CS111 Midterm Exam

Devika Karumathil Chaudhuri

TOTAL POINTS

85 / 110

QUESTION 1

Principles 10 pts

1.1 Define Info Hiding 0 / 2

✓ **- 2 pts wrong**

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 3 / 3

✓ **- 0 pts Correct**

QUESTION 2

ABIs and APIs 10 pts

2.1 ABI acronym 2 / 2

✓ **- 0 pts Application Binary Interface**

2.2 ABI definition 3 / 3

✓ **- 0 pts a binding of API to ISA**

2.3 ABI vs API 2 / 2

✓ **- 0 pts without recompilation, one binary program works on all compliant platforms**

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 1 / 3

✓ **- 0 pts Correct**

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

- 1 Point adjustment

static libraries are not already in memory

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 3 / 4

- 1 Point adjustment

static libraries are not brought in at compile time, but at link time.

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.**

Use I/O as example is not enough.

4.2 What drives queue changes 4 / 4

✓ **- 0 pts Correct**

4.3 Consequences of wrong queue 3 / 4

✓ **- 1 pts (c2) Partial correct.**

Why waste time by waiting?

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 3 / 3

✓ **- 0 pts Correct**

5.3 problem with special sub-pools 0 / 2

✓ **- 2 pts incorrect**

5.4 preventing that problem 0 / 2

✓ **- 2 pts incorrect**

QUESTION 6

Paging MMU 10 pts

6.1 diagram MMU, translation 5 / 5

✓ **- 0 pts Correct**

6.2 info in page table entry 3 / 3

✓ **- 0 pts Correct**

6.3 motivation for TLA buffers 2 / 2

✓ **- 0 pts Correct**

QUESTION 7

Synchronization Terminology 10 pts

7.1 indeterminate 0 / 2

✓ **- 2 pts Incorrect**

7.2 non-deterministic 0 / 2

✓ **- 2 pts incorrect**

7.3 race condition 2 / 2

✓ **- 0 pts Correct**

7.4 critical secstion 0 / 2

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

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

✓ **- 0.5 pts No mention serialization**

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 3 / 3

✓ **- 0 pts Correct**

8.3 criteria and mechanisms that fails 3 / 3

✓ **- 0 pts Correct**

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

✓ **- 0 pts Correct**

- 1 Point adjustment

pthread_cond_signal is not a broadcast

9.3 when choose semaphores 2 / 2

✓ **- 0 pts Correct**

9.4 when choose condition variables 0 / 2

✓ **- 1 pts you did not show why CVs were preferred ... not merely less unpreferred**

- 1 Point adjustment

count to 1

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).**

both involve system calls, and advisory locking is likely much more work for the programmer.

10.3 requirement for enforced 2 / 2

✓ **- 0 pts Correct**

QUESTION 11

Clock Algorithms 10 pts

11.1 what problem they solve 2 / 2

✓ **- 0 pts Correct**

11.2 elements of clock algorithm 2 / 2

✓ **- 0 pts Correct**

11.3 refrigerator LRU algorithm 2 / 2

✓ **- 0 pts Correct**

11.4 approximation of LRU 2 / 2

✓ **- 0 pts Correct**

11.5 regrigerator working set algorithm 2 / 2

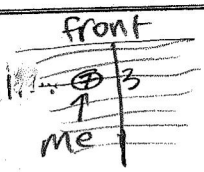
✓ **- 0 pts Correct**

Midterm 05/03/2018

Devika Chauchun

Student ID # 3045973339

Seat Row 3 Seat Col 7



Exam # 46

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 when a function or process takes in an input, does something to that input and outputs the result without knowing where the input was coming from or where it is going or why it is needed

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

- it gives each running process/ event + privacy. for example if a process is keeping track of bank accounts and needs to add two numbers together, the adding function should not know to which account the numbers belonged to, or even that its from a bank account process
- it also gives flexibility to rewrite ~~programs~~ functions without breaking the processes that call it b/c the processes don't know what is contain in that function

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

↳ when parts of a process are split up into different working components. Each component does not know about the existance of the others or what they do. Each component just does what they are supposed to do with the input they are given and output the result (this result may become the input to another component, but it doesn't know that)

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

- each function can be used in diverse ways
- each function only is used to perform one specific task; the tasks are split up into their own functions
- the functions are used just by their interface, not by how each function is implemented, so if the function's implementation was changed, the calling programs would still work (but not output behavior)

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

application binary interface

application programming interface

(b) Define the term?

The ABI binds the API to the ISA. It defines the format of the binary load module.

↑ instruction set architecture

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

- ABI compatibility means that there is no need to recompile the program, the created load module will run on any OS that has the same ABI, no matter what the API was
- API compatibility is just at the source code level, but would have to be recompiled if two OSs didn't have the same ABI

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

- The two OSs might want portability and communication, which requires that their programming syntax (API) is the same.
- One OS might want to use a different ABI when compiling, though, because maybe that ABI is more efficient, or it binds to their specific ISA better.

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

static libraries are already in memory. users would have to write their own functions if they didn't use the library. → less source code

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

there are stubs that point to the start of the specific object module. The program is linked with this stub's address, and the real address of the object mode is filled at load time.

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

1. shared libraries save memory, because (static libraries) instead of having one copy for each process that uses it, each process references the same spot in memory.
2. static libraries are brought in at compile time while shared libraries are brought in at load time when they are linkage editted into the program

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

Processes use I/O alot. In any other SCHEDULING method, time would be lost whenever I/O happened. In MLFQS, other processes run when I/O is happening.

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

1. process uses up its time slice before giving it up → it gets bumped down a queue
2. process gives up the CPU before time slice ends → it gets bumped up a queue

(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 it is a long process and it is in a high priority queue (short timeslices) it would keep getting preempted (and possibly take time away from a higher priority process)

(c2) If it is a process that frequently gives up the CPU, to do a lot of I/O, it would waste its time slice by waiting for the I/O to finish

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)?

possibility of internal fragmentation

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

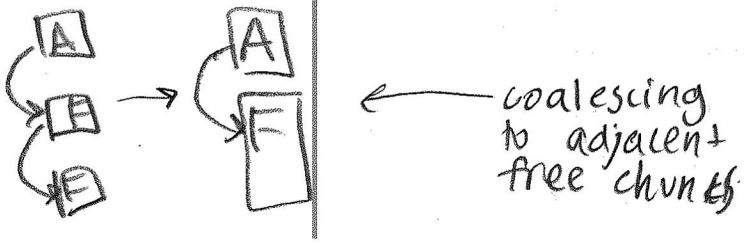
there can be different fixed sizes that are allocated: for example 1, 10, 100 bytes, the closest to what was requested would be returned. This helps reduce the amount of internal fragmentation

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

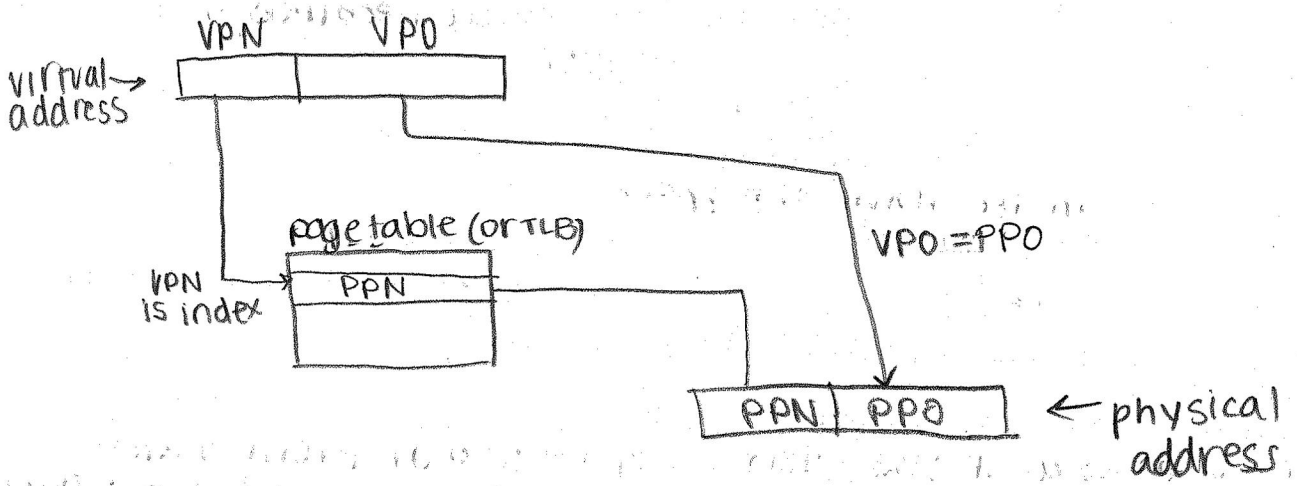
External fragmentation might arise now, because allocated segments are different sizes

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

contain a list of all memory that is allocated/free and coalesce when two adjacent nodes of the list are free → reduces external fragmentation



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



VPN : virtual page number
VPO : virtual page offset
PPN : physical page number
PPO : physical page offset

(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. valid bit : shows whether this address is a valid memory address
   0 : no it is not valid → page fault ; bring it into memory
   1 : yes it is valid ; already in memory

2. dirty bit : shows whether or not it has been written to but not saved to disk
   0 : clean
   1 : dirty : make sure to write the information back to disk before swapping out

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

TLB are on the CPU and are a lot faster than accessing page tables in memory. TLBs are basically caches of the page table. It takes many cycles to access memory to get to the page table, but the TLB speeds that part of address translation up. (If it is large enough, so no thrashing)

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

(a) "indeterminate"

when the output cannot be determined, because you don't know when things are going to happen

"The order of threads is ~~non~~ indeterminate"

these are switched

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

occuring at an unknown time/pace

"The output of this code is non deterministic"

(c) "race condition"

a condition that code may have in a critical section where the output will change if the events happen in a different order ← is not a part of code

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

the section of code in which the out put is timing dependent. The output might change if the times at which events occur change. ← is part of code

(e) "atomicity"

the instruction happens without interruption

↳ this is guaranteed

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. progress, fairness, correctness, performance

(a) correctness = provides mutual exclusion ALWAYS

Interupt disabling does not provide mutual exclusion for multi-core processes and is therefore not correct

(b) fairness = each thread has an equal chance to get the resource and is guaranteed to get it without having an unbounded wait time.

spinlocks that have a lot of contention are not fair. A waiting thread could have an unbounded waiting time if other threads keep getting the resource before it

(c) performance = the CPU is used efficiently, and cycles are not wasted

spinlocks have poor performance because if a thread that is waiting for the unavailable resource is scheduled, it will just spin for the entirety of the clock cycle, and nothing will happen (wasting the CPU)

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.

if the waiter gets signaled by a semaphore it is GUARANTEED to have the resource (doesn't need to check before using the resource)

if the waiter gets signaled by a conditional variable, it is not guaranteed to have the resource and needs to check again before using it

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

sempost semwait

conditional variables: wake up all the threads waiting on that signal so by the time one thread gets the signal and wakes up, another thread might have the resource

semaphores: sempost counts how many threads can use the resource and semwait decrements the count before the resource is used. when semwait returns, the count is already decremented, so you're guaranteed to have the resource

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

in a producer/consumer situation: conditional variables need both a mutex and a conditional variable. semaphores just needs a semaphore

when the producer has a character → sempost (increases the count of characters)

when the consumer says semwait → decrements the count; if the count > 0 the consumer processes the character

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

any other situation where you don't need a counter

- need to get notified when a specific variable is a certain value: conditionals would be better because there is no counting involved

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

the programmer has to keep track of what should be locked with advisory locking. with enforced locking, the lock is aquired and released automatically whenever that resource is used.

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

enforced locking creates a large overhead. If you had a resource that only needed to be locked sometimes, you don't need enforced locking, and the programmer can manually lock/unlock the resource.

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

- "class" programming where the resource is accessed through functions that do the lock/unlocking.
  - ↳ not allowed to access the resource directly

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

they address problems where a full search of something is required.

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

it approximates a full search with a guess pointer that points to the last object checked on the previous search. it then returns the first object it finds that matches the search criteria.

(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.

shelf 1

shelf 2

shelf 3

shelf 4

1 - when you buy a new item, put it on shelf 4 (buy does not mean use).

2 - when you use an item, put it on the shelf above the shelf it was on. If it was on shelf 1, keep it there

3 - to get the approximate LRU, (lets say every two days), pick an item from the lowest shelf. move everything to the lowest shelf.

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

after 2 days, every thing on shelf1 was used most recently and everything on shelf 4 was not used

(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.

(2) changes to

when you use an item, if it is less than three days from expiring move it down a shelf, else move it up a shelf

this change makes the bottom shelf LRU or almost expired food.