Some study questions for the first midterm

CPS 210, Fall 2012
Department of Computer Science
Duke University

September 26, 2012

Abstraction

One of the choices for our "new" OS course was to introduce some general concepts and terminology at the start, e.g., via the Garlan and Shaw paper, and also as a way to unify the various platforms we are studying: Unix/Linux, Android, and cloud hosting services.

So I have introduced some rather abstract terms: *context, entity, identity, component, connector, channel, label, attribute, event, principal, reference, subject, object, guard*. These terms are "abstract"because it is not clear exactly what they mean. That might frustrate systems students, who tend to want to learn things that are concrete, and that they know they can use (and explain on an exam). On the other hand, abstraction is a tool to organize knowledge and separate underlying concepts from details. We can't say precisely what the terms mean because they have slightly different meanings in the different systems. But they also capture structure that is common across these systems, and give us a common vocabulary to talk about them. Abstraction is a double-edged sword.

These first few weeks we have tried to lay the groundwork to go into more depth on these concepts through the semester. So the treatment thus far may raise more questions than it answers. This is especially true for the cloud hosting topics, for which we have just scratched the surface.

Here are some thought questions intended to exercise your familiarity with the underlying concepts and terms, and sharpen understanding of the similarities and differences among systems designed with varying goals and assumptions. Some of these questions open topics that we will discuss in more detail later.

1 Abstractions

1. Outline some ways that the six key Garlan/Shaw concepts appear in the systems we have studied. Give examples of instances of each concept. What new features or properties or structure does each instance add to the concept?

2. Garlan and Shaw present their six concepts as structuring patterns for use within a given software program. In contrast, we consider their use in OS abstractions for composing/combining multiple programs or applications. What features or properties or structure do these OS instances add to the concepts?

3. For example, what is an event (GS3)? Why is it useful for components to receive event notifications separately from information queried explicitly through an API? How are events used in the Unix process model, inside the kernel, and in Android? What information is propagated by means of events in these systems?

4. What is the relationship between a *context* and an *entity*? (See discussion on the term "entity" in the *Notes on Security.*) Can a context act as multiple entities? Can an entity span multiple contexts?

5. Can a context ever cross node boundaries?

6. Can a component ever cross context boundaries?

2 Unix/Linux

1. Unix/Linux is written in C. C has pointers and is unsafe. Operating systems should be safe. So why use C? What are the benefits or suitability of C for writing system software?

2. We discussed several examples of *channel.* In general, an endpoint of a bi-directional channel can query the system for security attributes of the other endpoint. (E.g., understand how this works in Android. Also, in a network setting, one end of a secure socket connection can obtain any certificates or other authentication info presented by the other side.) Here's the question: Unix pipes are channels that do not have this property: they provide no means to query who is on the other side. Why? Would that feature be useful? Do we sacrifice security by not having it?

3. Unix files are reference-counted. What events or actions change the reference count? What does the system do when the count goes to zero?

4. The Unix *fork* system call clones a process. What does that mean? What state associated with the process is copied? What happens when the cloned child exits?

5. A Unix kernel knows how much CPU time a process consumes. We used the *getrusage* system call to tell us how long your heap managers take to run through a stream of requests. How does the kernel know?

6. What is a context switch in Unix? (There are several different kinds.)

7. What is a process group? Why are process groups useful?

8. You read the original Unix paper. What are the central abstractions in Unix? How do they relate to the software architecture concepts in the G&S paper?

3 Android

1. Android runs apps in protected contexts. Several mechanisms are used to enforce isolation/sandboxing of Android contexts. Discuss.

2. One of the four types of Android components is a *service.* How is an Android service different from a network service, in terms of how it behaves or might behave? (Other than the obvious difference that the network service interacts with its clients over a network.)

3. An Android app can declare permissions required of any client of each service component. Different components in the app can have different permission requirements. However, the permissions held by an app are defined for the entire app and all of its components, once, at install time. Why the difference? Why maintain held permissions at app grain and required permissions at component grain?

4. When an Android app fails, a system service (I think the Activity Manager Service) pops up a dialog box describing the failure and asking the user if you want to send a bug report. How does the Android system know or find out that the app has failed? Can a user send fake reports about app failures, even if the app has not failed? How might Android defend against malicious bug reports?

5. Android requires that all installed apps are digitally signed under a keypair with an accompanying certificate. But the keypairs used to sign apps are not necessarily bound to an identity (e.g., the certificates may be "self-signed", in which a developer signs its own certificate.) Why is signing apps useful, even when Android does not know who signed them?

6. Active components in an Android app are reference-counted. Why? What events or actions change the reference count? What happens when it goes to zero?

4 Linux and Android

1. Android apps are (generally) written in Java, which is a type-safe language. A Java program cannot reference memory unless the program holds a valid reference to an object or class occupying that memory. Does that make protection by virtual address spaces unnecessary? Why doesn't Android run all apps in a single process/context?

2. "From the point of view of the Linux kernel an Android app is just data." Discuss.

3. What does the Linux kernel "know" of Android components? Can the kernel determine if a component has failed? Does it know how many components are active on the system?

4. The Linux kernel has little visibility into what is happening inside a program, or the program's internal structure. It knows only how much CPU time the process is consuming and what faults and traps it takes. In contrast, Android has visibility at the granularity of components. That requires a lot of new system machinery to implement the component abstractions. What or how does Android gain in terms of efficiency? How is this choice suited to the needs of a smartphone?

5 Cloud

1. We have discussed three "as a service" models for cloud hosting infrastructure, platforms, and application services. Often these services are offered in a commercial "public" cloud. In this case, what we mean by "as a service" is that a provider and its customers enter into economic transactions to provide the service for a price. It is the first OS model we have talked about in which money changes hands. The transaction must be economically attractive to both parties: the provider makes a profit, and the customer obtains the service at lower cost than it otherwise could. How does this happen? Why can't the customer provide the service for itself as cheaply as the provider can provide it?

2. IaaS cloud hosting is (in essence) virtual machines for rent, e.g., following the Amazon EC2 model. It is more flexible than a PaaS platform in that it permits the customer to install its own operating system. Why would the customer want to bother with that?

3. A local startup called rPath was based on the idea that virtual appliances are attractive to application software vendors as a means to package their product for delivery to customers. Why might that model be more appealing than selling software that the customer can install on their own operating system, without bothering to create a virtual machine for the virtual appliance?

4. One might argue that the virtual appliance model has been less successful as a commercial sales channel than originally envisioned. Some software companies prefer to host the software themselves and offer it to customers as an application service (SaaS), rather than sell the software as a virtual appliance and let the customers deploy it on their own infrastructure, or on their own cloud instances. What advantage does this choice offer to software vendors?

5. EC2 and other IaaS providers offer a service to create cloud storage volumes and attach them to virtual machines. A cloud storage volume is essentially a sequence of blocks that behaves like a disk. We will discuss the implementation of network storage volumes later. My question now is: how might your

virtual machine use a cloud storage volume? How should it appear to, say, processes running on a guest Unix system in the virtual machine?

6. In a PaaS model, the cloud provider accepts programs from customers and runs them on its own operating system, e.g., in processes on a Linux server. Generally, letting someone else choose what code runs on your system is unsafe. Is PaaS unsafe? How can a PaaS provider protect itself and also isolate multiple customers sharing the same platform?

7. One advantage of the cloud model is elastic scaling: a hosted application service can grow and shrink by acquiring and releasing cloud instances (e.g., VM nodes). It only pays for instances while they are active. But this implies that as the guest service grows, e.g., due to increasing load, it grows only by adding more virtual nodes. The nodes that it has do not get faster, but it has more of them. This means that the application must itself be distributed. What are the challenges for building such a service? How should the parts of the app interact and interconnect? How do they protect/isolate themselves from other users and apps on the cloud? How do they spread the load among the nodes of the application? [Note: these really are thought questions. Half of my distributed systems course focuses on these questions. But you already know enough to get the fundamentals.]

8. Android defines four component types that can be mixed together in various ways to build complex application functions. Could we use the same component types for applications in the cloud, e.g., implementing the same component model using network communication rather than local binder channels? Are any important component types missing that would be useful in the cloud but not in Android? Are any of the Android component types not useful in the cloud?