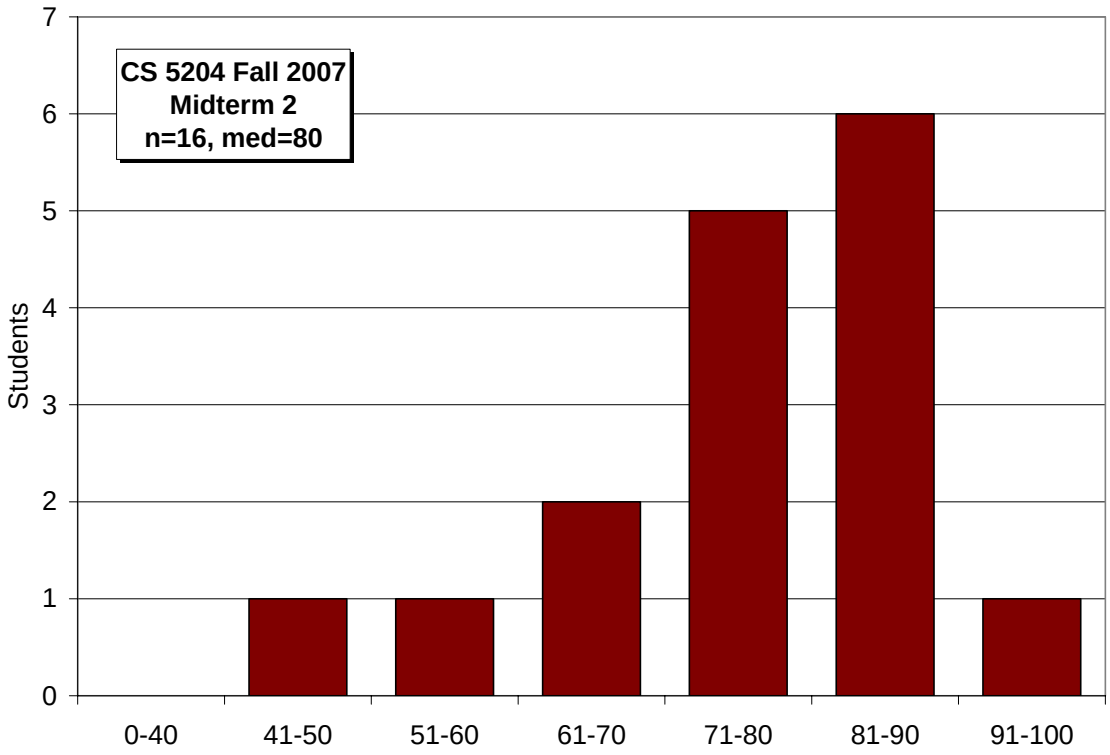
CS 5204 Midterm 2 Solutions

As announced, I counted the best 3 out of 4 scores, then multiplied by 4/3. The answers are shown below.

Problem	1	2	3	4	Sum	**Adjusted**
Avg	**13.9**	**18.9**	**12.1**	**16.9**	**61.9**	75.9
StdDev	**5.6**	**6.4**	**10.9**	**7.9**	**12.5**	12.3
Median	**15**	**20.5**	**15.5**	**20**	**61.5**	80
Min	**0**	**0**	**0**	**0**	**38**	44
Max	**20**	**25**	**25**	**25**	**90**	93
Possible	**25**	**25**	**25**	**25**	**100**	100



Solutions are shown in this style.
For 1c), 3c), and 4c) I assigned scores based on my judgment regarding how well-reasoned, complete, and balanced I viewed your discussion. Do not view my comments made in this document as a solution key.

1. Soft Updates vs Journaling (25 pts)

a) (5 pts) Why does the paper by Seltzer et al compare soft updates to both *synchronous* and *asynchronous* journaling approaches? Explain what each of the versions of the journaling approach accomplishes and when you would use which.

Writing a journal synchronously provides the same consistency guarantees as the synchronous metadata writes FFS itself does, and will also significantly slow down metadata operations (though it does provide faster recovery). However, synchronous writes are not required to keep file system metadata in an acceptably consistent and recoverable state. For normal file system operation, the log is written asynchronously (while still ensuring that the log reaches the disk before the metadata blocks to which it refers.)

b) (5 pts) Linux's ext2 file system, a Unix-style multilevel index based file system, does not use journaling, it uses write-behind caching, all metadata writes are asynchronous, and it does not keep track of any ordering dependencies. Mention 2 immediate implications of this design.

Implication #1: it's fast. Implication #2: it's unsafe and may encounter unrepairable failures, or the repair may lead to unacceptable outcomes. Read the eXplode paper for anecdotal evidence. (Aside: Seltzer et al call such an approach "not usable for a production system" – yet Linux users had to live with the associated risks for many years, until the arrival of ext3.)

c) (15 pts) *Explain* and *critique* the methodology used in the Soft Updates/Journaling paper to compare the two approaches quantitatively. Focus on the key points only.

Explanation:
To identify which of the two approaches to keeping metadata consistent is superior, the researchers used a methodology that eliminated those variables in which they were not interested by basing their investigation on the same file system (FFS). As a baseline, they created a version (asynchronous FFS) that represents one end of the consistency/performance axis: maximum performance, and no metadata consistency guarantees. Based on that, they were able to quantify how much performance each approach trades in to provide consistency. To ensure sufficient breadth, the researchers used micro- and macro benchmarks. To ensure fairness, they also allowed for different journaling configurations that represented the options users of journaling systems have.

Critique: (This is my personal view, yours may vary if you can justify it.)
Overall, the investigators' methodology is sound, convincing, and achieves the goal of comparing metadata consistency approaches. Its limitations are minor: one could mention here that only a single file system implementation, a single write-ordering model, and a single approach to writing the journal was evaluated.

2. Distributed Systems & RPC (25 pts)

a) (5 pts) Remote objects are endpoints for the invocation of a set of remote methods, just like local objects can be used to invoke local methods.

Suppose an RPC system provides the ability to pass references to remote objects as arguments to an RPC call. Briefly describe what an RPC runtime system has to do to support such functionality.

In order to pass remote objects by reference, the caller's runtime has pass along to the callee's runtime information that allows the callee to create a client-stub for invoking the passed object, and to bind to the passed object – for instance, an object identifier describing the object's location/port could be passed, allowing subsequent communication with the machine housing the object's implementation.

b) (10 pts) The RPC system as designed by Birrell made the decision to not let remote functions time-out (discussed in section 1.4), because doing so would "needlessly complicate the programmer's world." Explain what complication(s) could arise, and discuss a method of avoiding it/them.

If a client timed out without knowing that the server crashed, it could be that the server already processed the request and executed the call when the client timed out. In this situation, programmers would need to either use idempotent call semantics, or employ a technique that filters out repeated calls, such as sequence numbers or time stamps.

c) (10 pts) Suppose you implement a distributed bulletin board application for an asynchronous distributed system where users can post messages and see which messages were posted in response. When designing this application, would you choose a variant of logical clocks, or would you use physical clocks (and require clock synchronization)? Justify your answer.

In an application like this, one would probably want to display messages based on potential causality, as expressed in a happens-before relationship: user B posted message n after receiving message m from user A. Such a relationship can be established using logical clocks (such as vector clocks which can accurately determine a happens-before relationship between messages).

3. Nooks and JIFL (25 pts)

a) (5 pts) Suppose the designers of Nooks had had a system such as JIFL at their disposal when they built Nooks. Describe, in sufficient detail, how JIFL could have avoided the use of separate hardware protection domains for memory protection.

Instead of placing each extension in its own hardware protection domain, they could have instrumented each load and store instruction to perform a check whether the attempted access is allowed, and jump to an exception handler if it is not. (This technique is known as software fault isolation (SFI). See Wahbe's 1993 paper.)

b) (5 pts) Nooks states that it does not target malicious extensions, but buggy ones. Again assuming that JIFL-like just-in-time compilation had been available, what additional safety guarantees could Nooks provide?

The reason Nooks can't target malicious extensions is that the extension still execute in kernel mode, so they have access to privileged instructions. This is not a problem for JIFL, which could simply translate those privileged instructions to calls to routine that perform safety checks, or even inline those checks – very much like what VMWare ESX does for its guests.

c) (15 pts) In your opinion, are the implementation effort and complexity, as well as the measured performance overhead, worth the benefits that can be – or could be – reaped from applying just-in-time compilation to a kernel? Justify your opinion with facts and/or examples and consider possible alternatives.

(The following paragraph reflects my opinion as of the time of this writing:)
Good arguments in favor are the fact that just-in-time compilation has become better understood in recent years, lowering the implementation complexity, and that at least for fine-grained instrumentation tasks it clearly outperforms competing approaches such as Kprobes.
However, skepticism may also be appropriate – after all, instrumentation is only a method; real benefits will arise from building tools and analyses exploiting this method. Despite having implemented the method, at least this initial paper shows only limited impact as far as actual insights or possible improvements that can be gained.

4. eXplode and Rx (25 pts)

a) (5 pts) Both Rx and eXplode exploit checkpointing and replaying techniques. Outline 2 core differences in how these techniques are applied in these respective systems.

#1: Unlike Rx, which intentionally varies the execution environment when replaying from a check point, eXplode must ensure deterministic replay by choosing exactly the same sequence of choices.
#2: eXplode, being a model checker, must attempt to remember all states it has visited, by recording the choices made from checkpoint to reach the state. Rx, on the other hand, only keeps one checkpoint and only records replay choices as an optimization for learning.

b) (5 pts) eXplode checkers can optionally provide a method by which domain-specific knowledge can be used to express the equivalency of two different states, even if different sequences of choice points were taken to reach those states. What is the motivation for this function? Give an example of such equivalent states and sequences of choice points.

The motivation is to recognize states that should be consider identical, and thus explored only once, even if these states were reached differently. For example, creating two files A and B in the order A,B or in the order B,A should lead to an identical state.

c) (15 pts) Choose either Rx or eXplode for this question. Imagine that a group of PhD-level researchers would invest 10 man years in the system you chose. What additional impact and/or applications could you envision for the system you chose? In your opinion, where do you see intrinsic limits of the system (e.g., limits that cannot be overcome with significant investment in engineering)?

This is an open-ended question, and I will share my opinion here. To address additional impact, you could name some of the applications that the developers of Rx and eXplode list in their future work section, or mention your own. Please refer to the papers.
In my opinion, for either approach, there is a serious potential for intrinsic limitations that may affect their ultimate potential: in Rx's case, the number of errors that can be treated as allergies may be bounded – note, for instance, that the developers didn't see a single dangling pointer bug in their server apps, and had to manufacture one. In the future, this will become even more so with the development of tools that can eliminate those bugs during development.
In eXplode's case, the tools seems narrowly focused on the specifics of bugs that arise from mismanaging buffer cache write orders; it's difficult to see wider application of this approach since many specific properties are exploited (for instance, the fact that a filesystem's state is pristine on remount, and the fact that checking is based around crash disks.)