Today's plan

- layered operating system
- seL4
- capabilities
- seL4 API and Objects
- seL4 threads
- seL4 scheduler

A Layered Operating System

- book, figure 2-3 – (*n* sequential processes layered on top of a scheduler)
- lowest layer contains scheduler and interrupt handlers that do the least possible to handle devices, as well as a mechanism to allow communication among processors
- all other services, including file systems, networking protocols, etc are separate processes
- to handle a disk interrupt, the disk interrupt handler builds a message and "sends" it to the file system manager
- if the file system manager was waiting for a message, it now becomes ready
- if the file system manager has high priority, the scheduler now makes it the current process (unless another process has more priority)
- the file system manager therefore executes quickly after the interrupt, perhaps issuing another message to a process that had performed a read system call
- a system like this is a **microkernel** system
- a famous early microkernel was Mach 3.0
- L4 is a currently popular microkernel, you can run a modified Linux on top of it

Operating System Structures

- a **monolithic** kernel is a single program. No message passing is needed, procedure calls are enough. Linux follows this model
- a **microkernel** has a "small-as-possible" kernel, and all other services are implemented as processes. Minix follows this model
- a **virtual machine** provides multiple computing environments equivalent to actual computers by responding appropriately when a "user program" performs a privileged operation. The instruction set of the virtual computer may be the same as of the actual computer (most virtual machines), or independent (Bochs)

L4 Kernel

- inspired by the Mach microkernel
- designed to be as fast as possible (originally written in assembly)
- machine dependent

seL4 Kernel

https://sel4.systems/Info/Docs/seL4-manual-latest.pdf

- create and manage virtual address spaces
- create and manage threads
- inter-process communication (IPC)
  - IPC used to send interrupts to unprivileged device drivers
- capabilities define access rights
- small implementation of 8700 lines of C
- proof of integrity and confidentiality
- analysis of worst-case execution time

seL4 Capabilities

- in Unix, a file descriptor is an integer that identifies an open file in the kernel
- a capability is likewise a reference to data stored in the kernel, that identifies something that this process may do
- capabilities may be sent via IPC
- new capabilities may be created by limiting some of the rights of existing capabilities
- capabilities can be revoked

seL4 API

- three basic system calls: send, receive, yield
  - call combines send+receive, reply responds to call
- the destination of a send is identified by a capability
  - this destination may be a thread, or an operation inside the kernel
- send and receive can both block
  - rendezvous: transmission only happens when both ready
  - non-blocking versions are available
- seL4 starts an initial thread with all the possible capabilities, including to all unused memory

seL4 Objects

- objects are allocated by the kernel within memory to which a thread has a capability
  - the thread loses access to that part of the memory
- capability nodes (**CNodes**) for capabilities
- thread control blocks, **TCB**s, one for each thread
- **endpoints** for IPC, and **notifications** and **interrupts** for interrupt handling
- **memory**: untyped memory and device memory
- **virtual address space** objects: pages and ASIDs
- all objects are managed from user-space

seL4 Thread Creation

- a thread is created like any other object, by allocating memory for it
- to activate a thread, must:
  - specify a capability for it (CSpace) and a virtual memory (VSpace)
  - set up an initial stack pointer and instruction pointer
  - call the resume method to schedule the new thread
- an **affinity** call can move the thread to a different CPU

seL4 Scheduler

- “preemptive round-robin scheduler with 256 priority levels”:
  - **round-robin** means threads are executed in turn
  - **preemptive** means a thread may be suspended
  - seL4 uses **strict priority**, meaning lower-priority threads are only executed when no higher-priority threads are ready
  - however, when a **domain** exhausts its time slot, the next domain will be scheduled
    - `ksDomainTime` decremented on each call to `timerTick`
- scheduler is in `src/kernel/thread.c`