Operating Systems

- bootloaders/monitors get the system into a known state
- the next step is to transfer control to an OS (or a stand-alone application)
- variety of OSes — depends on need and resources

Operating System Choices

- range from very “thin” or “lightweigth” to full-fledged OS
  - thin: requires very little resources (RAM) but doesn´t provide much application support (typically no memory management, no networking, no device drivers, incomplete C library)
  - full-fledged: requires more resources (minimum RAM, MMU hardware) but allows almost any program to be compiled — from web servers to windowing systems

Examples OSes

- AT/Nucleous — a lightweight real-time operating system
- Symbian — popular in mobile phone markets
- uCos
- μC-Linux — version of Linux for CPUs without memory management units
- VxWorks — highly configurable Unix implementation
- Linux — same as the one used on desktops and servers

Linux 2.4 — Forked Projects

- many embedded systems are built on 2.4 kernel
- advantages:
  - solid, stable
  - familiar
- disadvantages:
  - no longer being improved
  - fewer device drivers

Linux 2.6

- with 2.6 many new architectures support (PowerPC)
- advantages:
  - device drivers
  - stock kernel easier (no patches needed)
  - still evolving
- disadvantages:
  - still evolving

Linux v. Other Choices

- Linux has no licensing fees
- No direct support for Linux (3rd party)
- Linux development environment not geared to embedded systems
- Linux has momentum (IBM: $100M/year investment; Nokia, others as well)

Compiling Linux

- Step 1. Download kernel source
  `http://www.kernel.org/`
- Step 2. Unpack the archive
  `tar xfz linux-2.6.16.18.tar.gz`
- Step 3. Configure
  ```
  cd linux-2.6.16.18
  make menuconfig
  ```
- Step 4. Compile
  `make`

Learning Ins-and-Outs of Linux

- Two things to familiarize yourself with:
  - menuconfig organization (find the options)
  - directory structure (find the options)

Menuconfig

- simple, ASCII terminal program
- cascading menu items
  - top-level: more general
  - low-level: specific
- configures both
  - build (what files get compiled)
  - options (what features are included)

Menuconfig — Tri

- options
  - yes/no : represented by *=yes, SPACE=no
  - build-in/leave-out/module:
    *=build-in, SPACE=exclude, M = loadable module
- choices effect...
  - features/capabilities (of course)
  - resident size of operating system (RAM)
  - development time (how long to compile Linux!)

Linux 2.4 Menuconfig

```
Code maturity level options -->
Loadable module support -->
Platform support -->
General setup -->
Memory Technology Devices (MTD) -->
Plug and Play configuration -->
Block devices -->
Multi-device support (RAID and LVM) -->
Cryptography support (CryptoAPI) -->
Networking options -->
ATA/IDE/MFM/RLL support -->
SCSI support -->
```

Linux 2.4 Menuconfig (cont'd)

```
IrDA (infrared) support -->
ISDN subsystem -->
Old CD-ROM drivers (not SCSI, not IDE) -
Console drivers -->
Input core support -->
Macintosh device drivers -->
Character devices -->
Multimedia devices -->
File systems -->
Sound -->
IBM 4xx options -->
USB support -->
```

Figuring Out Options

- many options have short (cryptic) help message
- result of menuconfig is a `.config` file
  - Makefile macros
  - #-defines in C source code
- don't understand an option in menuconfig, check the source (C and Makefiles)

2.4 Build Options

- `make menuconfig` — builds .config file; sets up make
- `make oldconfig` — sets up make from existing .config
- `make dep` — builds C/header dependencies for make
- `make bzImage` — builds an ELF executable, compresses
- `make zImage.initrd` — builds an ELF executable, compresses, adds ramdisk
- `make modules` — builds dynamically loadable modules
- `make clean` — removes dependencies, object, executable
- `make mrproper` — clean and remove .config

Linux 2.4 Build Example

(online)

2.4 Kernel Directory Structure

at the top level, there are several key directories

- `arch` — architecture-specific files (PPC, i386, etc.)
- `Documentation` — text file descriptions
- `drivers` — support for various peripheral devices
- `fs` — code for different filesystems
- `init` — start/stopping kernel
- `kernel` — scheduler, timer, etc.
- `net` — TCP/IP networking code
- `mm` — memory management

2.4 Drivers Directory Structure

under the `drivers` directory

- general support for various (general) high-speed and low-speed buses
- `block` — specific block-oriented devices (hard drive)
- `char` — specific stream-oriented devices (terminals)
- `net` — specific network interface chips (Ethernet NIC)
- `sound` and `video` — specific multimedia chips

Linux 2.4 Directory Example

(online)

Making Sense of Options

- practice, practice, practice
- learn the PC architecture
  - IDE v. SATA
  - I2C v. SPI
  - 16550 UART v. USB
- look at working .configs (/proc/ikconfig/config)

Making Sense of Source Code

- study easy things first
  - study a simple driver for familiar hardware
  - look at (arch) architecture-specific code
  - investigate networking stack
  - tackle kernel, memory management, etc.
- make small changes and try to compile
- /proc is your friend!