Operating Systems

Introduction to Xinu

Operating Systems

- Complex software running on the same processor that runs application software
- Arranging for control to pass from the OS to an application, and back, is one of the more interesting aspects of OS design
- Computing hardware is complex and can perform operations very quickly, but it is interesting how simplistic and crude it actually is
- An OS is designed to hide low-level hardware details and to create and abstract machine that provides applications with high-level services

OS: The Xinu Approach

- OS controls processes, manages resources and communicates with external devices
  - Allows multiple processes/users to share the machine and keep multiple devices operating simultaneously
- Provide high-level services atop low-level hardware
  - Hiding the details of the hardware and I/O devices to make computers easy (or just possible) to program
  - Provide a simplified, abstract machine
- Above are general goals of operating systems as well
  - Xinu aims to do this minimalistically and practically

Our Xinu Platform

- The book describes Xinu on the ARM-based BeagleBone Black and the x86-based Galileo
- Previously, we used a port of "Embedded Xinu" to the ARM processor, running on the Raspberry PI
  - The previous version of the book described running Xinu on a Linksys home wireless router

Embedded Systems and Cross compilation

- Xinu has been successfully used as an embedded system
  - Set top boxes, MP3 players
- Frequently, embedded systems are too simple to host their own compiler
- For this, we need a cross-compiler
  - Generate code for one type of system on another type of system
- In these environments, we then need a way to download the generated binary to the system
  - Easy in a simulation environment such as QEMU

Compile and Load

- Our BBs have a bootloader on the SD card
  - Boots a simple environment able to download the Xinu code that you have compiled
  - Downloads from a server via the Trivial File Transfer Protocol (TFTP)
- You will build Xinu and copy the file to the proper directory
  - /tftpboot

OS Organization

- As OSes have evolved, so has our understanding of common components and functionality
- Code is complicated and can easily become unmanageable
- Modern operating systems are structured in modules with a separation of concerns
- This reduces complexity and makes it feasible to understand, extend and maintain the OS

Layering of subsystems in Xinu

- In providing the desired abstract services, systems often build up layers of increasingly powerful primitives
- We can describe this organization in terms of layers or rings (concentric circles)
  - THE Multiprogramming System, Dijkstra, 1968
- The goal (as discussed) was to isolate functionality into subsystems that could be understood and reasoned about
  - And higher-level layers can utilize the services of lower-layer ones
- Note that layer K is not limited to calling layer K-1
  - As is generally the case in e.g. network stacks
  - The layers do not define strict control flow, rather they are a conceptual framework

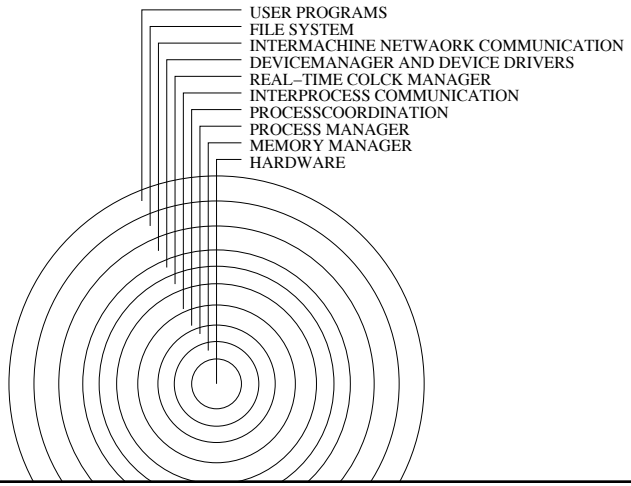
Layering of Subsystems in Xinu

- The lowest layer is considered to be the hardware and the highest, the user/program
  - Even though they are obviously not designed by the OS architect
  - Hardware design is somewhat guided by the needs of the OS
- At the heart is the **process scheduler** and **context switch** logic
  - Chooses a process and makes it ready to run
- The next layer is the **process manager**
  - Primitives to create, kill, suspend and resume processes
- The **process coordination** layer implements semaphores

Layering of Subsystems in Xinu

- Message passing is implemented in the **interprocess communication** layer
- The **real-time clock manager** schedules processes for time sensitive use cases
- The **device manager** and **device driver** layer implements device-independent input and output routines
- The next layer implements external communication - **intermachine communication**
- The highest layer implements and controls the **file system**
- The lowest layer is in fact the **memory manager**, which we will come back to

The Layers of Xinu



Xinu Services

- Programs access services by calling OS routines through system calls
  - True for Linux, UNIX, Windows, etc.
  - For example, putc writes a character to an I/O device

```
/* ex1.c - main */
#include <xinu.h>

/*------------------------------------------------------------------------
 * main -- write "hi" on the console
 *------------------------------------------------------------------------
 */

void main(void) {
    putc(CONSOLE, 'h');
    putc(CONSOLE, 'i');
    putc(CONSOLE, '\n');
}
```

The Console

- As seen in the example, the name CONSOLE refers to the basic I/O device
- To communicate with the system, one can connect to the serial device
  - "Serial" means that it transmits one bit at a time
  - The UART (Universal Asynchronous Receiver/Transmitter) is a device that will take bytes (in parallel) and transmit them serially
  - The "baud" refers to the number of symbols per second – the speed
  - Other parameters include the number of bits per character, number of "stop" bits and parity

Xinu Development Environment

- Students will use the Qemu hardware emulator
  - Some will use hardware devices as well
  - We have quite a few ARM-based Beagle Bone Blacks
- You will cross compile for ARM (or x86) from the departmental Linux systems or from your own machine
- Xinu will run on the emulator and you will interact with it over the console
- More to come in the discussion section!