

# CS110 - Principles of Computer Systems

## Midterm Exam Solution

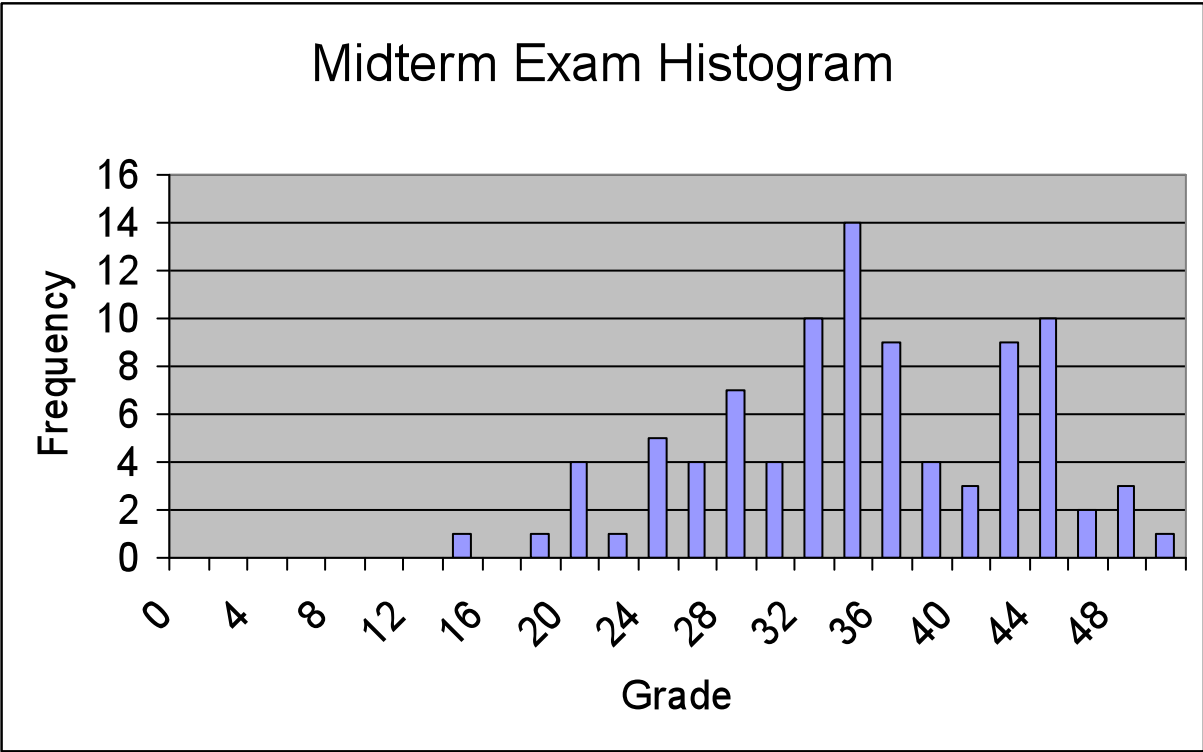
(Total time = 50 minutes, Total Points = 50)

Name: (please print) \_\_\_\_\_

In recognition of and in the spirit of the Stanford University Honor Code, I certify that I will neither give nor receive unpermitted aid on this exam.

Signature: \_\_\_\_\_

Mean: 33.9, Median: 33, Std dev: 7.8, Min: 14, Max: 49



1. (12 points) Recently Apple Computer changed their iPhone Developer License Agreement (an agreement all iPhone applications must abide by) to include the following section:

*3.3.1 - Applications may only use Documented APIs in the manner prescribed by Apple and must not use or call any private APIs. Applications must be originally written in Objective-C, C, C++, or JavaScript as executed by the iPhone OS WebKit engine, and only code written in C, C++, and Objective-C may compile and directly link against the Documented APIs (e.g., Applications that link to Documented APIs through an intermediary translation or compatibility layer or tool are prohibited).*

For each of the following concepts from class, describe how the above section relates to the concept. If there is no obvious relationship simply write none.

- a. Naming and naming conflicts
- b. Interpreters, memory, and communication links
- c. Layers and Layer bypass
- d. Virtualization – Emulation

*Solution:*

*a. Naming and naming conflicts: None.*

*b. Interpreters, memory, and communication links: The reason this section was added to the developer agreement is that developers were writing applications in other languages and then running an interpreter to convert those languages to one of the supported languages. It is this usage of an interpreter that is forbidden by this document.*

*c. Layers and Layer Bypass: The Flash-to-iPhone translator is a layer on top of the public API. Layer Bypass: The usage of private APIs bypasses the public API layer, and is therefore disallowed.*

*d. Emulation/Virtualization: The Flash-to-iPhone translator emulates the iPhone public API in another language (i.e. Flash). This is in the document as "intermediary translation". The document disallows this.*

*This question was 12 points, so each part was 3 points. Points were awarded as follows:*

*3 - Completely right*

*2 - Mostly right*

*1 - "Grain of truth" in answer*

*0 - Didn't answer, or answer is either completely wrong or irrelevant to the question.*

... more space available on next page (don't feel compelled to use it)....

Space for answering question 1

2. (6 points) Explain the tradeoff between generality and specialization in computer system design. Give an example that illustrates the tradeoff.

*Solution:*

*Generality makes a system useful for a larger target audience but the system may then have sub-optimal performance on each task; specialization means a system does well on some specific tasks but the user community is small. It can also be argued that generality results in more complexity and specialized systems have simpler designs.*

*Two good illustrative examples are Microsoft Word vs. PowerPoint, CPU vs. GPU.*

*4 points were given for a complete explanation and 2 points for a reasonable example.*

3. (8 points) Answer the following two questions about naming systems.
- Recall that name resolution is done within a *naming context*. Describe two ways name resolvers determine the correct naming context to use.
  - Is there a relationship between *user-friendly names* and *overloaded names*? If so, explain the relationship.

*Solution:*

*a. (4 pts.) Three ways listed in the handout of Lecture 4 on naming: 1) embedded in the name itself; 2) from the environment; and 3) hard-coded in the resolver. The handout also describes each of these ways in detail.*

*b. (4 pts.) Yes, there is a relationship. User-friendly names are a subset of overloaded names. User-friendly names are overloaded but overloaded names are not always user-friendly. Some examples are listed in the handouts.*

4. (8 points) Modern operating systems divide a program running in an address space into segments. Common segment types include (1) the code segment, containing the instructions that are executed, (2) the heap segment, used by dynamic memory allocation (e.g. malloc/new), and (3) the stack segment, used to hold the runtime stack of the process. Explain why a process' heap segment can have fragmentation problems while its stack and code segments codes don't.

*Solution:*

*(2pts.) Code: Not changing, static. The code is loaded in before the program begins running, therefore, since it is not changing there can't be fragmentation.*

*(2pts) Stack:*

*-Once out of scope, the variables' memory is freed -Memory is allocated in an organized/layered manner, and the amount is known ahead of time. Also it is freed in the reverse order that it is allocated. (Must mention both)*

*(2pts) Heap:*

*-Blocks are spread out, rather than contiguous. Memory is not necessarily allocated next to each other.*

*-Allocated chunks/blocks can be of different sizes. This means there is no uniform size and different regions of needed space, and free space won't be the same size and won't necessarily fit.*

*-No fixed order to which the blocks are allocated in general. Therefore, no guarantee as to whether or not a freed space may be needed by some other region later on.*

*-User manually requests blocks, so no foreseeable way to anticipate this and guarantee an arrangement of blocks that works out.*

5. (8 points) Answer the following two-part question.
- Explain why on modern computers we wouldn't want to run a small (< 25 instruction) subroutine using enforced modularity.
  - Given an example scenario of where we would want to use enforced modularity on a small subroutine.

*Solution:*

*a) The overhead of running with enforced modularity may be high relative to the code (25 instructions). If the ratio of overhead to actual run-time is higher for smaller subroutines, then running many subroutines will be not efficient, and many resources will be devoted to the overhead and that may overshadow the computation of the run-time.*

*b) It might still be prudent to use enforced modularity for security reasons. If you are running untrusted code, and don't want such code to corrupt things that are running natively, enforced modularity would be helpful. Another reason one could mention to get credit would be to limit the propagation of bugs and/or crashing, with significant explanation (mentioning propagation of bugs without sufficient explanation might result in partial credit).*

6. (8 points) You see the following statement on the CS110 discussion forum:

Although *non-blocking synchronization* doesn't have *deadlock* problems it can have *livelock* problems under high contention.

Does this statement make any sense? Explain your answer.

*Livelock, as defined in the book, is "An undesirable interaction among a group of threads in which each thread begins a sequence of actions, discovers that it cannot complete the sequence because actions of other threads have interfered, and begins again, endlessly." Another good definition would be that threads can change state without making progress.*

*We discussed atomic CAS in class as a way of doing nonblocking synchronization, so answers that talked about how a system using CAS to implement nonblocking synchronization might have one or more threads that cannot progress were preferred.*

*+4 Yes, the statement makes sense. Partial credit for these points was possible if a student didn't commit to 'yes' or had a less than convincing or info-dump-style explanation.*

*+4 Explanation included a good understanding of the differences between livelock and deadlock, and how livelock might affect a system using nonblocking synchronization. Note that the question references nonblocking synchronization, so any description of livelock problems caused by blocking synchronization were worth zero.*

7. (EXTRA CREDIT 5 points) Restate the Apple license section from Question 1 using concepts from the class to be more understandable to a technical person of what is allowed in an iPhone application

*Solution:*

*The agreement is divided into two sections, the first mostly dealing with layering and layer bypass, the second dealing with either emulation or interpreters.*

*+2 for translating each of the part of the license and using a term mentioned in class that applied appropriately.*

*+1 if the translation is correct, concise, and sticks to the material in the license without "info dumping".*

*Comments about Apple being draconian or working to specifically block out Adobe were amusing, but not worth extra credit.*