CS240
Operating Systems
Midterm Exam Review – Feb. 8, 2013

Your Name: _____

SUNet ID: _____ @stanford.edu

Check if you would like the exam returned to you via SCPD: ☐

In accordance with both the letter and the spirit of the Stanford Honor Code, I neither received nor provided any assistance on this exam.

Signature: _____

- The exam has 4 questions totaling 100 points.
- You have 60 minutes to complete them.
- If taken remotely, please scan the completed exam and email it to: `cs240-staff@scs.stanford.edu`.
- All questions require you to justify your answer to receive full credit, even multiple choice questions for which you circle the correct answer(s).
- Keep your answers concise. We will deduct points for a correct answer that also includes incorrect or irrelevant information.

1	/20
2	/30
3	/30
4	/20
Total	/100

1. [20 points]:

The Dune system allows a process to enter "Dune mode," in which it has safe access to privileged hardware features, including ring protection, page tables, tagged TLBs, interrupt vector, exception vector, and system call vector.

Safe access to the system call vector allows an application to do system call interposition. As a result, it can, for example, disable the `open` system call, re-define how `openat` behaves, and provide new system calls.

1. Can you use system call interposition to implement the sandboxing functionality provided by Capsicum's capability mode? If yes, describe how you would address the core issues Capsicum addresses. If no, describe why not.
2. Suppose that the correct answer above is yes. What do you expect the performance of a system call interposition Capsicum to be compared to the existing approach.

2. [30 points]:

The Capsicum authors believe that Mandatory Access Control (MAC) has several downfalls when compared to capability systems. Yet, systems such as Jif, HiStar and Hails enforce MAC.

Considering the following claims made by Capsicum:

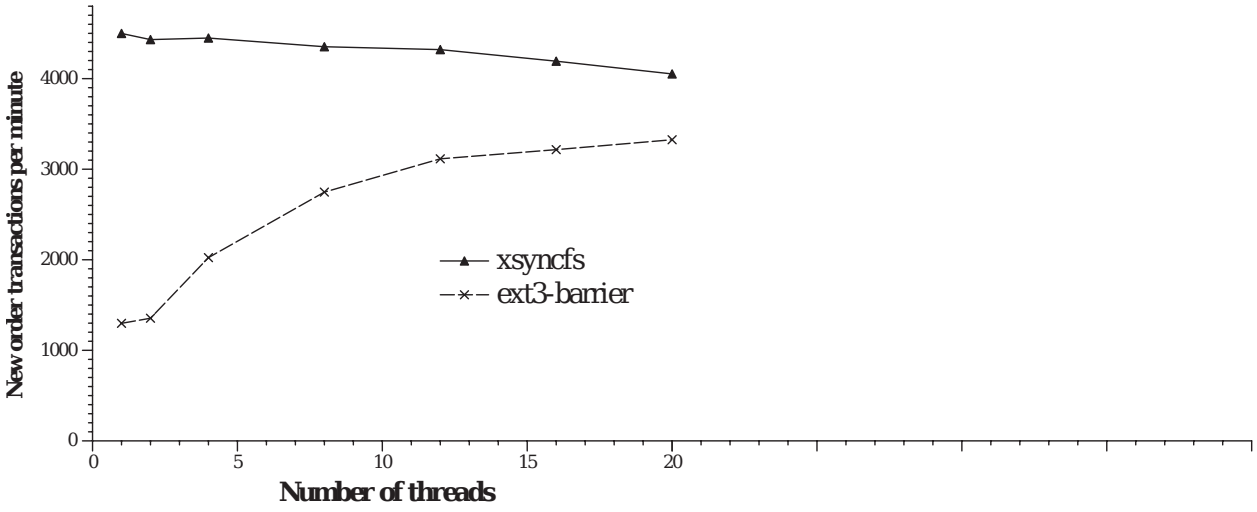
1. MAC systems perform "hundreds of access control checks."
2. In "all MAC systems, security policy must be expressed separately from code. This can lead to inconsistencies and vulnerabilities."
3. In systems like Capsicum "bugs can't arise due to potential inconsistency between policy and code."

In the context of Jif, HiStar and Hails:

1. Is the 1st claim true simply because MAC system designers don't know how to build systems or can HiStar/Hails fundamentally allow for security policies that Capsicum cannot.
2. Different from the 2nd claim, Hails argues that policy should be expressed separately from code. What some the trade-offs between the two different design approaches?
3. Can inconsistencies and vulnerabilities arise in Hails because of this code-policy separation?
4. Do the 2nd and 3rd claims hold true for Jif and HiStar?
5. Will the 3rd claim necessarily hold true when considering MAC policies? What kind of policy are the Capsicum author referring to?
6. What makes Jif, HiStar and Hails different from MAC systems such as SELinux and Seatbelt described in the Capsicum paper?

3. [30 points]:

1. xsyncfs is fast and provides strong durability guarantees. Why is it not used everywhere?
2. Can you use patches to guarantee external synchrony? If yes, how? If no, why not?
3. What do you expect the graph will look like as we increase the number of threads to 40? Explain.



4. Suppose you had a patchgroup-like abstraction that interacts with xsyncfs and are using memory-safe language like Java. How can you improve the SPECweb99 benchmark?
5. Can you implement external synchrony by simply overriding `libc`? If yes, explain how you would do it. If not, explain why not or guarantees you need to relax.

4. **[20 points]:**

How would Singularity's IPC mechanism be affected if we replaced Sing# with C?