# CENG 448 — Real Time Operating Systems

## Optional Components

Dr. Larry D. Pyeatt

South Dakota School of Mines and Technology

# Objects and Services

A good RTOS implements a very small kernel, and provides the deveoper with the option to add components as needed.

Tasks, semaphores, mutexes, and message queues are essential to an RTOS.

Many RTOS kernels provide additional objects and services

# Optional Kernel Objects

A good RTOS implements a very small kernel, and provides the deveoper with the option to add components as needed.

Tasks, semaphores, mutexes, and message queues are essential to an RTOS.

Many RTOS kernels provide additional objects to provide

- pipes,
- event registers,
- signals,
- condition variables, and
- other services.

# Pipes

Message queues provide exchange of structured data (each item is of some specific type.)

Pipes provide **unstructured** data exchange.

Typically, a pipe is viewed as a **stream of bytes**, and they are usually implemented as part of the file system.

The function to create a pipe returns two descriptors (handles), one for reading and one for writing.

Can be implemented as a message queue of characters, but a custom pipe implementation can be more efficient and powerful.

# Pipes – Part 2

A pipe has two components:

- a pipe control block, and
- a data buffer.

The states for a pipe are the same as the states for a message queue.

The OS may provide *named pipes*, which appear as files in the file system. This allows any task to open, read, and write to the pipe as if it were a file.

# Pipes – Part 3

Typical pipe operations:

| | |
|---|---|
| open | Opens or creates a pipe |
| close | Closes or destroys a pipe |
| read | read one or more bytes from a pipe |
| write | write one or more bytes to a pipe |
| fcntl | allows the pipe to be controlled in various ways, such as setting it for blocking/nonblocking operation, flushing the buffer, etc. |
| select | wait for specific conditions or wait on multiple pipes |

FreeRTOS is unusual in that it does not provide pipes, but *does* provide the select operation on message queues.

# Event Registers

Some RTOSs include an *event register* as part of the Task Control Block (TCB).

The event register is viewed as a set of event flags, which signal the occurrence of specific events.

The task can check its event register to see whether or not a particular event has occurred.

An external source (task or ISR) can also set event flags for the task.

The meaning of each flag may be application-defined.

Tasks may block, waiting on one or more specified event to occur.

# Event Registers – Part 2

Typical event register operations:

| | |
|---|---|
| send | sent an event to a task |
| receive | check whether on not events have occurred, possibly blocking |

# Signals

Similar to an event combined with an interrupt.

A task can register a *signal handler* for each signal.

When a signal occurs, the signal handler is called, asynchronously to the execution of the task.

Tasks can send signals to other tasks.

Tasks do not explicitly receive signals. . . they just happen asynchronously.

The signal control block is part of the TCB.

# Signals – Part 2

Typical signal operations:

| | |
|---|---|
| catch | install a signal handler |
| release | remove a signal handler |
| send | send a signal to another task (or to self) |
| ignore | prevent signal from being delivered |
| block | block a signal temporarily (it goes on a pending list and is delivered later) |
| unblock | allow pending signals to be delivered |

An ISR may perform basic interrupt servicing, then send a signal to a task to finish the processing.

Signals can also be used for synchronization, but are more computationally expensive than other methods.

# Condition Variables

A condition variable is a kernel object that is associated with a shared resource.

It stores the condition (state) of the shared resource, so that all tasks can verify that the resource is in some desired state before it is accessed.

Often implemented as a mutex plus a shared variable. Can be implemented at the Application level, instead of the Kernel level.
Typical condition variable operations:

| | |
|---|---|
| create | Create and initialize a condition variable |
| wait | Wait for condition variable to change |
| signal | Change condition variable and signal one task |
| broadcast | Signals a change to all waiting tasks |

# Optional Kernel Services

A good RTOS implements a very small kernel, and provides the developer with the option to add components as needed.

Optional services may include TCP/IP stack, remote procedure call component, file system, command shell, target debug agent, etc.

These services are more complex than Kernel Objects. They often include ISRs, tasks, and/or library functions.

# TCP/IP Stack

Provides transport-level network communications to support other services, such as Simple Network Management Protocol (SNMP), Network File System (NFS), Secure Sockets Layer (SSL), Telnet, File Transfer Protocol (FTP), Remote Procedure Call (RPC), etc.

Can provide reliable, connection oriented service (Transmission Control Protocol – TCP), or unreliable connection-less service (User Datagram Protocol – UDP).

Can operate over various physical links, such as Ethernet, Frame Relay, ATM, ISDN, RS232, etc.

Often appears to the application as standard Berkeley Socket Interface.

# RPC

First developed at Sun Microsystems in the 1980s, allows for distributed computing.

RPC server allows remote processors to use the local CPU to execute procedures (functions).

RPC client can make RPC calls to remote processors.

The definition of “remote” could be on the same board, or on another planet.

For more information, see this tutorial.

# File System

Provides efficient access to mass storage.

May include local devices, such as flash memory, SSD cards, USB flash drives, hard drives, etc.

May also include remote file systems through NFS, CIFS, or other protocols.

Mass storage devices store blocks of data. The file system organizes them into directories, files, etc.

May also provide

- pipes (anonymous or named)
- locks
- device files
- other services

# Command Shell

Provides an interactive task which allows the user to invoke commands such as `ping`, `ls`, `load`, and `route`.

Commands can be

- executed directly by the shell by making appropriate system calls,
- implemented as loadable program images, or
- dynamically created (and loaded) programs (tasks).

The command shell may even provide a built in programming language, such as BASIC, Lisp, or Forth.

# Target Debug Agent

May be provided through the command shell, TCP/IP, or simple serial connection.

Allows programmer to set up execution and data access breakpoints.

Allows viewing and changing system memory, registers, etc

Allows single-stepping and other program control.

When connected to the development host, typically allows source-level debugging, as well as machine instruction level debugging.

# Other Components

Some RTOS implementations provide even more services, such as SNMP, direct access to I/O devices, C standard library routines, and other high-level components.

In general, the older and more mature an RTOS is, the more optional components it provides.

FreeRTOS is very young, but is growing in power and features at a rapid pace.

# Component Configuration

Most RTOS implementations use two methods to select components.

**Configuration Header** The configuration header file, such as `FreeRTOSConfig.h` allows the programmer to control the components that are included by specifying a set of `#define` statements.

**Makefile** The `makefile` (or other project configuration file) allows the programmer to select source or object code to be included and linked into the project.