Linux Kernel Overview

What to Expect?

- W's of Kernel
- Linux Architecture
- Linux Kernel Startup
- Linux Kernel Functionality
- Linux Kernel Configuration
- Linux Kernel Compilation

What is a Kernel?

Core of a System

The Operating System

OS Core

- OS Core could be further classified as the following major functionalities
  - Inter Process Communication
  - Minimal Memory Management
  - Low-level Process Management & Scheduling
  - Low-level Input / Output

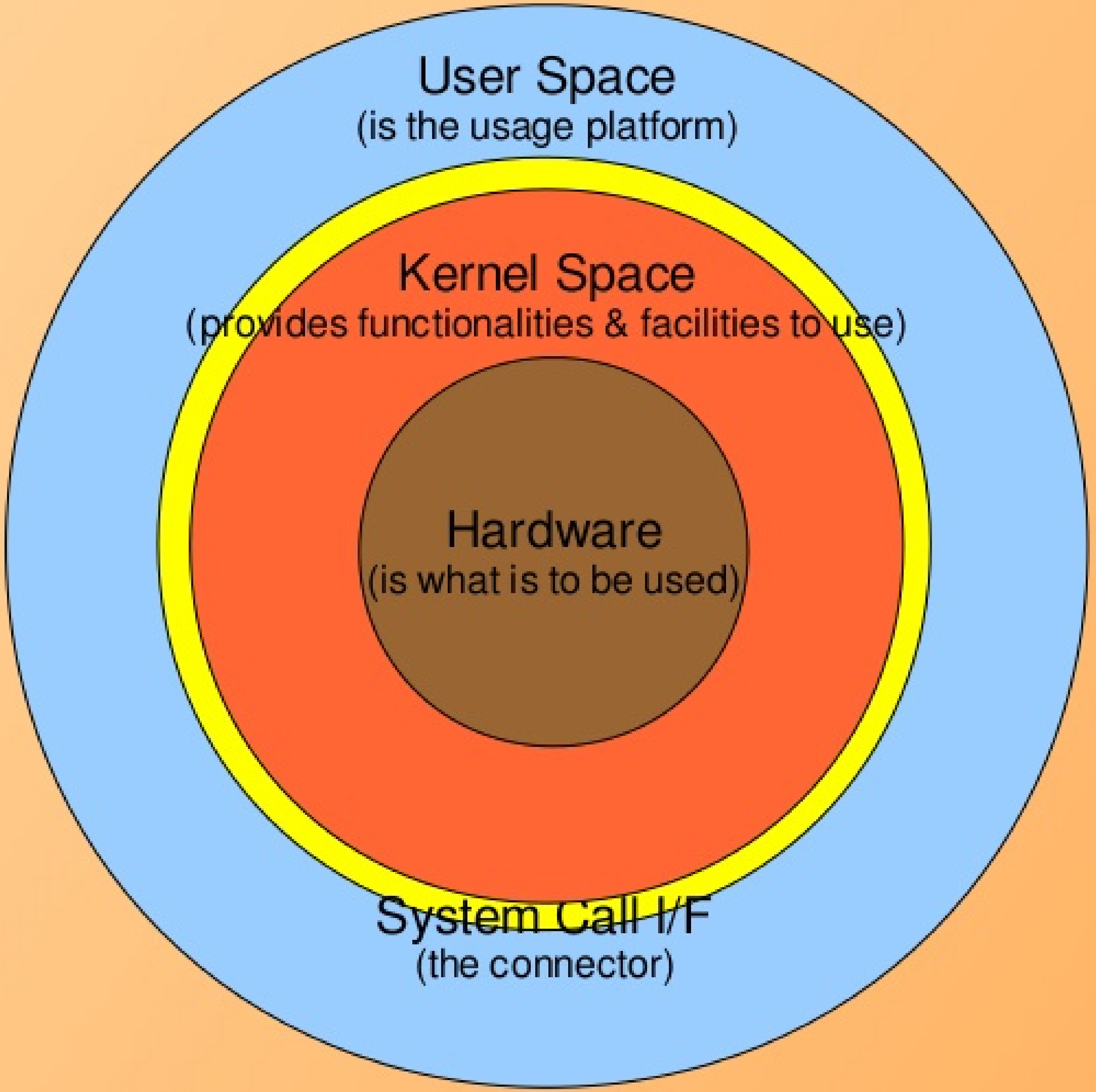
Types of Kernels

- Micro kernel
  - Also called the Modular kernel
  - Contains only the OS Core
  - Other OS stuff are typically provided as services
  - Examples: Amoeba, Mach, QNX
- Monolithic kernel
  - Contains all the OS related stuff
  - Either built into it statically or loaded dynamically
  - Examples: VxWorks, Linux

Micro vs Monolithic Advantages

- Micro Kernel
  - Flexible
  - Modular
  - Easy to implement
- Monolithic Kernel
  - Performance

Linux Architecture



Linux Architecture Details

- User Space
  - Memory where user processes run
  - Doing typical computation tasks
  - And accesses kernel space for privileged tasks
  - Through System Call interface
- Kernel Space
  - This is protected space
  - Place of all privileged happening
    - I/O access, Memory access, ... (System Resources access)
- Kernel processes can access User processes but not vice versa
- These levels are achieved by processor states
- How does system call switches user to kernel space?
- Check
  - Which space is executing command from root?
  - Do we need these levels and system calls if the kernel image and root file system are read only?

Process vs Thread

- Single Process Single Threaded
- Single Process Multi-Threaded
- Multi-Process Single Threaded
- Multi-Process Multi-Threaded
- What is Linux User Space?
- What about the Kernel Space?
- Need for Single vs Multi

Linux Kernel Startup

- CPU / Platform specific Initialization
  - CPU Speed Setup
  - MMU Setup
  - Board Id Setup
- Minimal Driver Initialization: FS specific
- Mounting Root File System
- Remaining Driver Initialization
- Doing Initcall & Freeing Initial Memory
- Moving to User Space by
  - Jumping to the first user process init

Linux Kernel Functional Overview

- Process Management
- Memory Management
- Device Management
- Storage Management
- Network Management

Linux Kernel Source

Let's get down to the Source Code

Linux Kernel Build System

- Key components
  - Makefile
  - Kconfig
- Configuring the Makefile
  - Setting up the kernel version (specially for the Desktops)
  - For Cross Compilation, need to setup
    - ARCH
    - CROSS_COMPILE
  - Or, invoke make with these options

Linux Kernel Configuration Methods

- make config
- make menuconfig
- make xconfig
- Others
  - make defconfig
  - make oldconfig
  - make <board_specific>_defconfig
- Check: Where is the menuconfig target?

Linux Kernel Configuration

- Code Maturity level Options
- General Setup
- Loadable Module Support
- Block Layer
- Networking
- Device Drivers
- File Systems
- Kernel Hacking
- Security Options
- Cryptographic Options
- Library Routines

Linux Kernel Compilation

- Cleaning Methods
  - make clean – Simple clean
  - make mrproper – Complete sweep clean, incl. Configs
- Also called Building the Kernel
- After configuring the kernel, we are all set to build it
- Build Methods
  - make vmlinux – To build everything configured for a kernel image
  - make modules – To build only configured modules
  - make – To build everything configured (kernel image & modules)
  - make modules_prepare – To only prepare for building modules

Linux Kernel Images

- Kernel Image should be understood by Stage 2 Bootloader
- Default kernel compilation builds vmlinux
- vmlinux is understood only by the desktop bootloaders
- So, for embedded systems, we would typically have to do the following
  - Creating linux.bin using <cross>-objcopy
    - Example: arm-linux-objcopy -O binary vmlinux linux.bin
  - And then, convert it into the bootloader specific image using some bootloader utility. For u-boot, it is done using mkimage
    - Example: mkimage -A arm -O linux -T kernel -C none -a 20008000 -e 20008000 -n “Custom” -d linux.bin uImage.arm

Linux Kernel Arguments

- console
- root
- initrd
- mem
- resume
- ...

What all have we learnt?

* W's of Kernel
* Linux Architecture
* Linux Kernel Startup
* Linux Kernel Functionality
* Linux Kernel Configuration
* Linux Kernel Compilation

Any Queries?