CS111 Midterm Exam

Terry (yunong) Ye

TOTAL POINTS

87.5 / 110

QUESTION 1

Principles 10 pts

1.1 Define Info Hiding 2 / 2

✓ **- 0 pts Correct**

1.2 Value of Info Hiding 1.5 / 3

✓ **- 1.5 pts miss "give greater flexibility to change the internal details"**

1.3 Define Modularity 1 / 2

✓ **- 1 pts the answer is not clear enough/not totally correct**

1.4 Good Modularity 0 / 3

✓ **- 3 pts cannot understand the logic**

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 0 / 2

✓ **- 2 pts wrong**

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 1 / 3

✓ **- 2 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 2 / 2

✓ **- 0 pts Correct**

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 3 / 3

✓ **- 0 pts Correct**

5.3 problem with special sub-pools 2 / 2

✓ **- 0 pts Correct**

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

indeterminate concerns the result, it doesn't 'run'

7.2 non-deterministic 0 / 2

✓ **- 2 pts** incorrect

7.3 race condition 2 / 2

✓ **- 0 pts** Correct

7.4 critical secstion 2 / 2

✓ **- 0 pts** Correct

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

✓ **- 1 pts** In a counting semaphore, the count can represent resource/completion availability, and a successful P grants the resource to the recipient. Mandatory queuing ensures no other process can take that resource.

9.3 when choose semaphores 2 / 2

✓ **- 0 pts** Correct

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 4 / 4

✓ **- 0 pts** Correct

10.3 requirement for enforced 1 / 2

✓ **- 1 pts** clients cannot directly access the protected object without going through methods that enforce locking

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

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

CS111 Midterm 05/03/2018

Name Terry Yunong Ye

Student ID # 004757414

Seat Row 6 Seat Col 4 Exam # 42

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 means the implementation detail or how it is implemented is hidden from others & just what it does

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

① esmplicity

② encapsulate complexity

③ support interface

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

Modularity is that if we can successfully use the parts without knowing implementation details of it.

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

A specific interrupt handler is a module that will handle the specific interrupt although don't know how it will handle specifically, so it just uses it when the corresponding interrupt happens.

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

Application Binary Interface.

(b) Define the term?

ABI is a mapping between API and ISA that helps define the register to use, linkage convention etc for application

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

You can just run ABI to see if it is correct, for API has to compile, debug ... etc

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

Yes when the Instruction set Architecture is different for two OS.

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

easy to implement (don't need a smart linkage editor)

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

↓ mechanism

There will be an entry table that stores the address of object module and incorporates the linkage editor will see which is needed and use the table to find the corresponding one

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

① there is only 1 copy of it in whole segment instead of 1 copy in every program that uses it.

② can update the library and the program will automatically use the updated one

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

The behavior of processes might change.

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

① after the allocated time slice finished, the process would be put into a level lower than the previous one.

② after a time period, the priority boost will move the process to uppermost priority 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 on a higher priority queue than supposed, then there will be convoy that multiple more important/interactive process gets behind it.

(c2) if it is on a lower priority queue than supposed, then there will be starvation that the process gets behind multiple interactive process and not scheduled for 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

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

special pools are fixed-size region with sizes that are more popular it will reduce the internal fragmention because when a requested size that is in special pools comes in, the special pool buffer will be allocated which is a perfect match without internal fragmentation

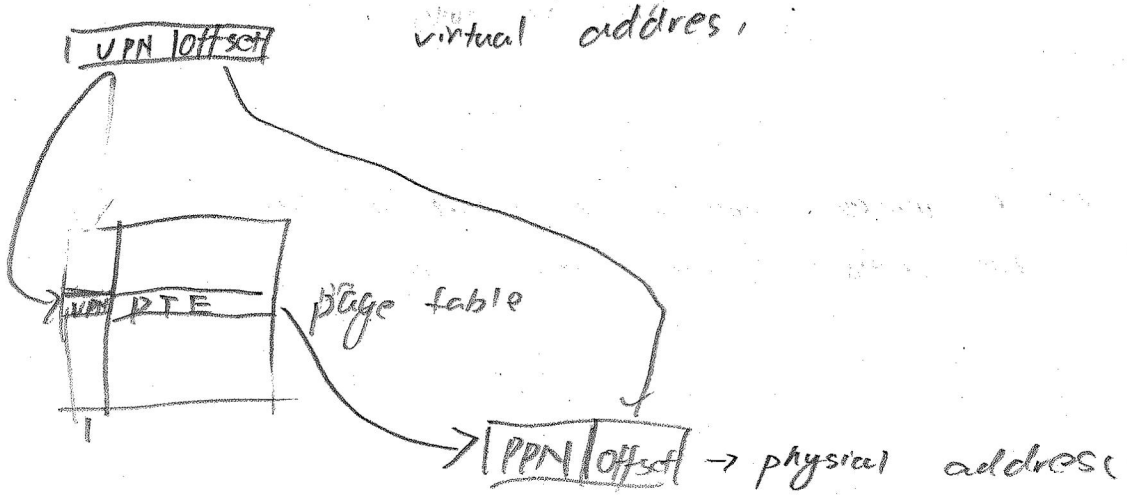
(c) What new problem is likely to arise when we create such pools?

The buffers may not be used and wasted. if less matching size requests comes in

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

Slab allocation which is designed for popular data structure and reinitiated less.

6: (a) Draw a diagram of a paging MMU, and illustrating how it translates a virtual address into a physical 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 reference bit : one bit that tells whether the page has been referenced or not . this is useful when determining (ex. LRU clock algorithm) which page to swap out.

② dirty bit . one bit that tells whether the page has been modified or not

useful when laundering

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

Because it is too expensive to have a page table for whole virtual address space in main memory. (too much memory space required)

and it would be too slow if the page table is put in secondary memory and for every translation have to go there to search.

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

(a) "indeterminate"

result can be different every time it runs.

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

non-deterministic process involves non-deterministic execution like message sent / I/O but the result will not be different.

(c) "race condition"

(result depend on timing)

parallel threads modify a shared resource at the same time.

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

(correctness depend on timing)

part of code that involve modification of shared variable

(racing condition is the scenario that critical section wants to prevent)

(e) "atomicity"

(1) if it can prevent race condition
(2) mutual exclusion
(3) atomic operation (all-or-nothing)

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)

correctness: if it can successfully or correctly provides mutual exclusion to critical section

(ex. disabled interrupt cannot work for multiprocessor because it only disables interrupt in one processor)

(b)

fairness: if every thread waiting for lock has a fair chance of acquiring the lock.

(ex. spin lock is bad for fairness because there might be thread that never gets the lock)

(c)

efficiency: if the locking mechanism will affect performance much.

(ex. spin lock is bad for efficiency because spinning keeps checking the same condition and wastes the cpu cycle.

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 notification is still valid. (if condition is still true) semaphore waiter can always assume the notification is valid and start the following operation

condition variable waiter has to check if condition is still true (while loop)

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

Because semaphore only wakes up 1 thread (first one) in its sem_post function. So no worry of other threads operating first.

Condition variable can have spurious waking where the thread can be waked up with others or by other process and find out the condition is not true and go back to waiting

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

when there are multiple processes and resources as it eliminates the overhead of spurious waking.

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

when there are muttiple threads waited for a condition the so condition variable can notify all threads at once.

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

absolutely prevent other operations on the locked object.

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

if a programmer wants to decide when the thing should be locked and if it is a resource that might not be operated by others

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

Privileged access to prevent other operation. and without the user.

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

global LRU and working set LRU.

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

a referenced bit that tells whether the page has been referenced.

~~N~~ pointer which points to the last swapped page frame.

(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.

start from the place where the last thing is put in,
go a fixed order (ex. right → down)
if a thing is no star, continue, put a star on it.
if a thing is starred, remove the thing
if it reaches the bottom, start from the start (ex. top-left) again, ... time

(put in a new thing with star and remove star when using it

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

because if it is not used since last time scanning,
it is probably not used for a long time.

(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.

target time = d0 days before expire

c) remove until the days is less than target time.

but still implements