COP 4600 – Summer 2014

Introduction To Operating Systems

Embedded Operating Systems

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cop4600/sum2014

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida

Embedded Operating Systems

- One of the more important and widely used categories of operating systems is embedded operating systems.
- The embedded system environment places unique and demanding requirements on the OS and calls for design strategies a quite different than that found in ordinary operating systems.
- We'll examine an overview of the concept of embedded OS and then look more closely at the principles of embedded OS.

Embedded Systems

- The term embedded system refers to the use of electronics and software within a product, as opposed to a general-purpose computer, such as a laptop or desktop system.
- More formally, an embedded system is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a dedicated function.
- In many cases, embedded systems are part of a larger system or product, as in the case of an anti-lock braking system in a car.
- Embedded systems far outnumber general-purpose computer systems in today's world. The table on the next page illustrates the broad range of applications of embedded systems.

Embedded Systems

Market	Embedded Device
Automotive	Ignition system Engine control Brake system
Consumer electronics	Digital and analog televisions Set-top boxes (DVDs, VCRs, Cable boxes) Personal digital assistants (PDAs) Kitchen appliances (refrigerators, toasters, microwave ovens) Automobiles Toys/games Telephones/cell phones/pagers Cameras Global positioning systems
Industrial control	Robotics and controls systems for manufacturing Sensors
Medical	Infusion pumps Dialysis machines Prosthetic devices Cardiac monitors
Office automation	Fax machine Photocopier Printers Monitors Scanners

Embedded Systems

- The range of systems listed in the table on the previous page illustrate the widely varying requirements and constraints of embedded systems.
- Some of these include:
  - Small to large systems, implying very different cost constraints, thus different needs for optimizations and reuse.
  - Relaxed to very strict requirements and combinations of quality requirements, for example, with respect to safety, reliability, real-time, flexibility, and legislation.
  - Short to long life-times.
  - Different environmental conditions in terms of, for example, radiation, vibration, and humidity.

Embedded Systems

- Different application characteristics resulting in static versus dynamic loads, slow to fast speed, computational versus interface intensive tasks, and/or combinations thereof.
- Different models of computation ranging from discrete-event systems to those involving continuous time dynamics (usually referred to as hybrid systems).

- Often embedded systems are tightly coupled to their environment. This can give rise to real-time constraints imposed by the need to interact with the environment.
- Constraints, such as required speeds of motion, required precision of measurement, and required time of duration, dictate the timing of software operations. If multiple activities must be managed simultaneously, this imposes more complex real-time constraints.

Embedded Systems

- The figure on the next page illustrates, in a general sense, the organization of an embedded system.
- In addition to the processor and memory, there are a number of elements that differ from the typical desktop or laptop computer:
  - There may be a variety of interfaces that enable the system to measure, manipulate, and otherwise interact with the external environment.
  - The human interface may be as simple as a flashing light or as complicated as real-time robotic vision.
  - The diagnostic port may be used for diagnosing the system that is being controlled – not just for diagnosing the computer.
  - Special-purpose field programmable (FPGA), application specific (ASIC), or even non-digital hardware may be used to increase performance or safety.
  - Software often has a fixed function and is specific to the application.

Embedded Systems

Software

FPGA/ ASIC

Memory

Auxiliary Systems (power, cooling)

Human interface

Processor

Diagnostic port

A/D conversion

D/A Conversion

Electromechanical backup and safety

Sensors

Actuators

External environment

Possible Organization of an Embedded System

Characteristics Of Embedded Systems

- A simple embedded system, with simple functionality, may be controlled by a special purpose program or set of programs with no other software.
- Typically, more complex embedded system include an OS. Although it is possible in principle to use a general-purpose OS, such as Linux, for an embedded system, constraints of memory space, power consumption, and real-time requirements typically dictate the use of a special-purpose OS designed for the embedded system environment.
- The following few pages describe some of the unique characteristics and design requirements for embedded OS.

Characteristics Of Embedded Systems

Real-time operation

- In many embedded system, the correctness of a computation depends, in part, on the time at which it is delivered. Often, real-time constraints are dictated by external I/O and control stability requirements.

Reactive operation

- Embedded software may execute in response to external events. If these events do not occur periodically or at predictable intervals, the embedded software may need to take into account worst-case conditions and set priorities for execution of routines.

Characteristics Of Embedded Systems

Configurability

- Because of the large variety of embedded systems, there is a large variation in the requirements, both qualitative and quantitative, for embedded OS functionality. Thus, an embedded OS intended for use on a variety of embedded systems must lend itself to flexible configurations so that only the functionality needed for a specific application and hardware suite is provided.
- For example, the linking and loading functions can be used to select only the necessary OS modules to load. Conditional compilation can be used.
- However, verification of the end product can be a problem for designs with a large number of derived tailored operating systems.

Characteristics Of Embedded Systems

<u>I/O device flexibility</u>

- There is virtually no device that needs to be supported by all versions of the OS, and the range of I/O devices is large.
- Some effort has been expended in developing special tasks to deal with relatively slow devices such as disks and network interfaces instead of integrating their drives into the OS kernel.

Characteristics Of Embedded Systems

Streamlined protection mechanism

- Embedded systems are typically designed for a limited, well-defined functionality. Untested programs are rarely added to the software. After the software has been configured and tested, it can be assumed to be reliable.
- Thus, apart from security measures, embedded systems have limited protection mechanisms. For example, I/O instructions need not be privileged instructions that trap to the OS; tasks can directly perform their own I/O.
- Similarly, memory protection mechanisms can be minimized .

Characteristics Of Embedded Systems

Direct use of interrupts

- General purpose OS typically do not permit any user process to use interrupts directly.
- In embedded systems, there are three legitimate reasons why it would make sense to allow an interrupt to directly start or stop tasks (by storing the task's start address in the interrupt vector address table), rather than going through OS interrupt service routines: (1) embedded systems can be considered to be thoroughly tested, with infrequent modifications to the OS or application code; (2) protection is not necessary (see previous page); and (3) efficient control over a variety of devices is required.

Approaches To Embedded Systems

- There are two general approaches to developing an embedded OS.
- The first approach is to take an existing OS and adapt it for the embedded application.
- The other approach is to design and implement an OS intended solely for embedded use.

Adapting An Existing OS

- An existing commercial OS can be used for an embedded system by adding real-time capability, streamlining operations, and adding necessary functionality.
- This approach most commonly today makes use of Linux, but FreeBSD, Windows, and other OS have been used.
- Such OS are generally slower and less predictable than a special-purpose embedded OS.
- Once advantage of this approach is that the embedded OS derived from the general-purpose OS is based on a set of familiar interfaces, which facilitates portability.

Adapting An Existing OS

- The main disadvantage of using a general-purpose OS is that it is not optimized for real-time and embedded applications.
- Considerable modification may be required to achieve adequate performance.
- In particular, a typical OS optimizes for the average case rather than the worst case for scheduling, usually assigns resources on demand, and ignores most, if not all, semantic information about an application.

Purpose-Build Embedded OS

- A significant number of OS have been designed from the ground up for embedded applications.
- Two prominent examples are eCos and TinyOS.
- While there can be many characteristics that would defined a purpose-built embedded OS, most would agree that the list shown on the next page would be common in embedded OS with real-time requirements. However, keep in mind that for complex embedded systems, the requirements may emphasize predictable operation over fast operation, necessitating a different a different set of design decisions, particularly in the area of task scheduling.

Characteristics of a Specialized Embedded OS

- Has a fast and lightweight process or thread switch.
- Scheduling policy is real time and dispatcher mode is part of scheduler instead of separate component.
- Has a small size.
- Responds to external interrupts quickly; typical response time would be less than 10 μs.
- Minimizes intervals during which interrupts are disabled.
- Provides fixed or variable sized partitions for memory management as well as the ability to lock code and data in memory.
- Provides special sequential files that can accumulate data at a fast rate.

Characteristics of a Specialized Embedded OS

To deal with timing constraints, the kernel of a specialized embedded OS should:

- Provide bounded execution time for most primitives.
- Maintain a real-time clock.
- Provide for special alarms and timeouts.
- Support real-time queuing disciplines such as earliest deadline first (EDF) and primitives for jamming a message into the front of a queue.
- Provide primitives to delay processing by a fixed amount of time and to suspend/resume execution.

eCos (Embedded Configurable Operating System)

- The eCos (Embedded Configurable Operating System) is an open source, royalty-free, real-time OS intended for embedded applications.
- The system is targets at high-performance small embedded systems. For such systems, an embedded form of Linux or other commercial OS would not provide the streamlined software required.
- The eCos software has been implemented on a wide variety of processor platforms including Intel IA32, PowerPC, SPARC, ARM, CalmRISC, MIPS, and NEC V8xx. It is one of the most widely used embedded OS.
- You can try eCos yourself by downloading it at http://ecos.sourceware.org/

eCos - Configurability

- An embedded OS that is flexible enough to be used in a wide variety of embedded applications on a wide variety of embedded platforms must provide more functionality than will be needed for any particular application and platform.
- For example, many real-time OS support task switching, concurrency controls, and a variety of priority scheduling mechanisms. A relatively simple embedded system would not need all of these features.
- The challenge is to provide an efficient, user-friendly mechanism for configuring selected components and for enabling and disabling particular features within components.
- The eCos configuration tool, which will run on Linux or Windows, is used to configure an eCos package to run on a target embedded system.

eCos - Configurability

- The complete eCos package is structured hierarchically, making it easy, using the configuration tool, to assemble a target configuration.
- At a top level, eCos consists of a number of components, and the configuration user may select only those components needed for the target application.
- For example, a system might have a particular serial I/O device. The configuration user would select serial I/O for this configuration, then select one or more specific I/O devices to be supported. The configuration tool would include the minimum necessary software for that support. The configuration user can also select specific parameters such as the default data rate and the size of the I/O buffers to be used.

eCos - Configurability

- This configuration process can be extended down to finer levels of detail, even to the level of individual lines of code. For example, the configuration tool provides the option of including or omitting a priority inheritance protocol.
- The next page illustrates the top level of the eCos configuration tool as seen by the tool user.
  - Each of the items on the list in the left-hand window can be selected or deselected.
  - When an item is highlighted, the lower-right hand window provides a description and the upper right-hand window provides a link to further documentation plus additional information about the highlighted otem.
  - Items on the list can be expanded to provide a finer-grained menu of options.

eCos – Configuration Tool

untitled - eCos Configuration Tool

File Edit View Build Tools Help

- Configuration
  - Redboot for SuperH options
  - Global build options
    - Global command prefix — sh-elf
    - Global compiler flags — -ml -m3 -Wall -Wp
    - Global linker flags — -ml -m3 -ggdb -nc
    - Build GDB stub ROM image
    - Build common GDB stub RC
  - eCos HAL — current
  - I/O sub-system — current
  - Serial device drivers — current
  - Infrastructure — current
  - eCos kernel — current
  - Dynamic memory allocation — current
  - ISO C and POSIX infrastructu — current
  - ISO C library — current
  - Math library — current
  - Wallclock device — current
  - current

Item	Conflict	Property
CYGPKG_HAL_SH_SH77	Unsatisfied	Requires CYG
CYGPKG_HAL_SH_SH77	Unsatisfied	Requires CYG
CYGPKG_HAL_SH_SH77	Unsatisfied	Requires CYG
CYGPKG_HAL_SH_SH77	Unsatisfied	Requires CYGI
CYGINT_HAL_SH_VARIA	Unsatisfied	Requires 1 ==

Property	Value
URL	ref/ecos-ref.html
File	/tmp/untitled_install/include/p
Macro	CYGBLD_GLOBAL_COMMAND_PF
Value	sh-elf
Default	sh-elf
Flavor	data

This option specifies the command prefix used when invoking the build tools.

CYGPKG_HAL_SH_7750, no radio buttons are set

Ready | 5 conflicts

eCos Configuration Tool

Item	Value
eCos HAL	current
Platform-independen	
HAL interrupt handlir	
HAL context switch su	
Explicit control over c	
Source-level debuggir	
ROM monitor support	
Build Compiler sanity	
Common HAL tests	tests/cache tests
ARM architecture	current
Enable Thumb inst	
Enable Thumb inte	
Use big-endian mo	
ARM CPU family	ARM7
Provide diagnostic	
Process all exceptic	
Support GDB threa	
Support for 'gprof'	
Linker script	src/arm.ld
ARM PID evaluatior	current
Startup type	ROM
	38400

eCos Configuration Tool

GNU make utility → Application source code

Application source code → GNU cross compiler

GNU cross compiler → GNU linker

— eCos kernel libraries
— target architecture libraries
→ GNU linker

GNU linker → Executable file

Creating a binary image to execute in the embedded system

eCos - Use

- The previous page illustrates a typical example of the overall process of creating the binary image to execute in the embedded system.
- The process is executed on a source system, such as Windows or Linux, and the executable image is destined to execute on a target embedded system, such as a sensor in an industrial environment.
- At the highest software level is the application source code for the particular embedded application. This code is independent of eCos and makes use of application programming interfaces (APIs) to sit on top of the eCos software.
- There might only be one version of this software, or there might be variations for different versions of the target embedded platform.
- In this particular example, the GNU make utility is used to selectively determine which pieces of a program need to be compiled.

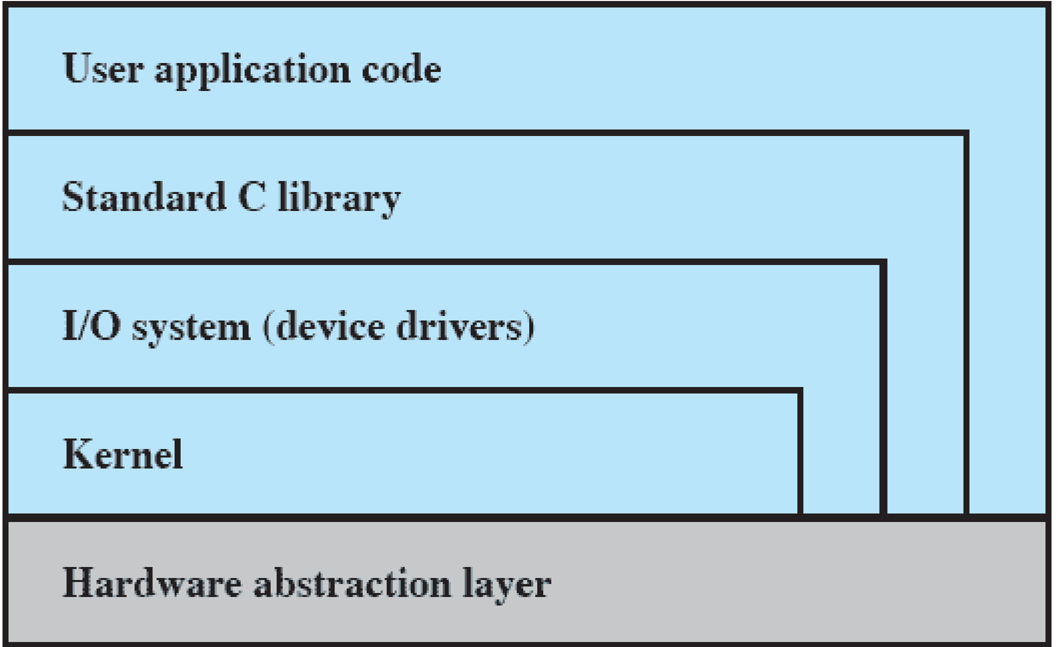
eCos - Use

- The GNU cross compiler, executing on the source machine, generates the executable code for the target embedded platform.
- The GNU linker links the application object code with the code generated by the eCos configuration tool. This latter set of software includes selected portions of the eCos kernel plus selected software for the target embedded system.
- The result is then loaded into the target system.

 © Dr. Mark Llewellyn

eCos - Components

- One of the key design requirements for eCos is portability to different architectures and platforms with minimal effort.
- To meet this requirement, eCos consists of a layered set of components as shown below.



eCos - Components

Hardware Abstraction Layer (HAL)

- At the bottom of the layered design is the hardware abstraction layer (HAL). The HAL is software that presents a consistent API to the upper layers and maps upper-layer operations onto a specific hardware platform.
- Thus, the HAL is different for each hardware platform.
- The next page illustrates how the HAL abstracts hardware-specific implementations for the same API call on two different platforms.
  - In this case, the call from an upper layer to enable interrupts is the same on both platforms, but the C code implementation of the function is specific to each platform.

eCos - Components

```
1 #define HAL ENABLE INTERRUPTS()         \
2     asm volatile (                      \
3         "mrs r3, cpsr;"                 \
4         "bic r3, r3, #0xC0;"            \
5         "mrs cpsr, r3;"                 \
6         :                               \
7         :                               \
8         :   "r3"                        \
9         );                              \
```

ARM Architecture

```
1 #define HAL_ENABLE_INTERRUPTS()              \
2     CYG MACRO START                          \
3     cyg uint32 tmp1, tmp2                    \
4     asm volatile (                           \
5         "mfmsr   %0;"                        \
6         "ori     %1,%1,0x800;"               \
7         "rlwimi  %0,%1,0,16,16;"             \
8         "mtmsr   %0;"                        \
9         : "=r" (tmp1), "=r" (tmp2));         \
10    CYG_MACRO_END                            \
```

PowerPC Architecture

eCos - Components

Hardware Abstraction Layer (HAL)

- The HAL is implements as three separate modules:
- Architecture: Defines the processor family type. This module contains the code necessary for processor startup, interrupt delivery, context switching, and other functionality specific to the instruction set architecture of that processor family.
- Variant: Supports the features of the specific processor in the family. An example of a supported feature is an on-chip module such as a memory management unit (MMU).
- Platform: Extends the HAL support to tightly coupled peripherals like interrupt controllers and timer devices. This module defines the platform or board that includes the selected processor architecture and variant. It includes code for startup, chip selection configurations, interrupt controllers, and timer devices.

eCos - Components

eCos Kernel

- The eCos kernel was designed to satisfy four main objectives:
  - Low interrupt latency: The time it takes to respond to an interrupt and begin executing an ISR (Interrupt Service Routine).
  - Low task switching latency: The time it takes from when a thread becomes available to when actual execution begins.
  - Small memory footprint: Memory resources for both program and data are kept to a minimum by allowing all components to configure memory as needed.
  - Deterministic behavior: Throughout all aspects of execution, the kernel performance must be predictable and bounded to meet real-time application requirements.

eCos - Components

eCos Kernel

- The eCos kernel provides the core functionality needed for developing multi-threaded applications:
  - The ability to create new threads in the system, either during startup or when the system is already running.
  - Control over the various threads in the system; for example, manipulating thread priorities.
  - A choice of schedulers, determining which thread should currently be running.
  - A range of synchronization primitives, allowing threads to interact and share data safely.
  - Integration with the system's support for interrupts and exceptions.

eCos - Components

eCos Kernel

- Some of the functionality that is typically included in the kernel of an OS is not included in the eCos kernel. For example, memory allocation is handled by a separate package. Similarly, each device driver is handled as a separate package.
- Various packages are combined and configured using the eCos configuration tool to meet the requirements of the application.
- This makes for a very small kernel. Indeed, for a very simple single threaded application, it could be run directly on HAL. Such a configuration could incorporate needed C library functions and device drivers, but avoid the space and time overhead of the kernel.

eCos - Components

I/O System

- Some of the functionality that is typically included in the kernel of an OS is not included in the eCos kernel. For example, memory allocation is handled by a separate package. Similarly, each device driver is handled as a separate package.
- Various packages are combined and configured using the eCos configuration tool to meet the requirements of the application.
- This makes for a very small kernel. Indeed, for a very simple single threaded application, it could be run directly on HAL. Such a configuration could incorporate needed C library functions and device drivers, but avoid the space and time overhead of the kernel.

eCos - Components

- Not complete – finish this section