KallistiOS

An embedded OS for Video Game Consoles

UMBC CMSC 421 - Spring 2012

Embedded Systems

- Computers (and OSes, of course) are everywhere!
- Low-power, low-memory devices make up a large proportion of the market
- These embedded devices require careful programming and much smaller code than many OSes today provide

Embedded Systems

- The video game consoles of yesteryear are very much like today's embedded systems
- They do not come with real Operating Systems installed on them -- they are included with/linked directly to the games
- If that's the case, then how are they an OS?

Embedded OSes

- Many current examples of embedded OSes look a lot more like traditional OSes, as the devices themselves are much more powerful
  - iOS
  - Linux (Android)
  - Windows Phone/Windows RT

Embedded OSes

- However, there are plenty of other embedded systems than just cell phones
  - On-board computers (ECUs and such) in cars
  - Medical equipment
  - Microcontroller-based systems

A Different Idea of an OS

- These low-powered devices require a fundamentally different idea of an OS than the other examples of embedded OSes
- Very little RAM, potentially no writable storage, a very specific set of devices to support, etc.
- Many features of an OS are not required or are completely useless on these!

A retrospective...

- As an example, lets take a closer look at the Sega Dreamcast
- Released in 1999 (1998 in Japan)

System Specifications

- 200 MHz Hitachi SuperH 4 processor
- 16 MB of system RAM
- PowerVR 2 GPU - 8 MB of Video RAM
- GD-ROM media (read-only)
- Various external peripherals (controllers, memory cards, camera, keyboard, mouse, network card)

What would its OS look like?

Enter KallistiOS

- KallistiOS is an embedded OS for video game consoles, including the Dreamcast
- Developed by the homebrew community without use of the official SDKs
- Lacks many of the abstractions of todays mainstream OSes, but makes up for it in its ease-of-use for programming and its relative speed

What KOS is

- A “pseudo-real-time OS”
  - Monolithic kernel with ability to load modules
- Hardware manager (interrupts, DMA, MMU, etc)
- Pseudo-POSIX layer (libc, pthreads, VFS)
- Hardware abstraction layer

What KOS does not do

- Full POSIX-compliance
- Multi-tasking (multiple independent processes)
- Memory protection

Well, how is that an OS?

- Think back to what an OS has as its main tasks...
  - Resource allocation
  - Control program
- Does KOS handle them? - Of course!

The KOS Kernel

- Divided into several subsystems:
  - Pseudo-POSIX layer
  - Virtual Filesystem
  - Threads
  - Networking
  - Hardware Support

Programming with KOS

- No user/kernel mode distinction (unless you want to provide it)
- Direct hardware access (for the most part)
- Several normal OS-like abstractions (libc, C++ iostreams, BSD sockets, partial OpenGL support)
- User programs statically link the kernel