Real-time Support in Operating Systems

Colin Perkins

http://csperkins.org/teaching/2003-2004/rtes4/lecture11.pdf

UNIVERSITY
of
GLASGOW

Lecture Outline

- Overview of the rest of the module
- Real-time support in operating systems
  - Overview of concepts
  - Examples of real time operating systems

Reading for this week: Chapter 12 + Appendix

Copyright © 2004 Colin Perkins
http://csperkins.org/

Real Time & Embedded Systems

- Course overview:
  - Lectures 1-10: theory of real-time systems, covering scheduling and resource management
  - Lectures 11-20: the pragmatics of building real-time systems with available operating systems and network stacks

- Assessment:
  - Two assessed problem sets in weeks 1-5
  - Assessed program design exercise in weeks 6-10
  - 20% of grade derived from assessed course work
    - 2 problem sets @ 4% each
    - 1 program design assignment @12%
  - 80% of grade derived from exam mark
    - Answer 3 out of 4 questions
    - 2 questions from each part of the course
    - Sample paper will be distributed later in the term

Copyright © 2004 Colin Perkins
http://csperkins.org/

Real Time & Embedded Systems

Week beginning	Tue, 15:00-16:00	Wed, 12:00-13:00	Thu, 12:00-13:00
12 January	No meeting	Lecture 1	Lecture 2
19 January	Q&A	Lecture 3	Lecture 4
26 January	Q&A	Lecture 5	Lecture 6
2 February	Q&A	Lecture 7	Lecture 8
9 February	Q&A	Lecture 9	Lecture 10
16 February	Lecture 11	Lecture 12	Lecture 13
23 February	Lecture 14	Lecture 15	Lecture 16
1 March	Individual work on program design assignment		
8 March	Q&A	Lecture 17	Lecture 18
15 March	Q&A	Lecture 19	Lecture 20

Copyright © 2004 Colin Perkins
http://csperkins.org/

Real Time & Embedded Systems

Topic	Lecture	Pre-Reading
Real-Time Support in Operating Systems	11	Chapter 12 + Appendix
Scheduling in Practice	12	
Operating System Support for Concurrency	13	
Introduction to Real-Time Communications	14	Chapter 11
Quality of Service for Packet Networks	15	
Real-Time Communications on IP Networks	16	
Real-Time on General Purpose Systems	17	Chapter 10
Real-Time Embedded Systems	18	
Low-Level Programming	19	
Review of Major Concepts	20	

Copyright © 2004 Colin Perkins
http://csperkins.org/

Real Time Support in Operating Systems

- Real time versus general purpose operating systems
  - Real time constraints
  - Implications on **r**eal **t**ime **o**perating **s**ystem (RTOS) design
- Real time operating system concepts
  - Overall system architecture
    - Microkernels and nanokernels
  - Time services and scheduling
    - Cyclic executives, tasks
    - Clocks and timers
    - Interrupts and devices
  - Optimising system calls
  - Other services
    - Messaging, signals and events
    - Concurrency, synchronisation and locking
    - Memory protection
- Examples of real-time operating systems

Copyright © 2004 Colin Perkins
http://csperkins.org/

Constraints on Practical Real Time Systems

- Need predictable behaviour
  - Raw performance less critical than consistent and predictable performance; hence focus on scheduling algorithms
  - Don’t want to fairly share resources
- Need to run on a wide range of hardware
  - Often custom hardware for each application
  - Not just peripherals: system-on-a-chip designs are common
- Often resource constrained
  - Limited memory, CPU, power consumption, size, weight
- Often safety critical
  - Limited functionality is easier to verify and certify
- Embedded and may be difficult to upgrade
  - Closed set of applications, trusted code
  - Strong reliability requirements
  - How to upgrade software in a car engine? A DVD player? After you shipped millions of devices?

Copyright © 2004 Colin Perkins
http://csperkins.org/

Implications on Operating Systems

- General purpose operating systems are not well suited for real time systems
  - They assume plentiful resources that need to be fairly shared amongst un-trusted users
  - Exactly the opposite of an RTOS!

- Instead want something that is:
  - Small and light on resources
  - Predictable
  - Customisable, modular and extensible
  - Reliable

  …and that can be **demonstrated** or **proven** to be so

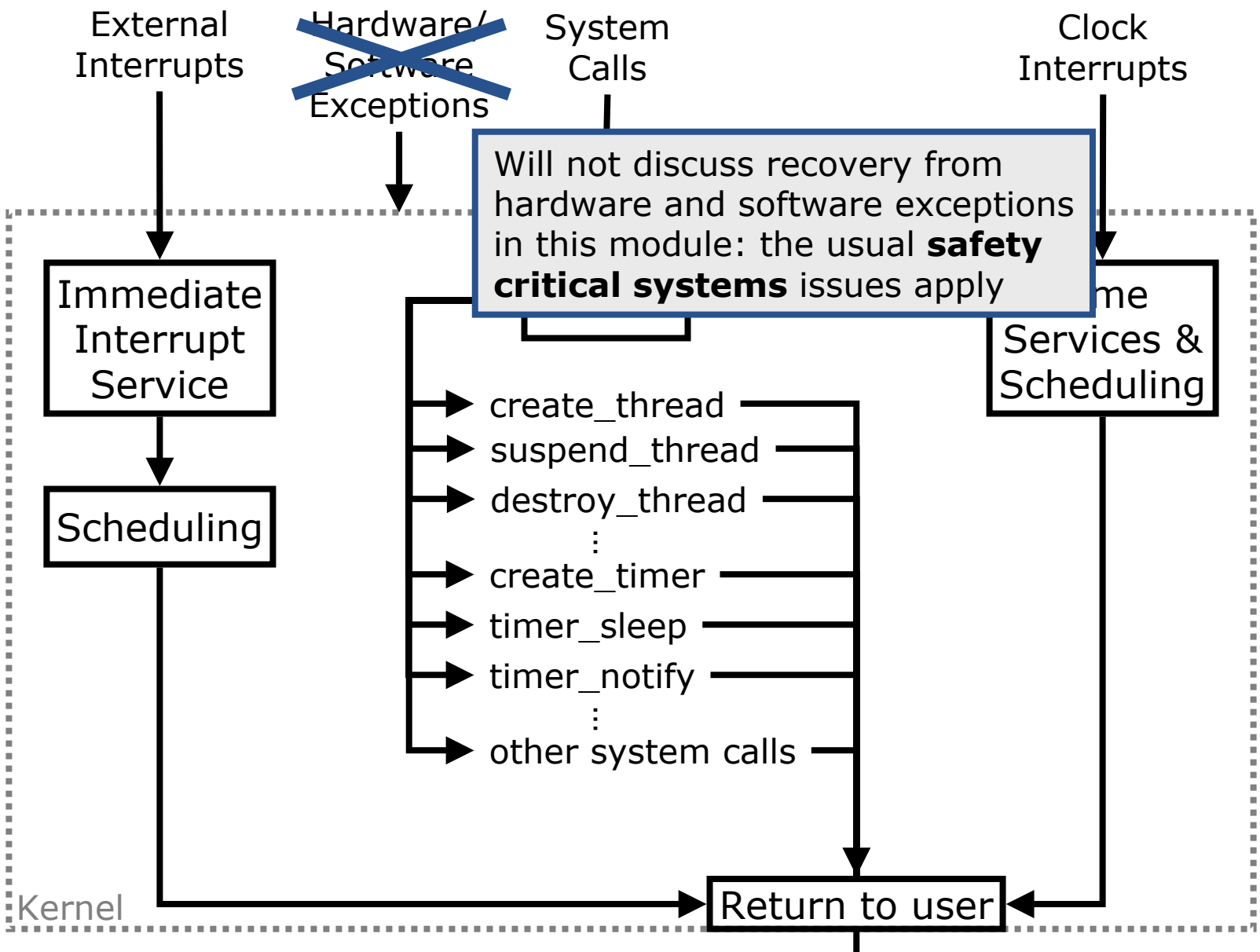
Copyright © 2004 Colin Perkins
http://csperkins.org/

Real-Time Operating System Concepts

- A real-time operating system is typically implemented as a microkernel, rather than a traditional monolithic kernel
  - Limited and well defined functionality
  - Easier to demonstrate correctness
  - Easier to customise
- Provide rich scheduling primitives
- Provide rich support for concurrency
- Expose low-level system details to the applications
  - Control of scheduling
  - Power awareness
  - Interaction with hardware devices

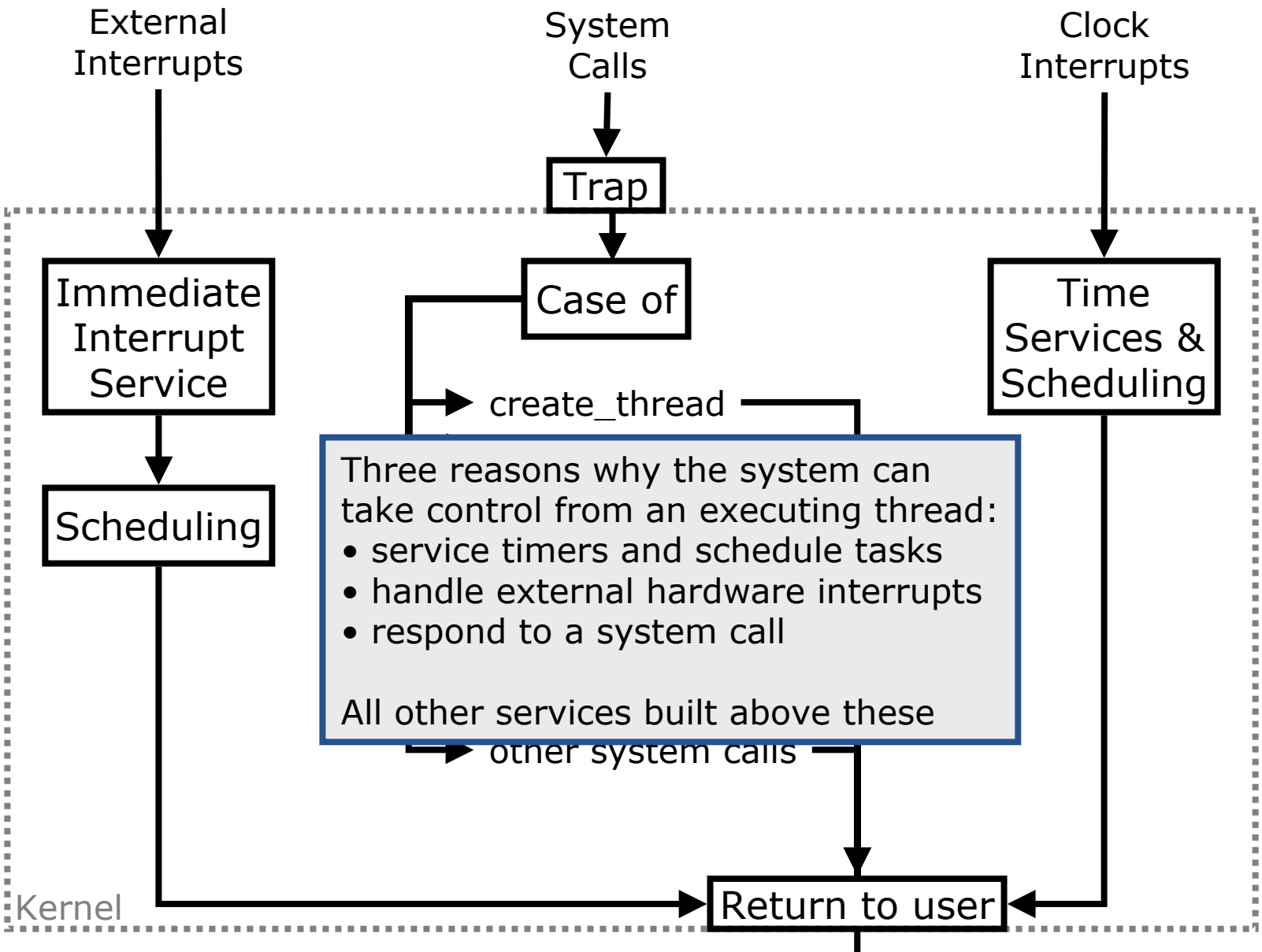
Copyright © 2004 Colin Perkins
http://csperkins.org/

Microkernel Architecture



Copyright © 2004 Colin Perkins
http://csperkins.org/

Microkernel Architecture



Copyright © 2004 Colin Perkins
http://csperkins.org/

Time Services: Basic

- The minimal time service is a scheduled clock pulse with fixed period
- Allows implementation of a static cyclic schedule, provided:
  - tasks periods are a multiples of the clock period
  - all interactions with hardware to be done on a polled basis
- Operating system becomes a single task **cyclic executive**:

```
setup timer
c = 0;
while (1) {
    suspend until timer expires
    c++;
    do tasks due every cycle
    if ((c % 2) == 0) do tasks due every 2nd cycle
    if (((c+1) % 3) == 0) {
       do tasks due every 3rd cycle, with phase 1
    }
}
```

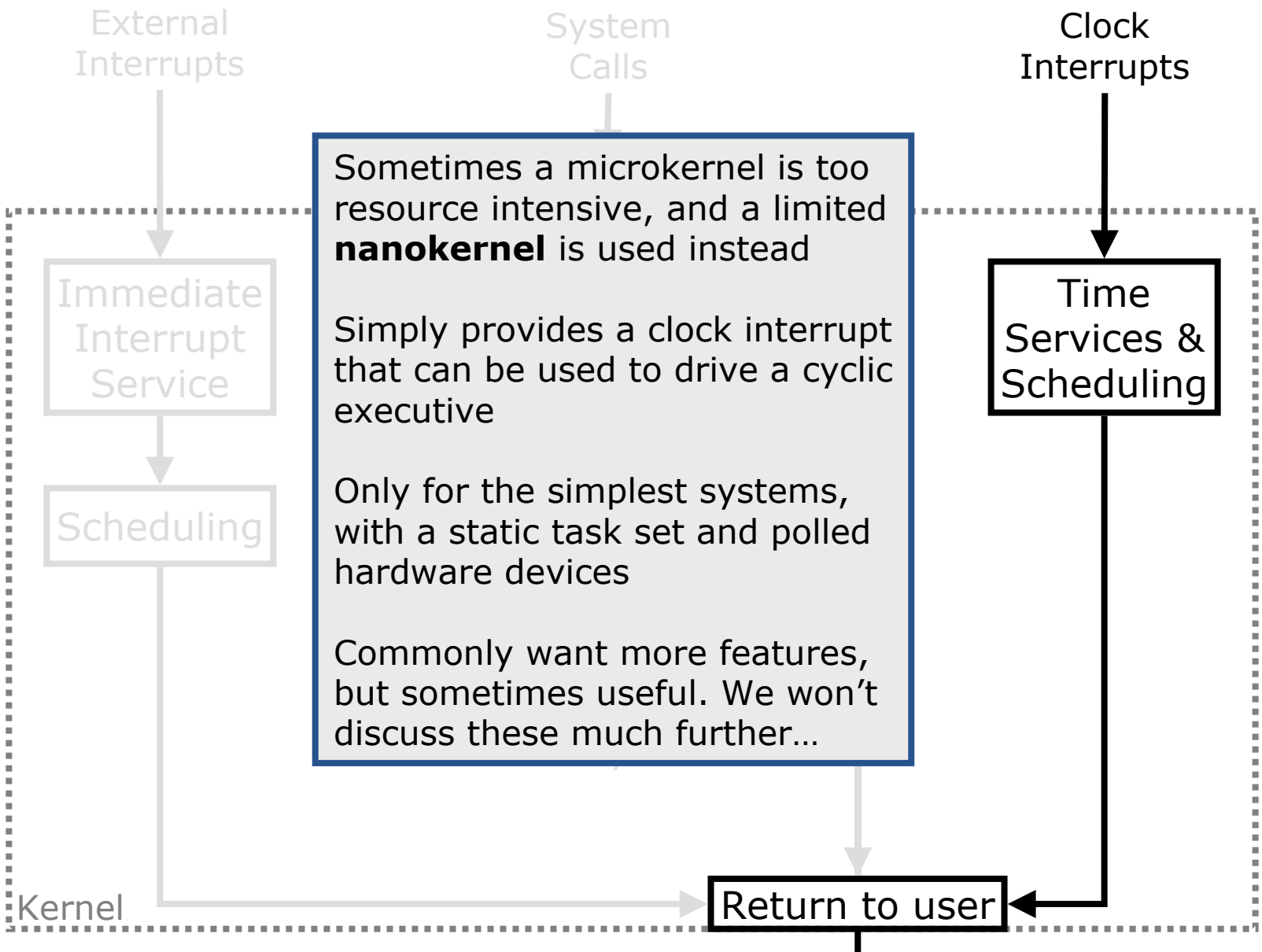
Copyright © 2004 Colin Perkins
http://csperkins.org/

Time Services: Helicopter Control Example

```
setup timer
c = 0;
while (1) {
    Suspend until 1/180th second timer expires
    c++;
    Select data source, collect and validate sensor data
    if (failure occurred) reconfigure system;
    if ((c % 6) == 0) {      // 30Hz
            Collect Keyboard input and mode selection
            Perform data normalization and transform coordinates
            Update tracking reference
            Compute outer pitch- and roll-control parameters
            Compute outer yaw- and collective-control parameters
    }
    if ((c % 2) == 0) {      // 90Hz
            Compute inner pitch-control parameters
            Compute inner roll- and collective-control parameters
    }
    Compute the inner yaw-control parameters
    Output flight control commands based on computed parameters
    Carry out built-in-test
}
```

Copyright © 2004 Colin Perkins
http://csperkins.org/

Nanokernel Architecture



Sometimes a microkernel is too resource intensive, and a limited **nanokernel** is used instead

Simply provides a clock interrupt that can be used to drive a cyclic executive

Only for the simplest systems, with a static task set and polled hardware devices

Commonly want more features, but sometimes useful. We won't discuss these much further…

Copyright © 2004 Colin Perkins
http://csperkins.org/

Time Services: Tasks and Scheduling

- Cyclic executives trivial to implement, but inflexible and only allow simple clock-driven schedules to be implemented
- If we want to implement other scheduling algorithms, need support for tasks
  - Operations to **`create`**, **`destroy`**, **`suspend`** and **`resume`** tasks
  - Can be implemented as either threads or processes
    - Processes not always supported by the hardware, or useful
- Scheduling triggered by:
  - Periodic timer interrupt
  - Threads being created or destroyed
  - Threads suspension and resumption
- Most systems support fixed priority scheduling of tasks
  - A few also support deadline scheduling

- Implementation of schedulers will be discussed in detail tomorrow

Copyright © 2004 Colin Perkins
http://csperkins.org/

Time Services: Other Clocks

- In addition to the **clock interrupt** used for scheduling, most systems support **time service interrupts** derived from high resolution hardware timers
  - Either "one-shot" timers or a high frequency interrupt source
  - Useful to application authors
- Two key issues should be noted:
  - The time service hardware often has nanosecond resolution, but this is rarely available to the application
    - Interrupt latencies, operating system overheads, etc, cause inaccuracy and limit the resolution to microseconds on most systems
  - Beware that the frequency of crystal oscillators varies with temperature
    - Supposedly identical systems in different environments observe different clock frequencies; problematic for networked systems
    - Causes observed clock rate to drift over time
- Will discuss clocks and timers more on Thursday

Copyright © 2004 Colin Perkins
http://csperkins.org/

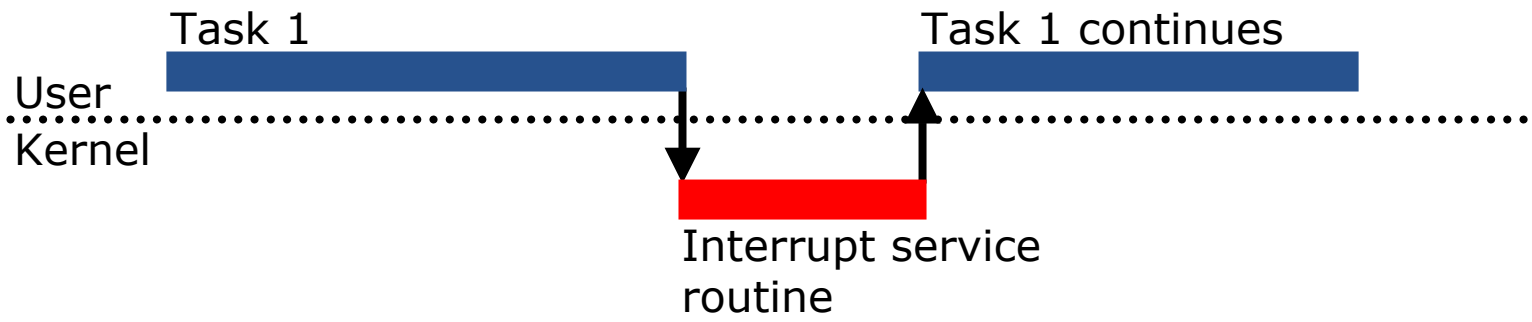
Interrupts and Device Interaction

- In addition to driving the scheduler and timing services, interrupts are often used to signal that hardware devices require attention
  - Useful for devices that require sporadic attention
  - Polling is an alternative, that may be lower overhead, for devices that need periodic attention
- Interrupts have priority, asserted by the hardware
- The system may support more devices than interrupt request lines
  - Require the interrupt service routine to poll multiple devices

Copyright © 2004 Colin Perkins
http://csperkins.org/

Interrupts and Device Interaction

- Interrupts may be serviced immediately
  - Fast response to device
  - Significant delay to currently scheduled task

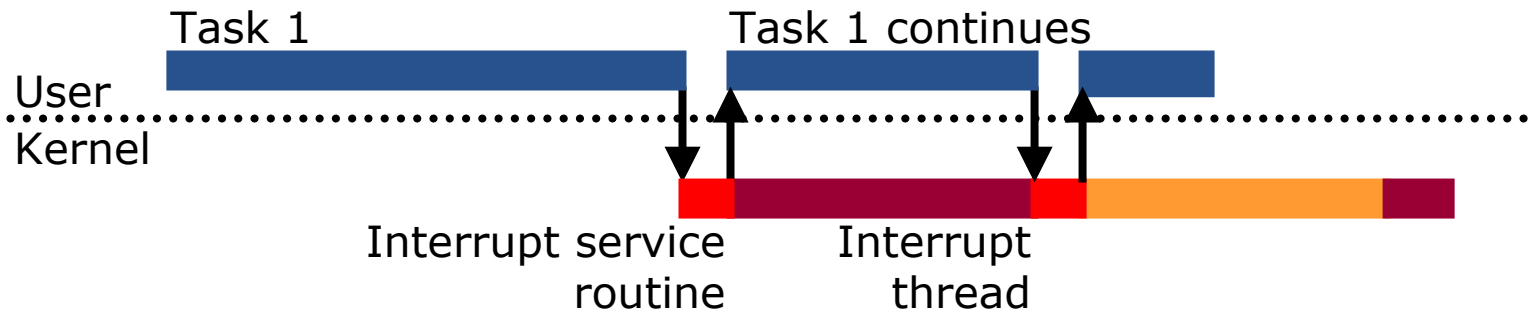


- Interrupts are disabled using the interrupt service routine
  - To safely modify the interrupt management data structures
- Problematic if there are multiple interrupt sources, since high priority interrupts can be blocked

Copyright © 2004 Colin Perkins
http://csperkins.org/

Interrupt Threads

- Alternatively, an interrupt service routine may schedule a kernel thread to complete the interaction
  - Slower to service the device
  - Potentially less impact on interrupted task, although this depends on the priority of the kernel thread



- Reduces the blocking during the interrupt service routine, and allows high priority interrupts to activate
  - Threads servicing the high priority interrupt run with high priority, and pre-empt those servicing other interrupts

Copyright © 2004 Colin Perkins
http://csperkins.org/

Interrupts and Device Interactions

- Interrupt latency varies:
  - Depending how many devices must be polled to locate the interrupt source
  - Depending on the tasks executing when the interrupt arrives
  - If multiple interrupts occurs at once
- Affects response time to hardware interrupts
- Affects clock and time service latency
- Implications:
  - Jitter on job release times
  - Inaccuracy on timers

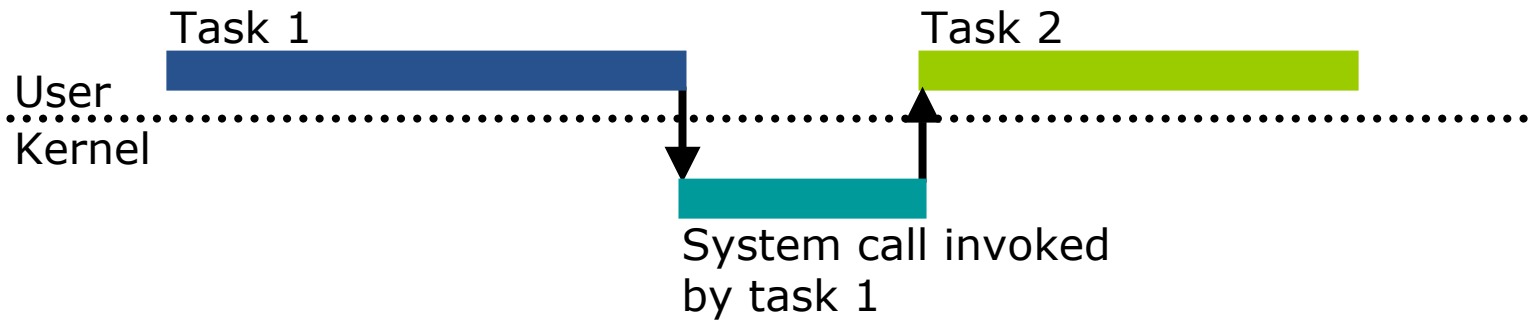
  …will affect scheduling

- Will discuss low level programming, interrupts and device interactions more in lecture 19

Copyright © 2004 Colin Perkins
http://csperkins.org/

Synchronous System Calls

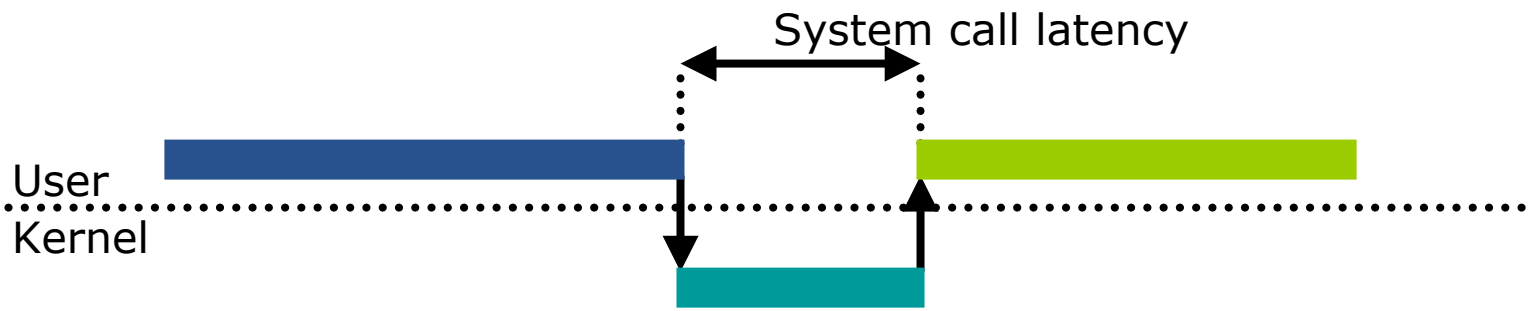
- Tasks invoke the microkernel through **system calls**
- The kernel usually executes in separate a memory space, protected from user tasks
  - A system calls traps to kernel mode, switches protection domains, and saves the context of the user thread
  - Reduces to a function call if the system doesn't implement memory protection
- The call name and arguments are retrieved from the stack, and executed by the kernel on behalf of the thread
- When a system call completes
  - the kernel saves the result
  - switches protection domains and returns to user mode (if necessary).
  - the highest priority thread is executed (may differ from the thread that initiated the call)



Copyright © 2004 Colin Perkins
http://csperkins.org/

Synchronous System Calls

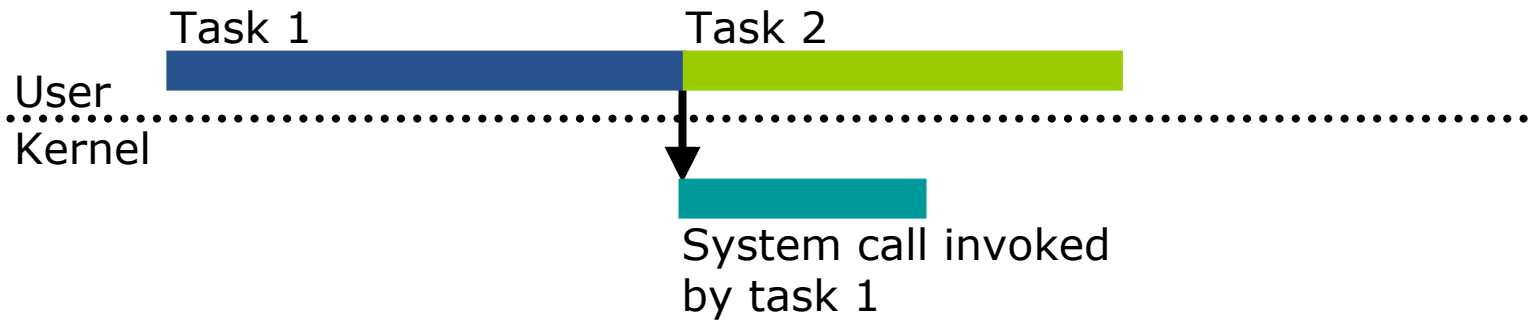
- Some systems cannot be pre-empted during system calls
  - Worst case system call latency must be taken into account when scheduling
    - Systems calls might block for 100s of milliseconds, if they result in access to a hardware device
  - Example: low latency patches for Linux to reduce audio skip



Copyright © 2004 Colin Perkins
http://csperkins.org/

Asynchronous System Calls

- An alternative is an **asynchronous** system call, which asks the kernel to perform an operation and returns immediately
  - Non-blocking networking and I/O functions
  - Inter-process communication
- Such calls generate a **kernel thread** to service the request
  - The kernel thread may execute with various priority values, depending on its use
- Complicates design, since results returned asynchronously



Copyright © 2004 Colin Perkins
http://csperkins.org/

Other Operating System Features

- In addition to basic time and scheduling services, interrupts, system calls many standard operating system features are also useful in real time operating systems:
  - Messaging, Signals and Events
  - Synchronisation and Locking
  - Memory Access and Protection
- Concepts and services often need to be slightly modified from their general purpose equivalents
- Typically implemented as **system service providers** outside the microkernel
  - Operating system literature refers to these as **servers**; we use the term system service provider to avoid confusion with other server tasks scheduled at the application level
  - Allows them to be optional, only implemented when needed

Copyright © 2004 Colin Perkins
http://csperkins.org/

Messaging, Signals and Events

- In addition to typical inter-process communication features, real time operating systems provide:
  - Message queues (synchronous, as an alternative to pipes)
    - Messages delivered in priority order
    - Priority inheritance based on message reception
    - Notification and wakeup on message arrival
  - Asynchronous event notification (application defined signals)
  - Real time signals, priority queued, carrying data

- Messaging, signals and events will be discussed more on Thursday

Copyright © 2004 Colin Perkins
http://csperkins.org/

Synchronisation and Locking

- In addition to mutexes, locks, condition variables and semaphores, a real-time operating system may support:
  - Priority inheritance and restoration
  - The priority ceiling protocol, or the similar ceiling priority protocol
  - Ability to disable priority inheritance to reduce overheads, if not needed

- The synchronisation and locking primitives provided by typical real-time operating systems will be described on Thursday

Copyright © 2004 Colin Perkins
http://csperkins.org/

Memory Access and Protection

- Many embedded real time systems do not use memory protection
  - Any task can directly access any other task's memory
  - Any task can directly access kernel memory
- If you have a closed system that has been proved correct, then memory protection is just unnecessary overhead
  - Time overheads, and unpredictability
    - Context switch overhead
    - Unpredictable access times, especially if virtual memory used
  - Memory overheads
    - Protection provided on a per-page basis, leads to wastage
    - Overhead of maintaining the page tables and protection maps
- Most commercial RTOS provide memory protection as an option, if supported by the hardware
  - Requires the system to fail-safe if an illegal access trap occurs
  - Useful for complex (or reconfigurable) systems

Copyright © 2004 Colin Perkins
http://csperkins.org/

Summary of RTOS Features

- Modular and scalable
  - The microkernel is small
  - Configurable through system service provider routines
- Efficient
  - Low overhead, highly optimised
- System calls and interrupt handling optimised
  - To make the non-premptible regions small and predictable
  - Many actions deferred to kernel threads, with appropriate priority
- Scheduling
  - Fixed priority scheduling, many priority levels, reconfigurable
  - Limited support for deadline scheduling
- Priority inversion control
- Clock and timer resolution on the order microseconds
- Memory management
  - No paging, no cache, protection optional

Copyright © 2004 Colin Perkins
http://csperkins.org/

RTOS Standards

- Operating systems are notoriously non-standard
- IEEE 1003 POSIX®
  - "Portable Operating System Interface"
  - Started as a subset of Unix functionality, various (optional) extensions have been added to support real-time scheduling, signals, message queues, etc.
  - "The name POSIX was suggested by Richard Stallman. It is expected to be pronounced pahz-icks, as in positive, not poh-six, or other variations. The pronunciation has been published in an attempt to promulgate a standardized way of referring to a standard operating system interface."
- Widely implemented:
  - Widespread support in RTOS
  - Partial support in Unix variants
  - Limited support in Linux and Windows
- Will discuss more later in the module…

Copyright © 2004 Colin Perkins
http://csperkins.org/

Example Real Time Operating Systems

Commercial RTOS:

- QNX/Neutrino
- VxWorks
- RTLinux

General purpose systems with real-time extensions:

- Windows NT
- Unix/Linux

Copyright © 2004 Colin Perkins
http://csperkins.org/

Example Real Time Operating Systems

Commercial RTOS:

- QNX/Neutrino
- VxWorks
- RTLinux

Traditional hard RTOS:
- Dedicated real time operating systems
- Provide all the features discussed
- Fully POSIX compliant, with extensions

The book talks about these, and others, in some detail

General purpose systems with real-time extensions:

- Windows NT
- Unix/Linux

Copyright © 2004 Colin Perkins
http://csperkins.org/

Example Real Time Operating Systems

Commercial RTOS:

- QNX/Neutrino
- VxWorks
- RTLinux

- Microkernel hard RTOS
- Supports most features described
- Runs Linux as a low priority server

General purpose systems with real-time extensions:

- Windows NT
- Unix/Linux

Copyright © 2004 Colin Perkins
http://csperkins.org/

Example Real Time Operating Systems

Commercial RTOS:

- QNX/Neutrino
- VxWorks
- RTLinux

General purpose systems with real-time extensions:

- Windows NT
- Unix/Linux

- Suitable for soft real time applications
- Support a subset of RTOS features

Copyright © 2004 Colin Perkins
http://csperkins.org/

Example Real Time Operating Systems

Commercial RTOS:

- QNX/Neutrino
- VxWorks
- RTLinux

General purpose systems with real-time extensions:

- Windows NT
- Unix/Linux

Will be used as examples during the rest of the module…

Copyright © 2004 Colin Perkins
http://csperkins.org/

Summary

By now, you should know:

- The structure of the rest of the module
- An overview of real time operating systems, and how they differ from conventional systems
- An understanding of some of the sources of timing inaccuracy in real systems

Copyright © 2004 Colin Perkins
http://csperkins.org/