Symbian OS for EE207

John Pagonis, Symbian UK Ltd

02 March 2004

Today’s Menu

- Intro
- Some Symbian OS Kernel Basics
- Software Interrupt Handling
- Hardware Interrupt Handling
- Discussion

Class Bootstrapping ;-)

Interrupts help un–blocking !

Process, Thread ?

Kernel, Driver, Executive ?

Micro-kernel vs Monolithic ?

Memory Protection and multi-tasking ?

Static vs Dynamic linking and shared libraries ?

RTOS vs OS vs embedded OS ?

Purpose of this lecture

- to re-enforce material covered this year
- to present an example of a non-trivial OS that uses the ARM processor
- to demonstrate how the knowledge from your course is immediately applicable ”out-there”

The ARM is Out-There ;-)

more than 80% of all mobile phones run on ARM !!

there is at least 1 ARM processor sold every second

since Nov’04 there are at least 1 million Symbian OS phones sold every month...

all run on ARM

some phones have 3 or more ARM processors !!!!

many modern hard disks have 2 ARM7 CPUs !!

Symbian OS kernel - birds-eye view

Symbian OS v7.0s - not for any newer

lightweight 32bit pre-emptive multi-tasking kernel - uses loadable drivers

follows a hybrid design combining characteristics from both micro-kernel and monolithic kernel architectures.

embedded - can execute from ROM

designed for battery-powered mobile devices

designed for resource constrained operation

designed for event-driven user interaction

Mostly written in C++ and a small part in assembly (ARM, x86, M*Core)

Memory protection - Kernel executes in supervisor mode; other programs don’t

Kernel Characteristics

- Message passing between client-server threads
- Networking and Telephony are hosted in user-side servers
- Filesystems are hosted in a user side server
- Device drivers are dynamically loaded and run kernel-side
- The scheduler and any scheduling policy is implemented in the kernel
- Memory management is implemented in the kernel
- User-side programs access kernel services through software interrupts

Interrupts

- We use interrupts in order to request attention !

  e.g., baby cries, dog barks, bell rings, kettle whistles, student raises hand

- In the computer domain there exist two kinds of interrupts

  Software Interrupts - synchronous

  Hardware interrupts - asynchronous

- Interrupts are a processor mechanism by which to request attention

Software Interrupts

- are a processor mechanism (feature) – not all processors offer this !
- are used as a controlled pre-programmed path to OS services
- where a program seeks attention from the OS about some processing that it needs done
- design-wise, allow system service encapsulation by the OS, much like C++ methods do (on a different scale of course)
- in most cases are also used to switch from USER mode to SVC mode
- allow us to separate concerns between OS and applications
- called synchronous because they occur as part of a program's execution sequence (e.g., `PC` points to `SWI` instruction after fetch–decode)

Software Interrupts - Symbian OS

In Symbian OS programs never link to the kernel directly but interface to it through a shared library – called the *user library*

that library contains the necessary instructions to interface to the OS and request its services

programs call the user library functions, which are pre-programmed to cause a software interrupt

it is the only mechanism for a program to interface outside it’s process

Software Interrupts - Symbian OS

Executive Calls are the user library calls that allow a user thread to enter a processor privileged mode so that it can access, in a controlled and predefined way, hardware resources or kernel space objects or services.

- we call them *Executive Calls* – some OSs call them *System Calls*
- what an executive call does is to switch control to the kernel executive
- all Executive Calls are implemented in terms of the SWI instruction (on the ARM)

Software Interrupts Processing - Symbian OS

- `SWI` is issued
- CPU looks up operation in address 0x08 of the exception vector
- CPU decodes that operation, which is a branch to the *Interrupt Handler*
- Interrupt Handler checks the exec call type and ID and branches to it
- the executive call type and ID are passed as the `SWI` operand
- the collection of all such exec calls make up what is called the *Executive*

Clarifications

- the Executive is just a *part* of the kernel – it is called Kernel Executive
- because of memory protection, exec calls are the only way for programs to communicate with the OS and potentially other programs
- Symbian OS distinguishes between 2 types of exec calls (fast and slow)
- the counter–part of the Executive is the user library where all exec calls are setup (register parameters, `SWI` operands etc.)

Hardware Interrupts

- a mechanism by which an external to the processor device may request its attention
- by nature hardware interrupts are asynchronous (since they're outside the remit of the interrupted CPU)
- such interrupts are manifested as a signal assertion on a processor's interrupt line (request pin)
- the number of hardware interrupts and lines is determined by the CPU and SoC designs
- they can be used for data I/O, synchronisation or event notification

Hardware Interrupts

- systems make use of *interrupt handling routines* that exist to service each possible interrupt signal, such routines are usually referred to as *ISRs*
- addresses of ISRs are usually contained in an interrupt vector table
- oftentimes there are more hardware interrupt sources than available CPU interrupt request pins (e.g., the ARM has only 2)
- thus several interrupt sources need to share a single interrupt request signal line
- systems therefore needs to determine which interrupt source caused the interrupt and dispatch the relevant handling routine
- to complement the CPU, systems usually use an interrupt controller that is used to identify (and many times) arbitrate interrupts from multiple devices.

Hardware Interrupt Handling - Symbian OS

- Symbian OS has been designed to cater for such cases where a hardware design needs to re-use an interrupt pin between multiple source devices.
- Symbian OS makes use of *interrupt chaining*
- ISRs that correspond to the same interrupt signal, but different source, are chained together to form a singly linked list
- Symbian OS does not allow for *nested interrupts*
- Symbian OS does have the notion of *interrupt prioritisation*

Hardware Interrupt Handling - Symbian OS

- interrupt request signal is raised,
- ARM processor enters IRQ mode and branches to interrupt handler - pointed from address `0x18` of the ARM interrupt vector table
- interrupt handler immediately adjusts the `LR` to point a word lower than it pointed before and stores it together with some more registers on the IRQ mode stack
- handler looks to discover the source of the interrupt by checking the interrupt controller register for pending interrupts
- handler dispatches to the ISR(s), by looking into an interrupt vector table
- in that vector table usually lies a *service chain* of ISRs

Hardware Interrupt Handling - Symbian OS

In Symbian OS because ISRs are have no priorities and nested interrupts are not permitted, they have to be very short in order to avoid blocking other interrupts for too long.

Therefore interrupt handling in Symbian OS is separated in 2 halves

ISRs - Interrupt Service Routines

DFCs - Deferred Function Calls

ISR Responsibilities

- Check whether the interrupt source that it is specifically responsible for servicing has, in fact, a pending interrupt. This is necessary when several interrupt sources share an interrupt signal
- Clear the interrupt bit in the interrupt controller register
- Acknowledge to the device that its interrupt request has been received (note: some devices need this, others dont)
- Do any necessary I/O
- Queue a DFC to continue processing any data if necessary

DFC Responsibilities

- interrupt signal must be fully handled before other interrupt signals can be serviced
- perform processing that would otherwise be impossible or inappropriate inside the service routine e.g. :

  call general Kernel functions

  signal a thread (the kernel now is in a known state and the ready list of tasks can be indirectly altered)

  access any previously allocated memory and existing data structures

Questions ?