RTOS

Real Time Operating System Concepts

Part 2

EmbeddedCraft

crafting of intelligent systems

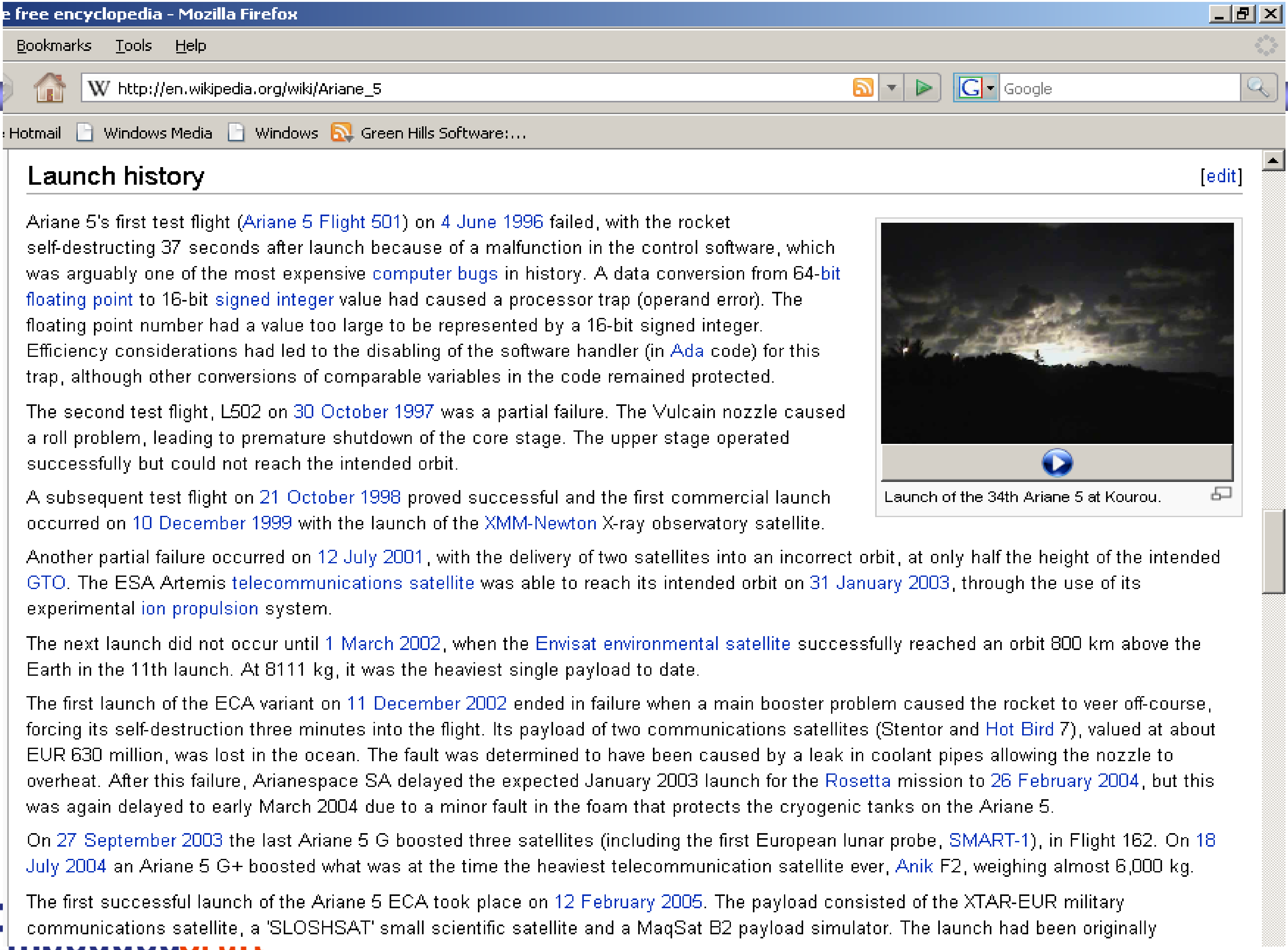
Real time System Pitfalls - 4: The Ariane 5 satelite launch rocket

Rocket self destructed in 4 June -1996.
Exactly after 40 second of lift off at an attitude of 3700 meters, the launcher exploded and became a ball of fire.
Cost: $500 Million USD

Reason:

Bad floating-point exception handling.
A 64 bit floating no is converted into 16 bit signed value.
During one calculation the converted value was more then of 16 bit.

This error is know as “**500 million dollar software error**”



Launch history

[edit]

Ariane 5's first test flight (Ariane 5 Flight 501) on 4 June 1996 failed, with the rocket self-destructing 37 seconds after launch because of a malfunction in the control software, which was arguably one of the most expensive computer bugs in history. A data conversion from 64-bit floating point to 16-bit signed integer value had caused a processor trap (operand error). The floating point number had a value too large to be represented by a 16-bit signed integer. Efficiency considerations had led to the disabling of the software handler (in Ada code) for this trap, although other conversions of comparable variables in the code remained protected.

The second test flight, L502 on 30 October 1997 was a partial failure. The Vulcain nozzle caused a roll problem, leading to premature shutdown of the core stage. The upper stage operated successfully but could not reach the intended orbit.

A subsequent test flight on 21 October 1998 proved successful and the first commercial launch occurred on 10 December 1999 with the launch of the XMM-Newton X-ray observatory satellite.

Launch of the 34th Ariane 5 at Kourou.

Another partial failure occurred on 12 July 2001, with the delivery of two satellites into an incorrect orbit, at only half the height of the intended GTO. The ESA Artemis telecommunications satellite was able to reach its intended orbit on 31 January 2003, through the use of its experimental ion propulsion system.

The next launch did not occur until 1 March 2002, when the Envisat environmental satellite successfully reached an orbit 800 km above the Earth in the 11th launch. At 8111 kg, it was the heaviest single payload to date.

The first launch of the ECA variant on 11 December 2002 ended in failure when a main booster problem caused the rocket to veer off-course, forcing its self-destruction three minutes into the flight. Its payload of two communications satellites (Stentor and Hot Bird 7), valued at about EUR 630 million, was lost in the ocean. The fault was determined to have been caused by a leak in coolant pipes allowing the nozzle to overheat. After this failure, Arianespace SA delayed the expected January 2003 launch for the Rosetta mission to 26 February 2004, but this was again delayed to early March 2004 due to a minor fault in the foam that protects the cryogenic tanks on the Ariane 5.

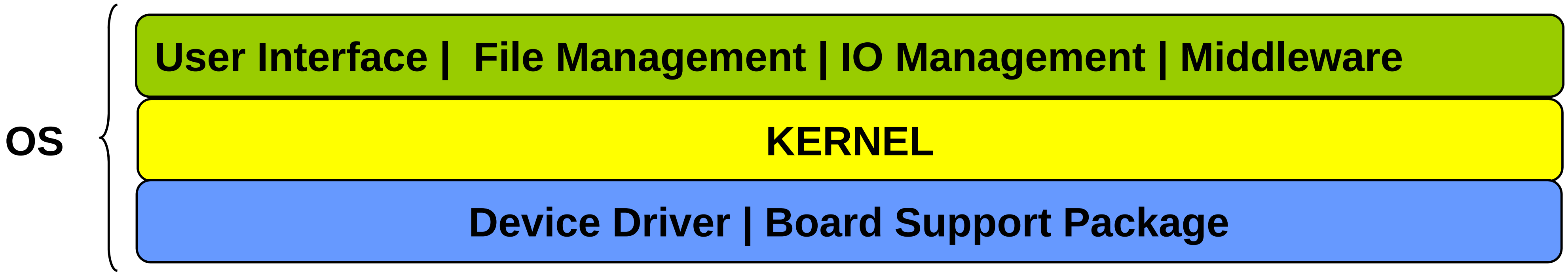
On 27 September 2003 the last Ariane 5 G boosted three satellites (including the first European lunar probe, SMART-1), in Flight 162. On 18 July 2004 an Ariane 5 G+ boosted what was at the time the heaviest telecommunication satellite ever, Anik F2, weighing almost 6,000 kg.

The first successful launch of the Ariane 5 ECA took place on 12 February 2005. The payload consisted of the XTAR-EUR military communications satellite, a 'SLOSHSAT' small scientific satellite and a MaqSat B2 payload simulator. The launch had been originally

Real time System Pitfalls - 5: NASA Mars path finder

NASA mission Pathfinder landed on the mars on 4 july 1997.
The mission was successful , a perfect launch and pathfinder also start sending important data.
But after few days it start resetting itself.

OS overview



OS

User Interface | File Management | IO Management | Middleware

KERNEL

Device Driver | Board Support Package

OS overview: what kernel is doing…

Kernel:

the smallest portion of the operating system that provides

task scheduling,
dispatching,
and intertask communication.

KERNEL

OS overview: what kernel is doing…

The one program running at all times on the computer” is the kernel. Everything else is either a system program (ships with the operating system) or an application program

The kernel is the first part of the operating system to load into memory during booting, and it remains there for the entire duration of the session because its services are required continuously.

Thus it is important for it to be as small as possible while still providing all the essential services needed by the other parts of the operating system and by the various application programs.

KERNEL

Kernel: System Calls

System calls are the similar to the function calls as in C language programs excepts that they are used to access the services provide by kernel to the system

System call also know as API (Application Programming Interface) or Kernel Calls

Ex:-
pthread_create() system call to create task in Linux

CreateTask () system call used to create task in INTEGRITY RTOS

There are also system calls to close the task.

And so on.

Kernel: Kernel Types

Kernel types

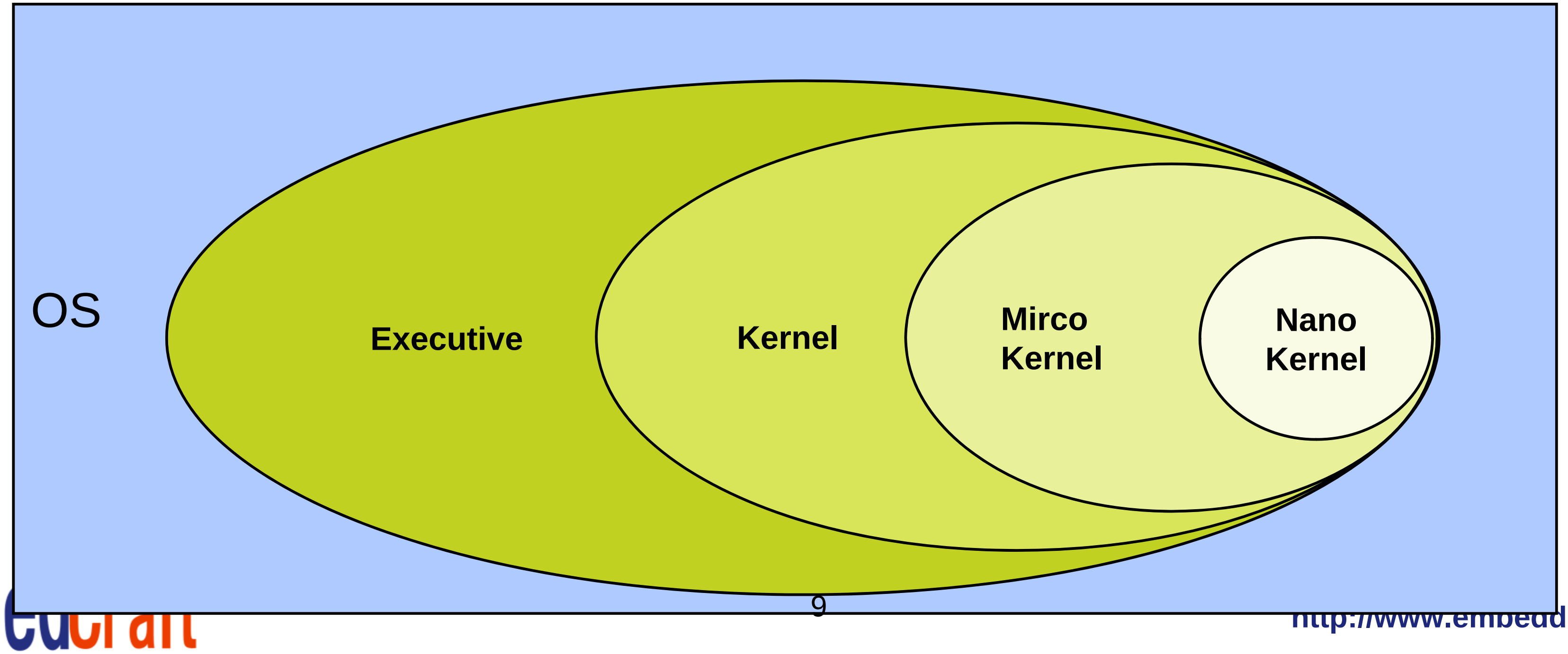
Nanokernel - the dispatcher

Microkernel - a nanokernel with task scheduling

Kernel - a microkernel with intertask synchronization

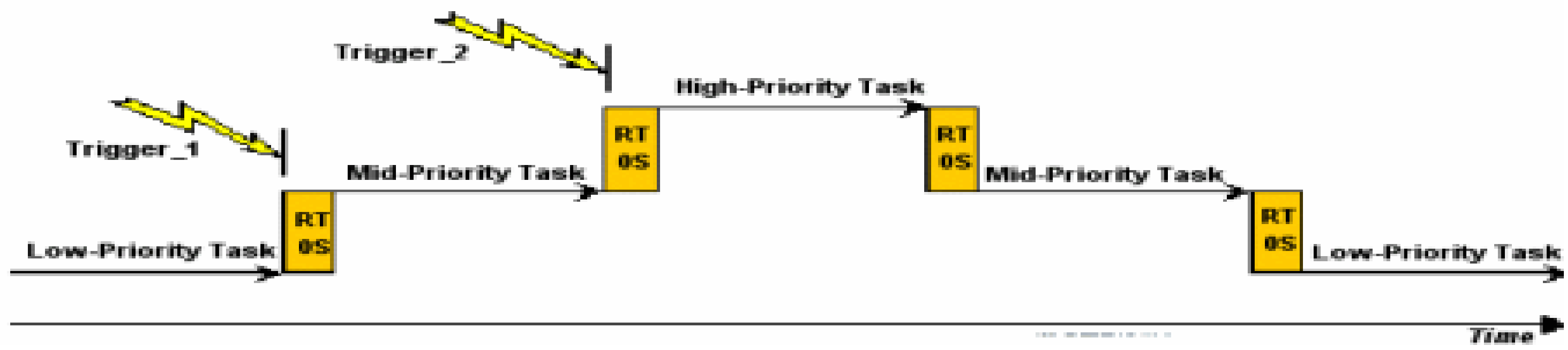
Executive - a kernel that includes privatized memory blocks, I/O services, and other complex issues. Most commercial real-time kernels are in this category.

Operating system - an executive that also provides generalized user interface, security, file management system, etc



Kernel: Scheduling of Task

- Most RTOSs do their scheduling of tasks using a scheme called "priority-based preemptive scheduling
- Each task in a software application must be assigned a priority, with higher priority values representing the need for quicker responsiveness.
- "Preemptive" means that the scheduler is allowed to stop any task at any point in its execution, if it determines that another task needs to run immediately
- Such a type of kernel which support preemptiveness are known as preemptive kernel



KERNEL

Kernel: Synchronization among tasks

-Different tasks may need to share resource and at a particular task it may possible that two task may try to access the same resource.

-If they do so this will result in ambiguous stage.Resource may be any thing like memory or I/O device.

-To avoid this problem it is necessary to lock that resource untill particular task is using that resource, (mutual exclusion should present there)

- solution is Semaphore

Kernel: Semaphore

-A **semaphore** is a variable which is used to access to shared resources in a multitasking environment.

-Semaphore restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore), and can signal that they have finished using the resource (incrementing the semaphore)

- semaphore are of two types mainly
  - Binary Semaphore or Mutex
  - Counting Semaphore

Binary Semaphore or Mutex

-This is used to create the share the one resource.

- when a task access this Semaphore, then priority of semaphore became same as priority of that task

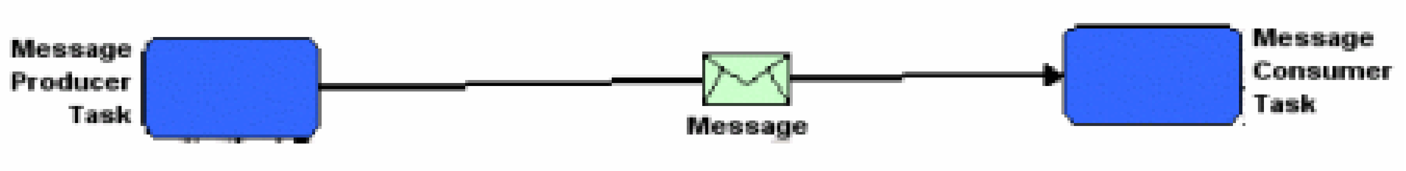
Counting Semaphore

-This is used to create the share the more then one of same type of resource.

- priority method is same as Mutex

Kernel: Intertask Communication

-RTOS offer a variety of mechanisms for communication and synchronization between tasks. These mechanisms are necessary in a preemptive environment of many tasks, because without them the tasks might well communicate corrupted information or otherwise interfere with each other.



-Information may be communicated between tasks in two ways: through global data or by sending messages

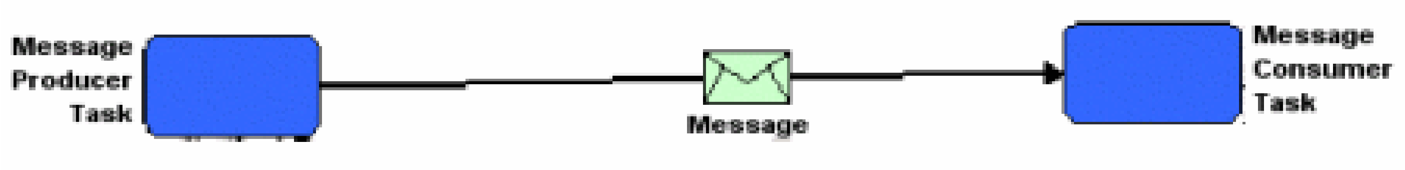
-Message Sending is the main mechanism which is used for that purpose.

Kernel: Message Sending Methods

-Two methods are there

-Message Mailbox

-Message Queues



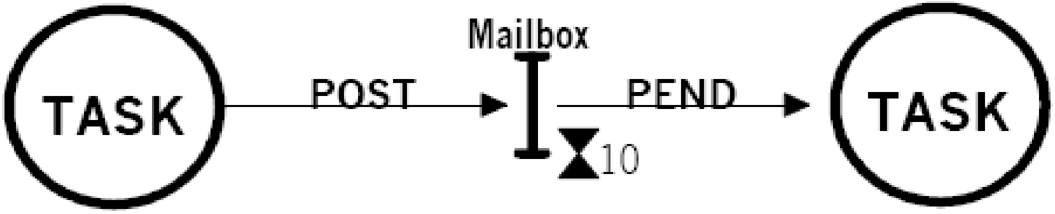
Kernel :Message Mailbox

-Messages can be sent to a task through kernel services.

-A Message Mailbox, also called a message exchange, is typically a pointer size variable. Through a service provided by the kernel, a task or an ISR can deposit a message (the pointer) into this mailbox.

-Similarly, one or more tasks can receive messages through a service provided by the kernel. Both the sending task and receiving task will agree as to what the pointer is actually pointing to.

-A waiting list is associated with each mailbox in case more than one task desires to receive messages through the mailbox.



KERNEL

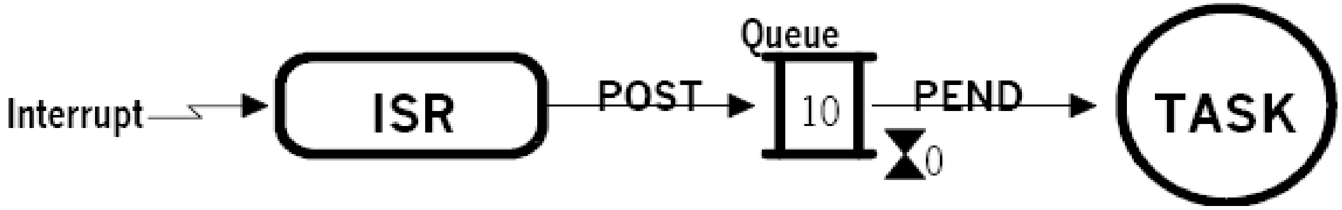
Kernel: Message Queues

-A message queue is used to send one or more messages to a task.

-A message queue is basically an array of mailboxes. Through a service provided by the kernel, a task or an ISR can deposit a message (the pointer) into a message queue.

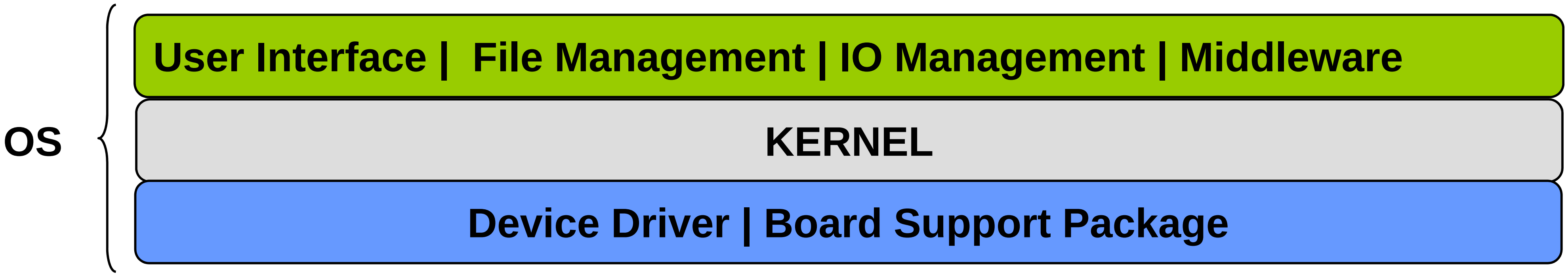
- Similarly, one or more tasks can receive messages through a service provided by the kernel. Both the sending task and receiving task will agree as to what the pointer is actually pointing to.

- Generally, the first message inserted in the queue will be the first message extracted from the queue (FIFO).



KERNEL

OS overview



OS

User Interface | File Management | IO Management | Middleware

KERNEL

Device Driver | Board Support Package

OS Add on: User interface

Two type of interface is possible

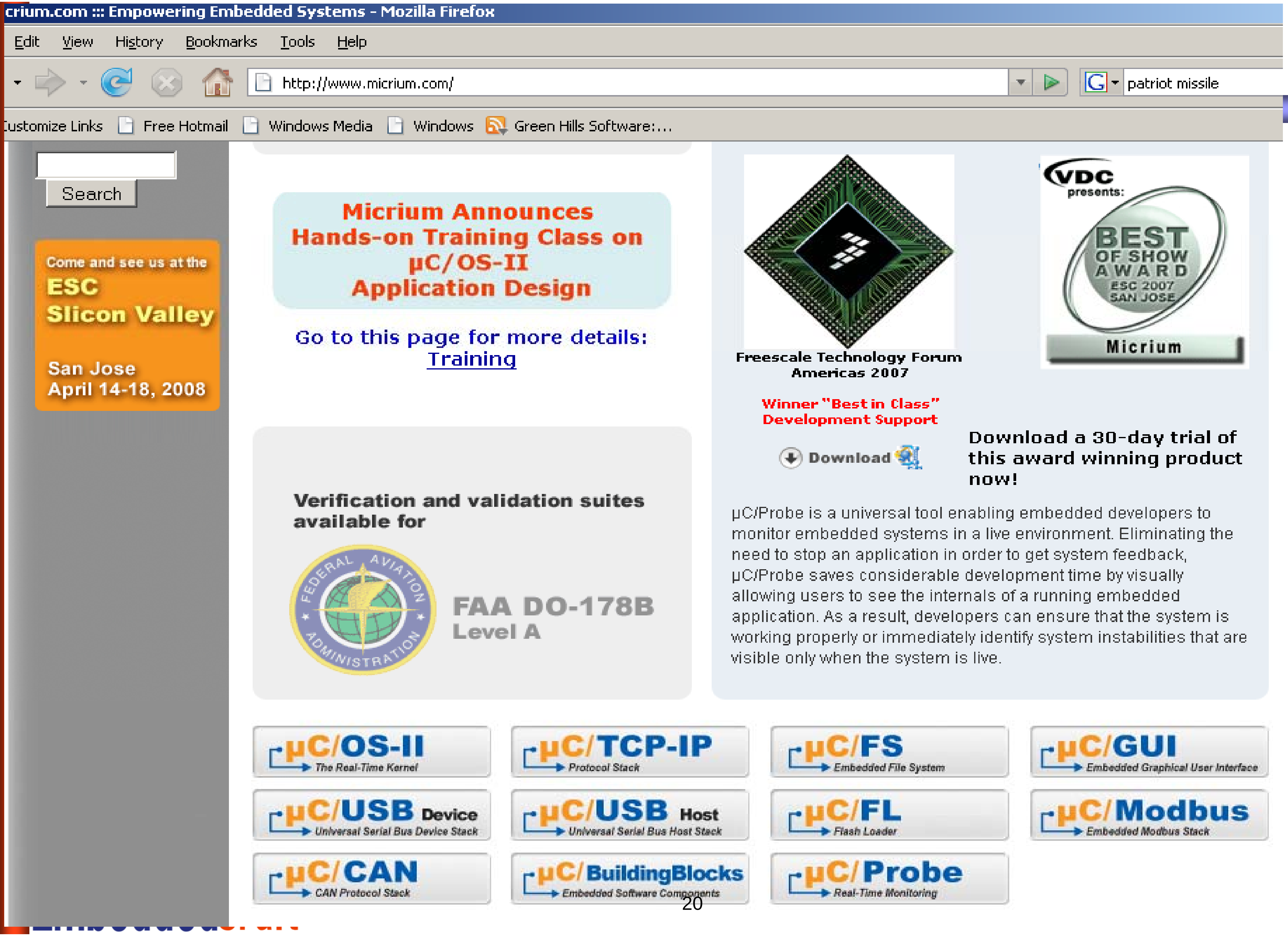
1. Graphical User interface
2. Command Line (through shell)

OS Add on: File Management

Different type of file managements are possible

1. FAT 32
2. Wear Leveling file system (for Flash memory)

Next few slides will give show some commercial file systems provided by companies



Come and see us at the
ESC
Slicon Valley

San Jose
April 14-18, 2008

**Micrium Announces
Hands-on Training Class on
µC/OS-II
Application Design**

Go to this page for more details:
Training

Verification and validation suites available for

FAA DO-178B
Level A

Freescale Technology Forum
Americas 2007

Winner "Best in Class"
Development Support

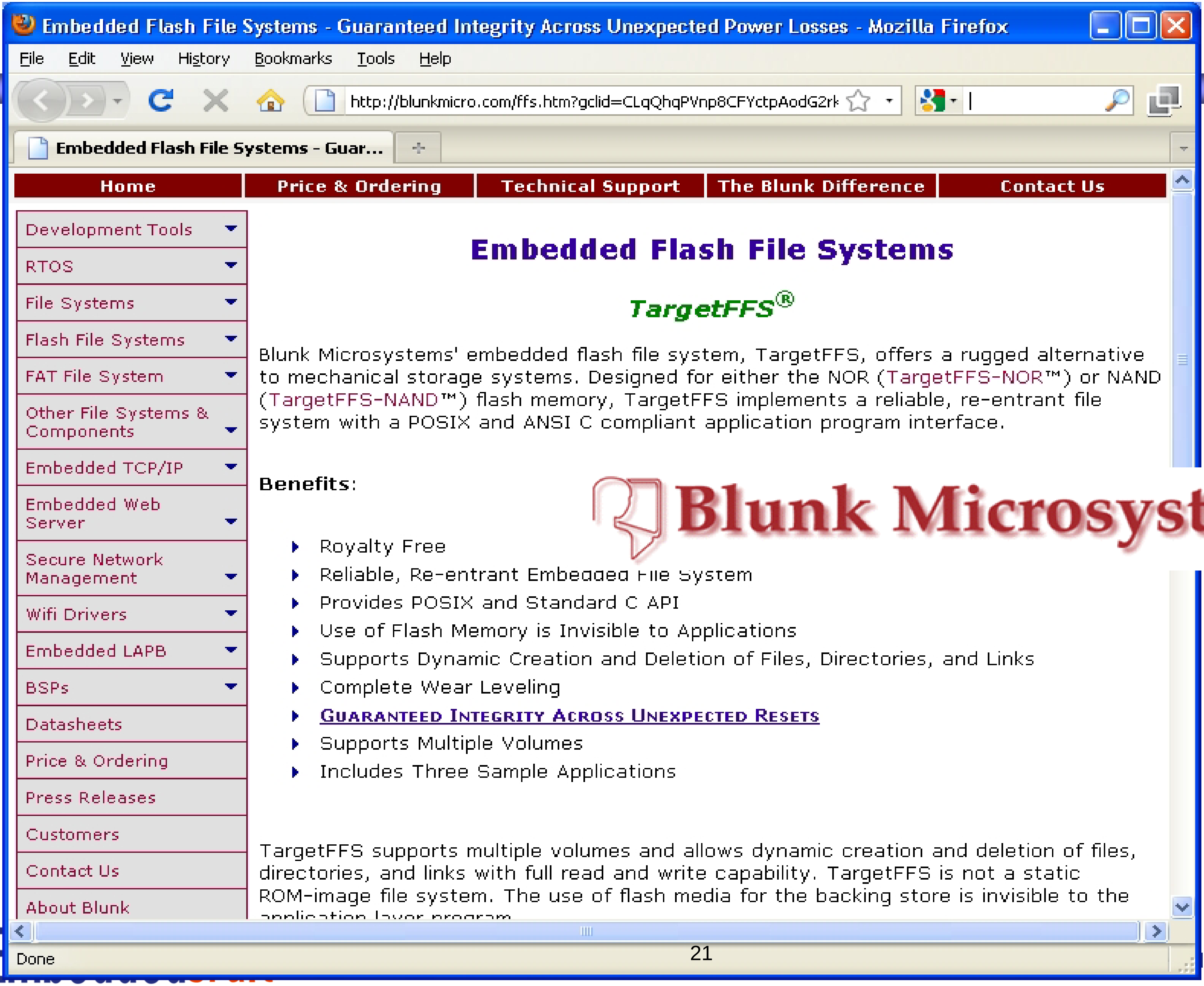
VDC presents:
BEST OF SHOW AWARD
ESC 2007 SAN JOSE
Micrium

Download

Download a 30-day trial of this award winning product now!

µC/Probe is a universal tool enabling embedded developers to monitor embedded systems in a live environment. Eliminating the need to stop an application in order to get system feedback, µC/Probe saves considerable development time by visually allowing users to see the internals of a running embedded application. As a result, developers can ensure that the system is working properly or immediately identify system instabilities that are visible only when the system is live.

µC/OS-II The Real-Time Kernel
µC/TCP-IP Protocol Stack
µC/FS Embedded File System
µC/GUI Embedded Graphical User Interface
µC/USB Device Universal Serial Bus Device Stack
µC/USB Host Universal Serial Bus Host Stack
µC/FL Flash Loader
µC/Modbus Embedded Modbus Stack
µC/CAN CAN Protocol Stack
µC/BuildingBlocks Embedded Software Components
µC/Probe Real-Time Monitoring


Embedded Flash File Systems - Guaranteed Integrity Across Unexpected Power Losses - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://blunkmicro.com/ffs.htm?gclid=CLqQhqPVnp8CFYctpAodG2rk
Embedded Flash File Systems - Guar...
Home
Price & Ordering
Technical Support
The Blunk Difference
Contact Us
Development Tools
RTOS
File Systems
Flash File Systems
FAT File System
Other File Systems & Components
Embedded TCP/IP
Embedded Web Server
Secure Network Management
Wifi Drivers
Embedded LAPB
BSPs
Datasheets
Price & Ordering
Press Releases
Customers
Contact Us
About Blunk
Done

Embedded Flash File Systems

TargetFFS®

Blunk Microsystems' embedded flash file system, TargetFFS, offers a rugged alternative to mechanical storage systems. Designed for either the NOR (TargetFFS-NOR™) or NAND (TargetFFS-NAND™) flash memory, TargetFFS implements a reliable, re-entrant file system with a POSIX and ANSI C compliant application program interface.

Benefits:

Blunk Microsystems

- Royalty Free
- Reliable, Re-entrant Embedded File System
- Provides POSIX and Standard C API
- Use of Flash Memory is Invisible to Applications
- Supports Dynamic Creation and Deletion of Files, Directories, and Links
- Complete Wear Leveling
- GUARANTEED INTEGRITY ACROSS UNEXPECTED RESETS
- Supports Multiple Volumes
- Includes Three Sample Applications

TargetFFS supports multiple volumes and allows dynamic creation and deletion of files, directories, and links with full read and write capability. TargetFFS is not a static ROM-image file system. The use of flash media for the backing store is invisible to the application layer program.

OS Add on: TCP/IP and USB Stack

TCP/IP stacks are also used in RTOS. These stacks are available commercially by software vendors.

Most of the vendors who provide, RTOS also gives TCP/IP

These are various utilities stacks which are added into kernel.

Like

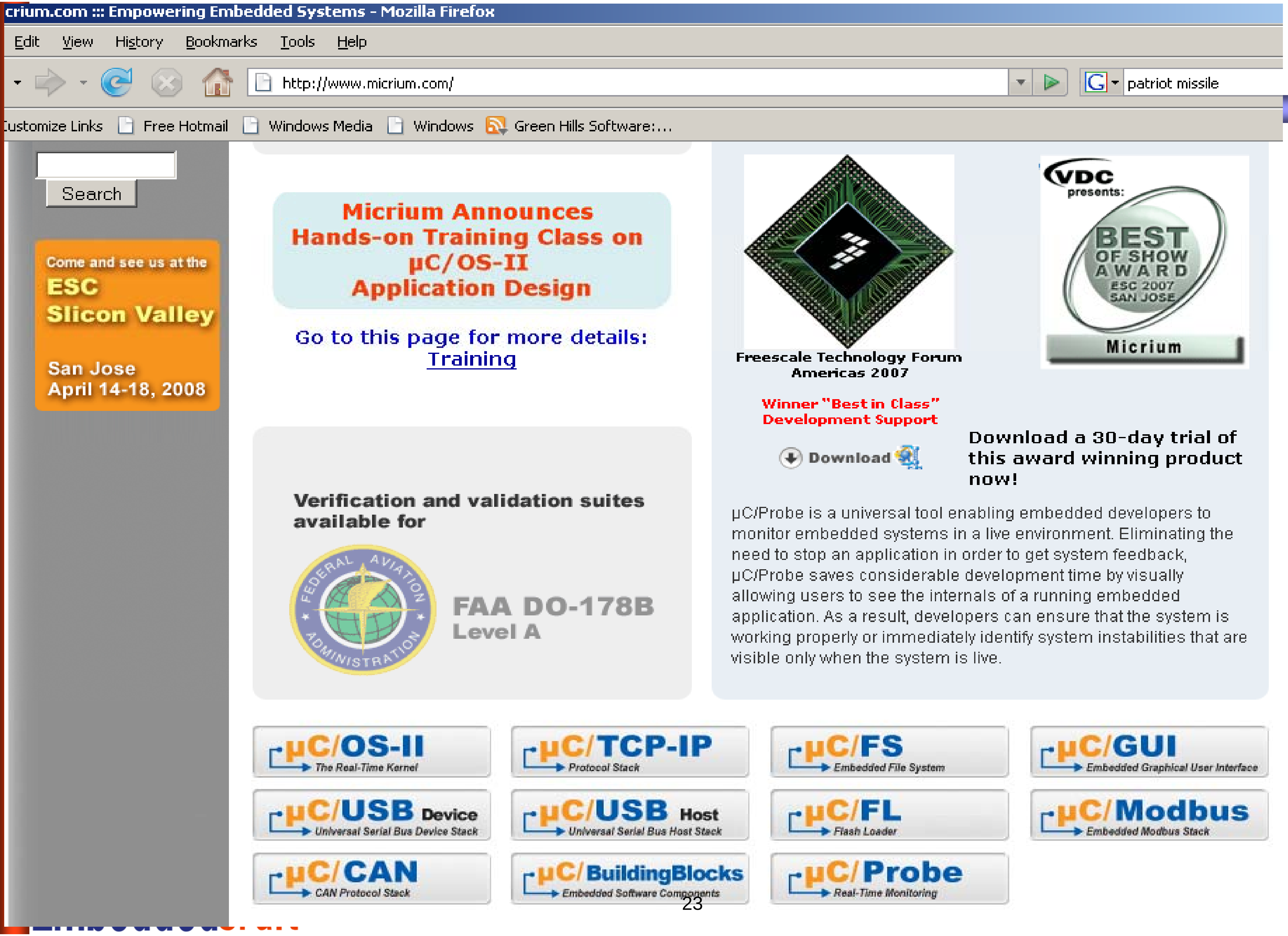
Communication Protocol Stack

USB Protocol Stack

File Servers and various file system

Bluetooth Protocol Stack

Graphics library



Come and see us at the
ESC Slicon Valley
San Jose
April 14-18, 2008

Micrium Announces Hands-on Training Class on µC/OS-II Application Design

Go to this page for more details: Training

Freescale Technology Forum Americas 2007

Winner "Best in Class" Development Support

VDC presents: BEST OF SHOW AWARD ESC 2007 SAN JOSE Micrium

Download

Download a 30-day trial of this award winning product now!

Verification and validation suites available for

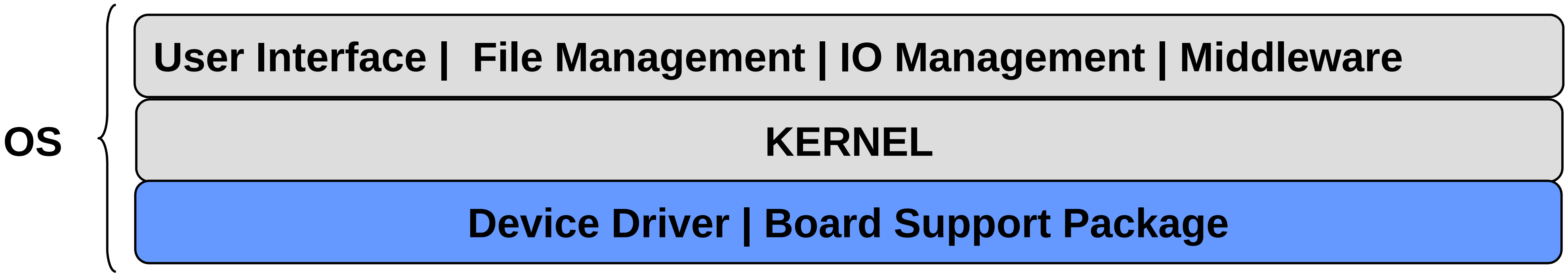
FEDERAL AVIATION ADMINISTRATION

FAA DO-178B Level A

µC/Probe is a universal tool enabling embedded developers to monitor embedded systems in a live environment. Eliminating the need to stop an application in order to get system feedback, µC/Probe saves considerable development time by visually allowing users to see the internals of a running embedded application. As a result, developers can ensure that the system is working properly or immediately identify system instabilities that are visible only when the system is live.

µC/OS-II The Real-Time Kernel
µC/TCP-IP Protocol Stack
µC/FS Embedded File System
µC/GUI Embedded Graphical User Interface
µC/USB Device Universal Serial Bus Device Stack
µC/USB Host Universal Serial Bus Host Stack
µC/FL Flash Loader
µC/Modbus Embedded Modbus Stack
µC/CAN CAN Protocol Stack
µC/BuildingBlocks Embedded Software Components
µC/Probe Real-Time Monitoring

OS overview



OS

User Interface | File Management | IO Management | Middleware

KERNEL

Device Driver | Board Support Package

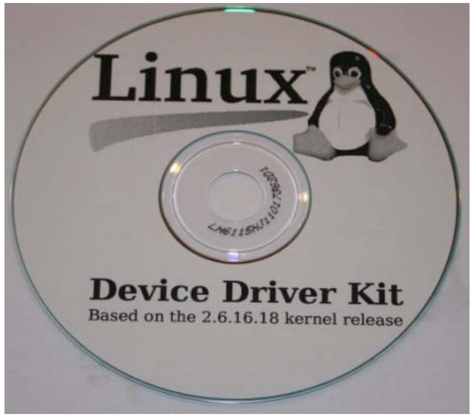
OS Add on: Device Driver

-A **device driver,** or **software driver** is any system that allows other programs to interact with a hardware or peripheral.

-Driver is an interface for communicating with the device, or emulates a device. A driver typically communicates with the device through the bus or communications subsystem that the hardware is connected to.

- When a program invokes a routine in the driver, the driver issues commands to the device, and when the device sends data, the driver invokes routines in the program.

- Drivers are hardware dependent and Operating system specific.



Device Driver | Board Support Package

OS Add on: Board Support Package

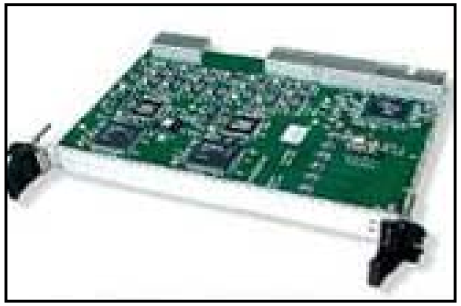
-BPS is board support package.

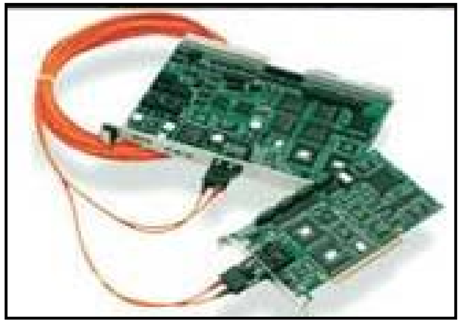
-A “Board Support Package” (BSP) is a set of software that enables a particular operating system to run on a logical board. The operating system features able to be used with the board is directly proportional to the enabling code provided in the BSP

-A BSP typically has to support these CPU, RAM & Flash at a minimum in order for the operating system to function

- For a particular board there is BSP for particular OS.

- For Example for EP8260 board there will be different BSPs for Threadx and different BSP for Linux.





Device Driver | Board Support Package

Why Should I Use an RTOS?

- True that many or most applications can be written without the support of an RTOS.

A few reasons to consider using an RTOS :

- The job of writing application software is generally easier using an RTOS, because the use of a kernel enforces certain disciplines in how your code is structured.
- While the illusion of *concurrency* can be created without the use of an RTOS (though not always), it almost always results in a much more complex piece of software.

But story is not end here….

- In addition to basic scheduling and context switching, a real-time kernel typically provides other valuable services to applications such as:
- **Time Delays**
- **System Time**
- **Inter-Process Communication (IPC)**
- **Synchronization**

Disadvantages of Real-Time Kernels

- Extra cost of the kernel at Software
- More ROM/RAM

EmbeddedCraft
crafting of intelligent systems

http://www.embeddedcraft.org

EmbeddedCraft is the information portal for everyone. This site is useful for those who are working in embedded system domain or start new career in this field.