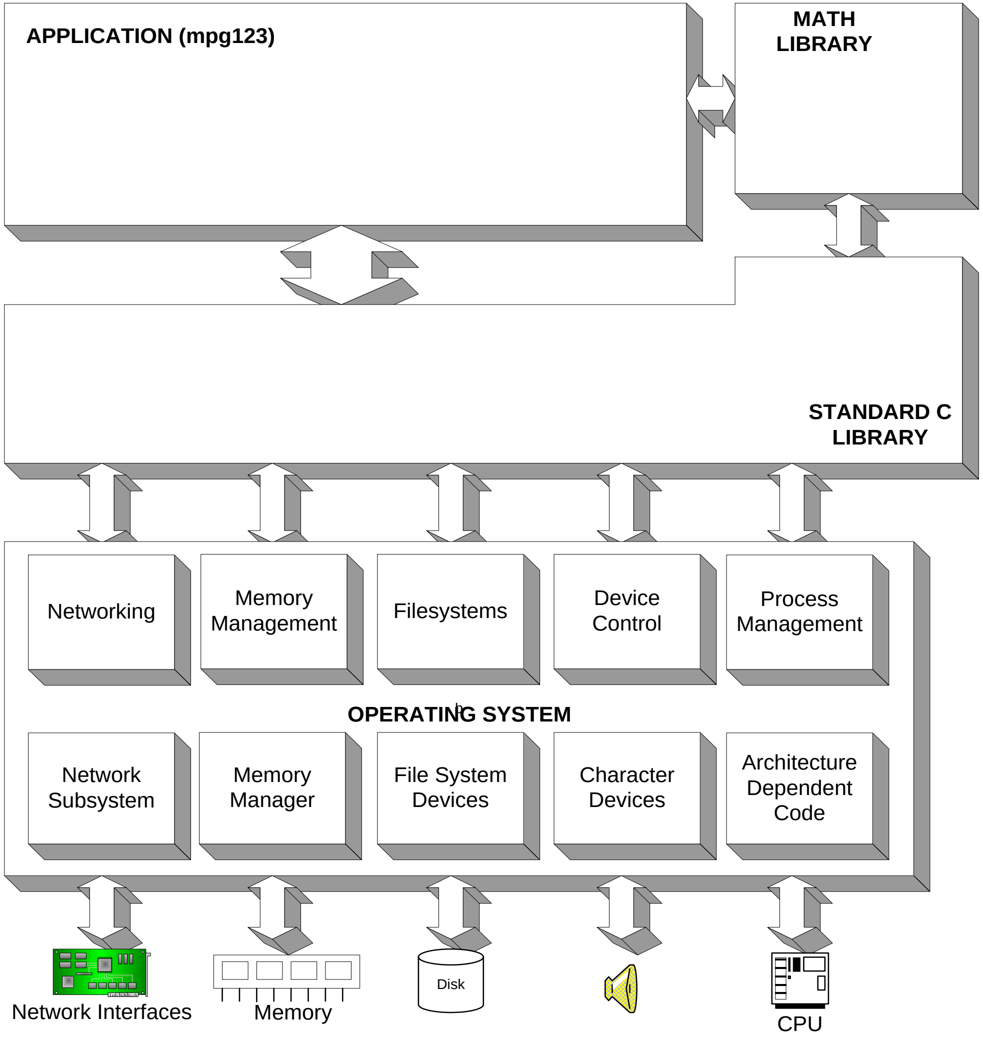
Kernels and Device Drivers

Linux Execution Environment



- Program
- Libraries
- Kernel subsystems

Process

- Process: program in execution.
  - Unique “pid”. Hierarchy
- User address space vs. kernel address space
- Application requests OS services through TRAP mechanism
  - x86: syscall number in eax register, exception (int $0x80)
  - result = read (file descriptor, user buffer, amount in bytes)
  - Read returns real amount of bytes transferred or error code (<0)

- Kernel has access to kernel address space (code, data, and device ports and memory), and to user address space, but only to the process that is currently running
- **“Current**” process descriptor. “**current→pid**” points to current pid
- Two stacks per process: user stack and kernel stack
- Special instructions to copy parameters / results between user and kernel space

Splitting the Kernel

- Process management
  - Creates, destroys processes
  - Supports communication among processes
    - Signals, pipes, etc.
  - Schedules how processes share the CPU
- Memory management
  - Virtual addressing

Splitting the Kernel

- File systems
  - Everything in UNIX can be treated as a file
  - Linux supports multiple file systems
- Device control
  - Every system operation maps to a physical device
    - Few exceptions: CPU, memory, etc.
- Networking
  - Handles packets
  - Handles routing and network address resolution issues

Device drivers

- Device drivers
  - Black boxes to hide details of hardware devices
  - Use standardized calls
    - Independent of the specific driver
  - Main role
    - Map standard calls to device-specific operations
  - Can be developed separately from the rest of the kernel
    - Plugged in at runtime when needed

The Role of the Device Driver

- Implements the *mechanisms* to access the hardware
  - E.g., show a disk as an array of data blocks
- Does not force particular *policies* on the user
  - Examples
    - Who many access the drive
    - Whether the drive is accessed via a file system
    - Whether users may mount file systems on the drive

Policy-Free Drivers

- A common practice
  - Support for synchronous/asynchronous operation
  - Be opened multiple times
  - Exploit the full capabilities of the hardware
- Easier user model
- Easier to write and maintain
- To assist users with policies, release device drivers with user programs

Loadable Modules

- The ability to add and remove kernel features at runtime
- Each unit of extension is called a *module*
- Use **`insmod`** program to add a kernel module
- Use **`rmmod`** program to remove a kernel module

Classes of Devices and Modules

- Character devices
- Block devices
- Network devices
- Others

Character Devices

- Abstraction: a stream of bytes
  - Examples
    - Text console (**`/dev/console`**)
    - Serial ports (**`/dev/ttyS0`**)
  - Usually supports **`open`**, **`close`**, **`read`**, **`write`**
  - Accessed sequentially (in most cases)
  - Might not support file seeks
  - Exception: frame grabbers
    - Can access acquired image using **`mmap`** or **`lseek`**

Block Devices

- Abstraction: array of storage blocks

- However, applications can access a block device in bytes
  - Block and char devices differ only at the kernel level
  - A block device can host a file system

Network Devices

- Abstraction: data packets
- Send and receive packets
  - – Do not know about individual connections
- Have unique names (e.g., **`eth0`**)
  - – Not in the file system
  - – Support protocols and streams related to packet transmission (i.e., no **`read`** and **`write`**)

Other Classes of Devices

- Examples that do not fit to previous categories:
  - USB
  - SCSI
  - FireWire
  - MTD

File System Modules

- Software drivers, not device drivers
- Serve as a layer between user API and block devices
- Intended to be device-independent