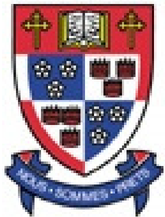
Real Time and Embedded Systems

by

Dr. Lesley Shannon

Email: lshannon@ensc.sfu.ca

Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc351



Simon Fraser University

Slide Set: 2

Date: September 13, 2011

Opening Thoughts

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning."

- Author **Rick Cook**, *The Wizardry Compiled*

Slide Set Overview

- What is the "Kernel"?
- Microkernels and Monolithic kernels
- QNX
- Linux/PetaLinux

The KERNEL

The Kernel

- The kernel is the …

The Kernel

- Abstracts low-level system resources from applications
- Parts of the OS critical to its correct operation
  - Operates in supervisor mode
  - Everything else is in user mode
- Operates as trusted software:
  - Intended to implement protection mechanisms that cannot be changed through actions of untrusted software
- Decides which thread runs

The Kernel

- Starts the execution of the selected thread
  - AKA ***switches context*** to the selected thread
- Context switching threads involves
  - Saving the currently running thread's registers and other context information
  - Loading the new thread's registers and context into the CPU
- What does the context switch remind you of?

Context for Process/Thread Context Switches: Recall Context Switches for subroutines and interrupts

- “Branch subroutine” (ie bsr)
  - Save PC and registers used on stack

- Interrupts (ie traps)
  - Save PC, registers used, AND SR on stack

The Kernel

- Keeps track of the state of all threads for thread scheduling

- Possible thread states:
  - Running
    - means the thread is actively running on the CPU
  - Ready
    - means that the thread ***could*** run right now

The Kernel

- Possible Thread States (cont'd):
  - Blocked
    - Thread is waiting for something to happen
    - There are multiple reasons a thread can end up in a blocked state
      - The kernel keeps track of why a thread can't run

Types of Kernels

Types of Kernels

- Monolithic kernels
  - Linux, UNIX, MS-DOS
- Microkernels
  - QNX, Mach, CHORUS, GNU/Hurd, L4…
    - Windows NT was "supposed" to be, but not
- Other kernel types

Monolithic Kernels

- The entire kernel is run in the kernel space in supervisor mode
- Uses a set of primitives or system calls to implement operating system services such as:
  - process management,
  - concurrency, and
  - memory management
  - … all in kernel mode

Monolithic Kernels

- The code integration is very tight and difficult to do correctly
- Since all the modules run in the same address space, a bug in one module can crash the whole system.
- When done well, the tight internal integration of components makes a good monolithic kernel highly efficient.

Monolithic Kernel: “The layers in a Linux System”

Monolithic Kernel: The “Structure of the Linux Kernel”

Microkernels

- Provides no operating-system services at all (pure form)
  - only the *mechanisms* needed to implement such services including:
    - low-level address space management,
    - thread management, and
    - inter-process communication
- Designed to be Modular and Small

Microkernels

- It is the only part of the system executing in kernel mode
- The actual operating-system services are provided by user mode servers including:
  - device drivers,
  - protocol stacks,
  - file systems, and
  - the user interface
  - … outside the kernel

Microkernel Structure

Other Kernels

- There are other kernels, but you don’t need to worry about them

- If you are interested, check out:
  - Hybrid Kernels
  - Nanokernel
  - Exokernel

QNX

QNX

- The higher-level OS functionality is provided by:
  - A tiny kernel that provides minimal services
  - A team of ***optional*** cooperating processes
- The QNX kernel does not have many typical OS services
  - These are provided by optional processes

QNX

- Optional system processes could include:
  - Filesystem managers
  - Character device managers
  - Graphical user interface (Photon)
  - Native network manager
  - TCP/IP
- System processes are essentially the same as user-written applications

QNX

QNX

- The Kernel provides:
  - Thread services
    - via POSIX primitives
  - Signal services
    - via POSIX primitives
  - Synchronization services
    - via POSIX primitives

QNX

- The Kernel provides (cont'd):
  - Timer services
    - via POSIX
  - Scheduling services
    - via POSIX realtime scheduling algorithms
  - Process management services
    - Process manager + kernel = procnto
      - Manages processes, memory, and pathname space

QNX

- The Kernel provides (cont'd):
  - Message-passing services
    - The kernel routes all messages between all threads in the system

- Inter-Process Communication (IPC) services
  - Allows communication between all processes (application or device drivers)

QNX

- Unlike threads, the kernel is never scheduled for execution
  - The code in the kernel is only executed as the result of:
    - an explicit kernel call,
    - an exception, or
    - a response to a hardware interrupt

Linux/PetaLinux

PetaLinux

- PetaLinux 2.1 implements a full Linux Kernel
  - Version 2.6.37; updates include a device tree
- Therefore PetaLinux has many typical OS services
- Recall the “Structure of the Linux Kernel”

Linux

- System processes include:
  - Filesystem managers
  - Character device managers
  - Block device drivers
  - Network Device Drivers
  - TCP/IP protocols
- Can also include ***Loadable Kernel Modules (LKMs)***
  - Think Dynamically Loaded Libraries (DLLs)

Linux/PetaLinux

- The PetaLinux Kernel provides:
  - Thread services
    - via POSIX primitives
  - Signal services
    - via POSIX primitives
  - Synchronization services
    - via POSIX primitives

Linux/PetaLinux

- The PetaLinux Kernel provides (cont'd):
  - Timer services
    - via POSIX
  - Scheduling services
    - via POSIX "realtime" scheduling algorithms
      - Real-time FIFO; Real-time Round Robin; Timesharing
  - Process management services & Memory Management services

QNX

- The Kernel provides (cont'd):
  - Inter-Process Communication (IPC) services
    - Allows Communication between all processes (application or device drivers)

- Message-passing services are supported via libraries
  - Check out the Message Passing Interface (MPI) functions designed for C and Fortran-77

Questions?

- What type of kernel is the QNX kernel? What about Linux?

- Why would someone today still choose to use a monolithic kernel?

Questions?

- What type of kernel should be used in Embedded and Real Time systems? Why?

- Recalling our discussion of processor architectures, why might a multi-threaded implementation of an application be slower than a single-threaded version?

Question

- What additional information has to be saved for a thread context switch from that of a context switch between subroutines or a software interrupt?

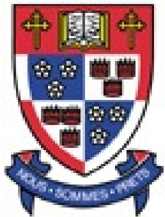
Real Time and Embedded Systems

by

Dr. Lesley Shannon

Email: lshannon@ensc.sfu.ca

Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc351

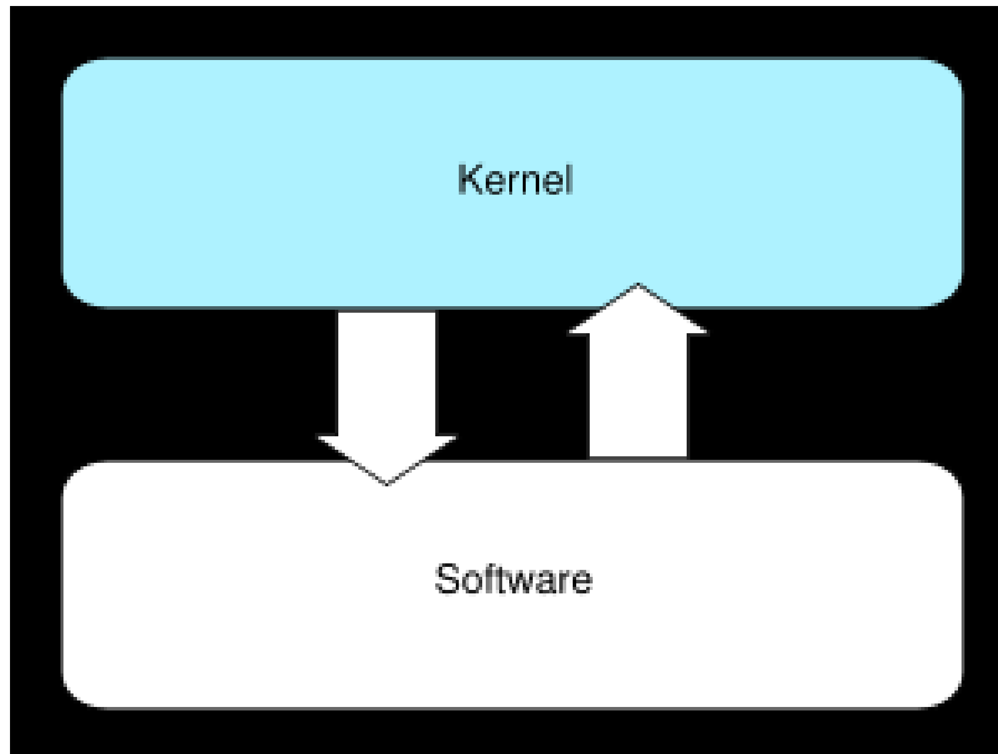


Simon Fraser University

Slide Set: 2

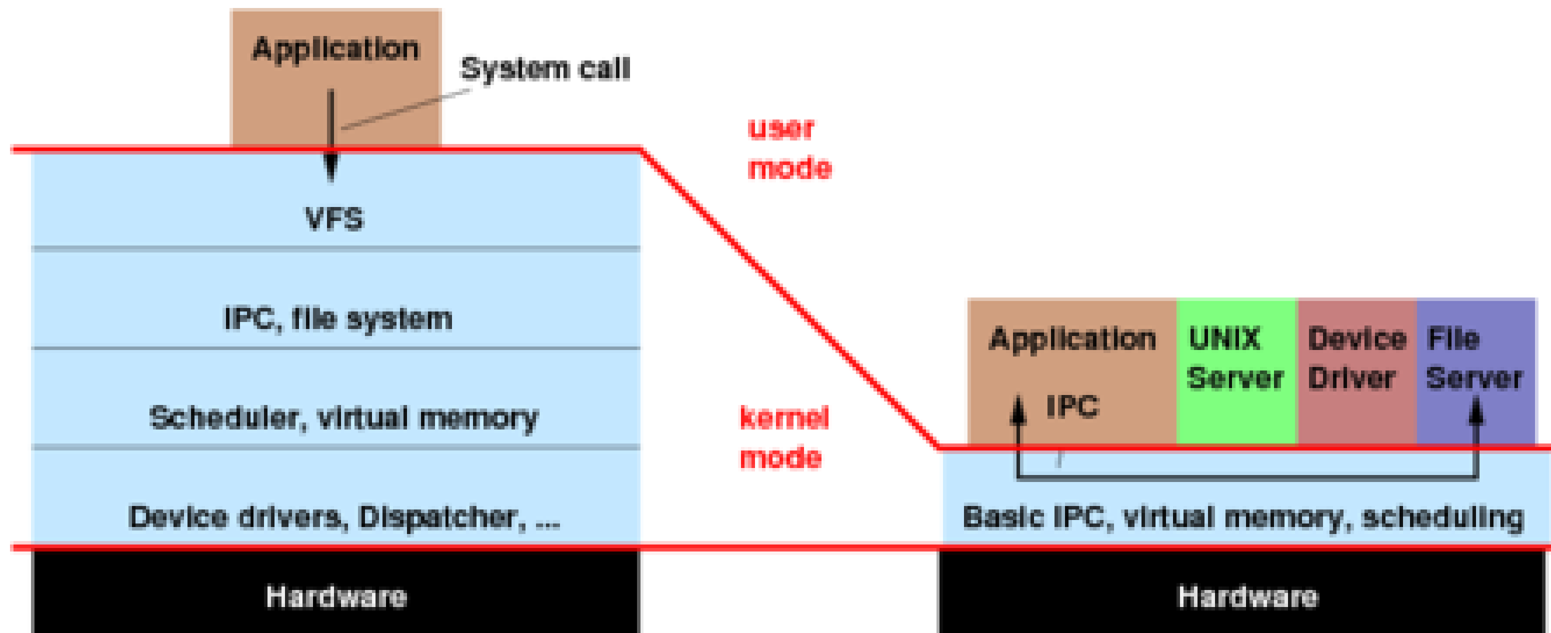
Date: September 13, 2011

Monolithic Kernels



Courtesy of Wikipedia

Microkernels



Courtesy of Wikipedia

Kernel-level threads vs User-level Threads

Kernel-level Threads

- Supported directly by the O/S
- The O/S has a separate thread for each process**
  - Performs O/S activities on behalf of the O/S
  - What is the name of this thread?
    - [Hint: Recall the Collaboration graph Notation discussion]

***At least*- depends on the multi-threading model

Kernel-level Threads

- Performs thread creation, scheduling and management in kernel space
- Thread management inside the kernel is generally slower that in user space
  - i.e. Takes longer to create and manage kernel threads than user threads

User-level Threads

- Run on top of the kernel
- Only exist within a process
  - Cannot access a thread in a different process
    - Isolation and modularity provide reliability
- Used by programmers to handle multiple flows of control within a process

User-level Threads

- The kernel is unaware of user-level thread
- Implemented with a thread library
  - Example?
  - Supports creation and scheduling management outside of the kernel
    - Done in user space without the kernel
      - Generally fast to create and manage