, updating an executable is much simpler since the libraries are not attached to it.

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

We don't know how long each process will take.

(b) State TWO DISTINCT ways a process might find its way onto the right queue.

If it runs and takes up its entire time slice, it will be moved down a queue. However, if it gives up the CPU or is blocked by an I/O request before the time slice is up, it will remain

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

(c1) A really long process does not belong on a high priority queue. If it's there it will hog the CPU and use it without giving other processes in lower priority queues a chance.

(c2) On the other hand, a short process does not belong on a low priority queue. If it ends up there, it may take a long time before it gets a turn to run, increasing turnaround time.

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, which is when there exist unused/wasted space inside allocated memory.

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

Some regions will need to expand if they contain the end of a stack or heap. Having regions larger than requested mitigates this problem.

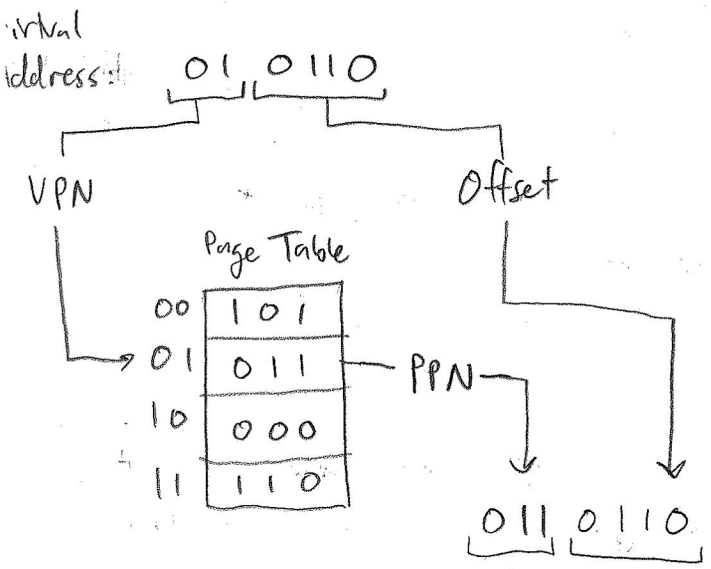
(c) What new problem is likely to arise when we create such pools?

When separate pools are created, external fragmentation arises, which is when space between allocated space becomes too small to be usable. This inbetween space is thus wasted.

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

Defragmentation reorganizes allocated memory to be continuous instead of scattered. This merges the scattered free space together, which stomps out external fragmentation.

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



VPN is used to index into page table to find the correct PPN. The offset remains the same.

(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

The PTE may also contain a valid bit which tells us whether the physical page frame number is the one we should use or not. If not valid, we will have to go into memory/disk to retrieve it.

It may also contain a present bit which tells us whether the page is present in memory or on the disk.

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

TLB's are much faster since they work like a cache storing the most recently used pages. This aims to minimize going to the page table in memory which cuts the number of memory accesses to 1.

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

(a) "indeterminate"

unknown instead of unpredictable

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

The results are unpredictable. This can be due to the unpredictability of race conditions.

(c) "race condition"

when threads compete for a shared resource

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

Section that involves modifying a shared resource. Racee condition describes a situation. A critical section describes a section of code.

(e) "atomicity"

merging instructions to turn into a single block that cannot be interrupted within.

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.

(a) performance - does not waste energy ; effient design

Ex: spin locks have bad performance since they constantly check if unlocked

(b) correctness - locks are mutually exclusive meaning when one has the lock, no other has the lock.

Ex: disabling interrupts don't guarantee correctness

(c) progress - avoids deadlocks where both threads are waiting for the other to progress.

Ex: mutexes are deadlock prone if not used correctly since threads can easily take 2 locks at a time.

(d) fairness - locks allow all threads through eventually - no starvation

Ex: spin locks have no queue so some threads can potentially starve.

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.

After resuming after waiting on a semaphore it can assume it was next in queue. However, there is no queue for condition variable so fairness is not guaranteed.

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

Semaphore have a counter that describes how many threads are using the shared resource while condition variables only allow one at a time. Semaphores also have a queue to guarantee fairness while condition variables do not.

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

Read-write locks work well for semaphores since more than one thread can read at a time (but only 1 thread can write at a time)
The queue guarantees that all threads will be able to read

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

Having a condition variable mark whether a process has completed or not is a good choice when all other processes read from that variable. Anything where fairness is not important

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

Enforced locking strictly ensures mutual exclusion will occur (you cannot access a file when another is writing to it)

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

When all users are merely reading from a shared file, no mutual exclusion is required. Advisory locking is loose and will not care. It's less overhead.

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

enforce locking requires system calls every time a user tries to access an unavailable resource.

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

They keep track of how recently used an object is. This addresses thrashing which is when young items are thrown out constantly.

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

Each item as a timer that keeps track of age. A target age is used as the threshold limit for how old an item can go on being unused.

(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.

Set a target age (anything older should be thrown out).
Each item ages on its own. Age is defined as amount of time that passes when object is not in use. Also, each item has a recently-used marker that begins at 1, and is set to 1 before every use. Have a progressive clock-scan that goes through each item and If recently used marker = 1, set it to 0. If marker = 0, Check the object's age. If it's past the target age, throw it out to make room for a new item.

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

The recently used bit starts at 1 and is decremented to 0 every time the clock-scan passes through. However, it goes back to 1 if it is used. The clock-scan throws out items it scans at 0 which means it has not been used for 2 cycles, which is a reasonable approximation of LRU.

(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.

Have the target age be the time before something expires, so that everything below the target age should be thrown out.