**CS 153**

**Design of Operating Systems**

**Ch. 1 Introduction**

* 1. **The Client/Server/Service Paradigm**

**server** – *object*, facility, resource, device, service provider, instance of a class, record, struct.

**part** – *attribute*, field, member variable, instance variable, state variable

**service** – *(the kind of a) message*, operation, (signature of a) member function, mailbox, port

**issuing a service request** – *sending a message*, causing an event, accessing a resource, applying a stimulus, submitting a job

**handler** – *method*, service routine, body of a member function

**client** – *sender*, customer, service consumer, service recipient

**thread** – *active object/server/resource*, lightweight process

* *server* – has identity, state and behavior
* *service request* – event that affects a server
  + *service*
  + *arguments*
* server’s state – values of its attributes
* *rendering of a service* – server’s behavior in response to a given event
* *class* – each server is an instance of such
  1. **Operating Systems**

protection, coordination, and scheduling

*kernel* – contains the routines and data structures for the handling of kernel-invoking events: traps, interrupts, and system calls

*application-program interfaces* – libraries of functions for connecting to and managing resources

*daemons* – kernel executes them to assist in managing and keeping track of resources. They run in user-mode, using system calls when necessary

* 1. **Key Considerations**

*concurrency, protection, and dynamic binding*

* 1. **Hardware Support**

To faciliate the considerations, we have traps, interrupts, system calls, protection, memory management, I/O, storing/restoring of registers, and atomic operations.

* + 1. **Kernel Invocations**

*interrupt* – response to an action performed by a device external to that CPU

*trap* – direct result of an action by its currently running program, specifically an action that requires interruption

*system call* – request to the kernel to grant access to some system resource

Differences between kernel invocations and function calls:

* *vectored transfer of control* – happens via a level of indirection: CPU looks up the handler in a vector table.
* *suppression of this CPU’s protection level* – more instructions can be executed and more locations can be accessed without causing a protection trap
* *suppression of interrupts* – fewer devices are allowed to interrupt this CPU…postponed until unblocked
* *restoration of context* – protection and interrupt status must be simultaneously restored