

MaRTE OS: Overview and Linux Version

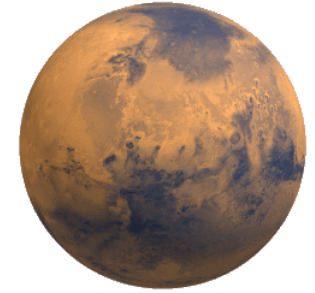
Mario Aldea Rivas (aldeam@unican.es)

**Departamento de Electrónica y Computadores
Universidad de Cantabria (Spain)**

Real-Time Systems Group, York, November 2004.



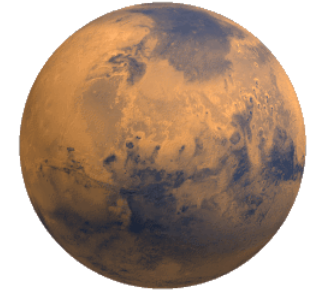
MaRTE OS: Minimal Real-Time Operating System for Embedded Applications



- It is an operating system because:
 - Allows running applications in a bare machine
 - Gives (and controls) access to hardware resources
- Different from general-purpose Operating Systems (Linux, ...)
 - **MaRTE OS** does NOT provide a GUI (nor a shell!)
 - **MaRTE OS** does NOT support a file system
- Main points
 - API for Application-Defined Scheduling
 - Can be used as a POSIX-threads library for Linux
- One result of my PhD
 - Professor Michael González Harbour (Supervisor)

1. MaRTE OS

Main Characteristics



- **Follows the Minimal Real-Time POSIX.13 subset**
 - **Concurrence at thread level**
- **All services with a time-bounded response**
 - **also time-bounded interrupt latency**
- **Single address space shared by kernel and application**
- **Monolithic kernel**
- **Written in Ada (some C and assembler code)**
- **Can execute concurrent Ada and C applications**
- **Portable to different platforms**
- **Free code (GPL). Download and documents:**
<http://martel.unican.es>

Functionality Provided to Applications



Follows the Minimal Real-Time POSIX.13 subset

POSIX (Portable Operating System Interface)

- Very large: inappropriate for embedded real-time systems

POSIX.13: defines four real-time system subsets (profiles)

Profile	File System	Multiple Processes
Minimal	NO	NO
Controlled	YES	NO
Dedicated	NO	YES
Multi-purpose	YES	YES

Minimal: intended for small embedded systems

- no MMU, no disk, no terminal (controller of a “Toaster”)

Functionality Provided to Applications (cont'd)



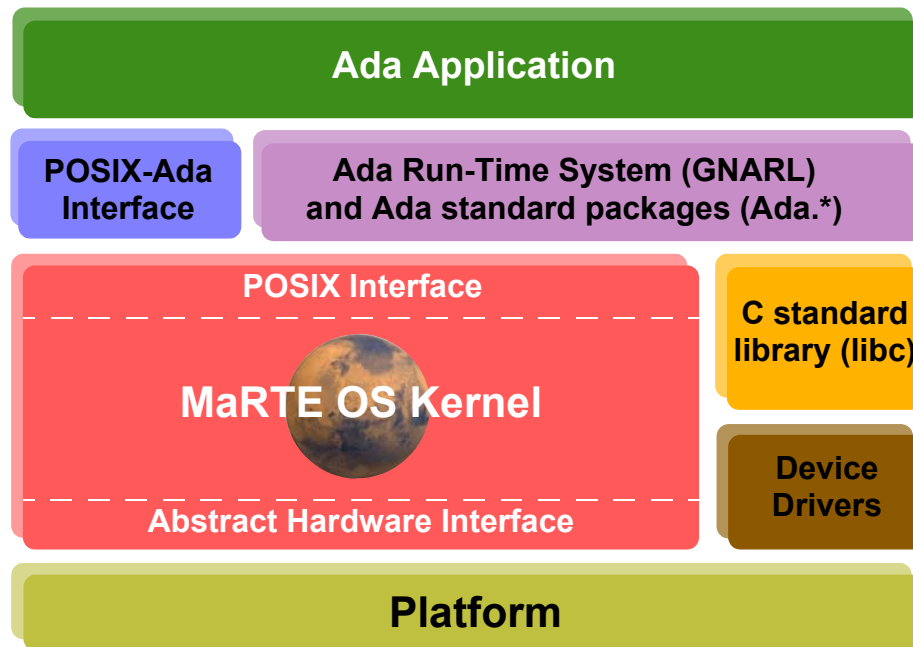
Functionality included in the minimal profile:

- **Threads (policies FIFO, Round-Robin and Sporadic Server)**
- **Mutexes, condition variables, semaphores**
- **Signals**
- **Clocks and timers**
 - **CPU-time clocks and timers**
- **Thread suspension, absolute and relative delays.**
- **Device “files” and device I/O (`open()`, `read()`, `write()`, ...)**

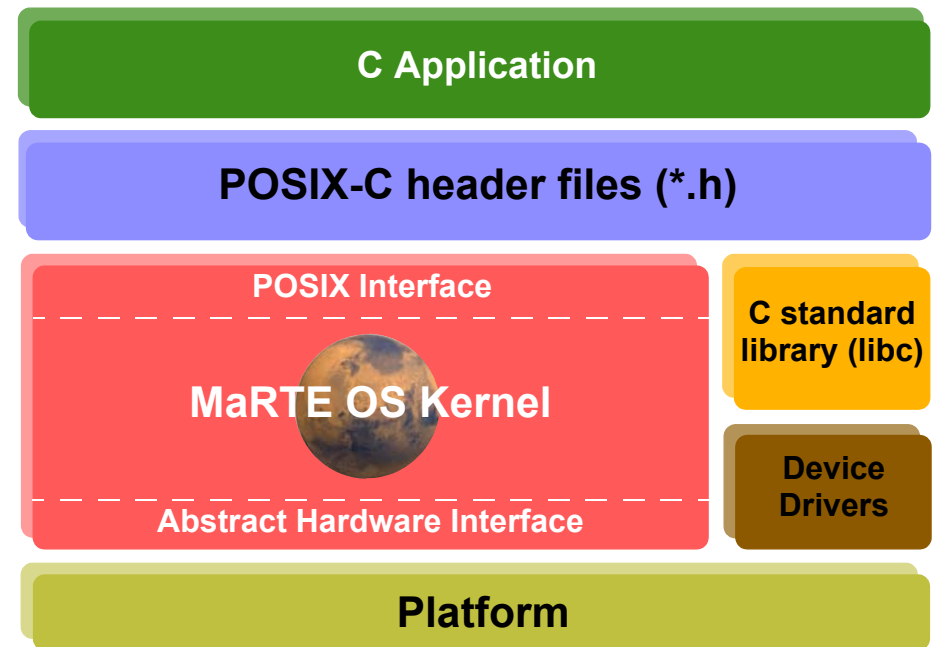
Extra functionality

- **Hardware interrupts management**
- **Application-defined scheduling**

Architecture



Ada aplicación executing on MaRTE OS



C aplicación executing on MaRTE OS

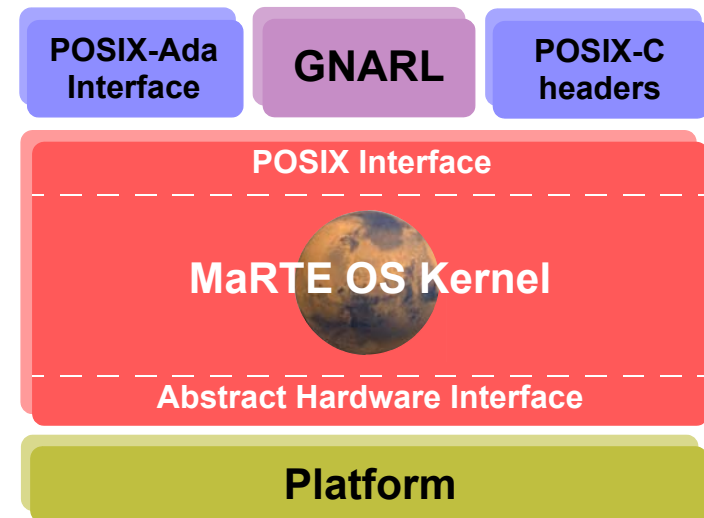
Architecture (cont'd)

Kernel interface developed according to POSIX-C because:

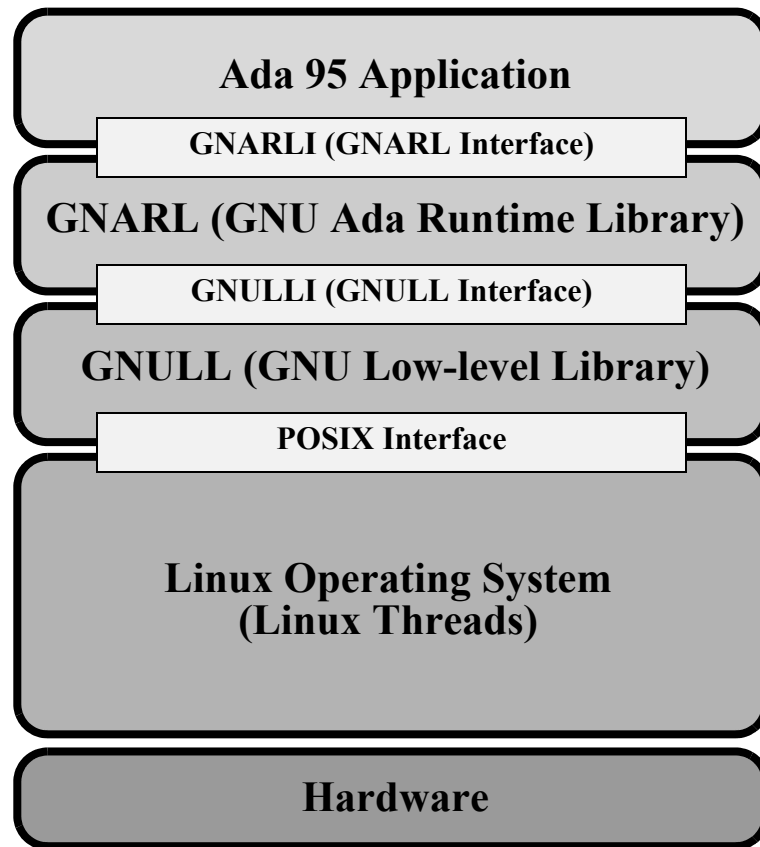
- GNARL is layered on top of a POSIX-thread library
- Ada exceptions cannot be handled by C programs

Abstract Hardware Interface

- simplifies portability
- Clock
- Timer
- Context Switch
- Interrupts management

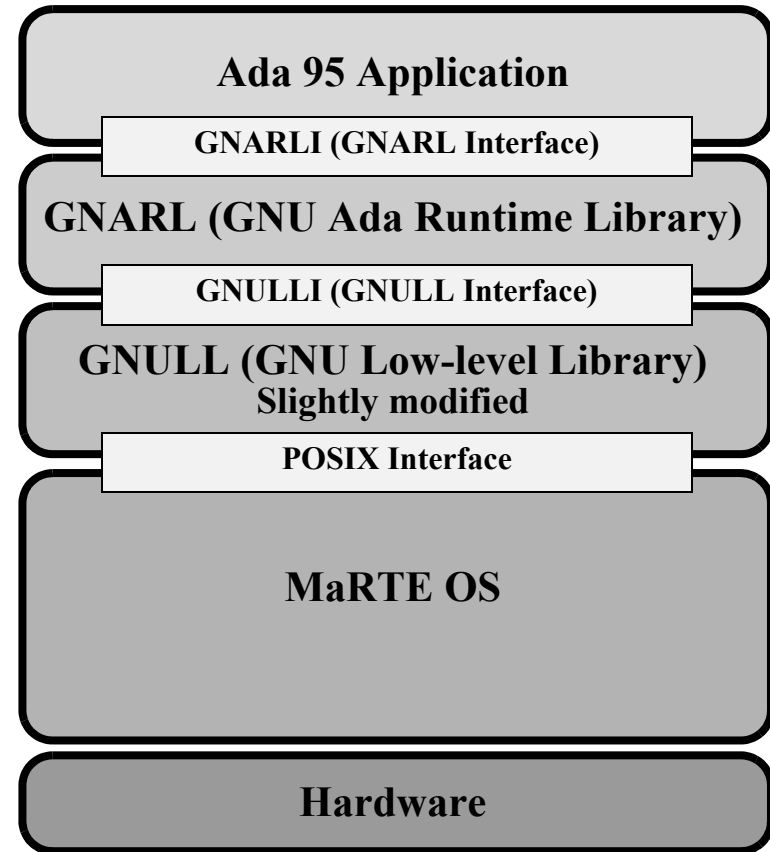


Adaptation of GNAT run-time library



GNARL in Linux

➔
Adapted



GNARL in MaRTE OS

Cross development environment (Architecture x86)

Cross environment.

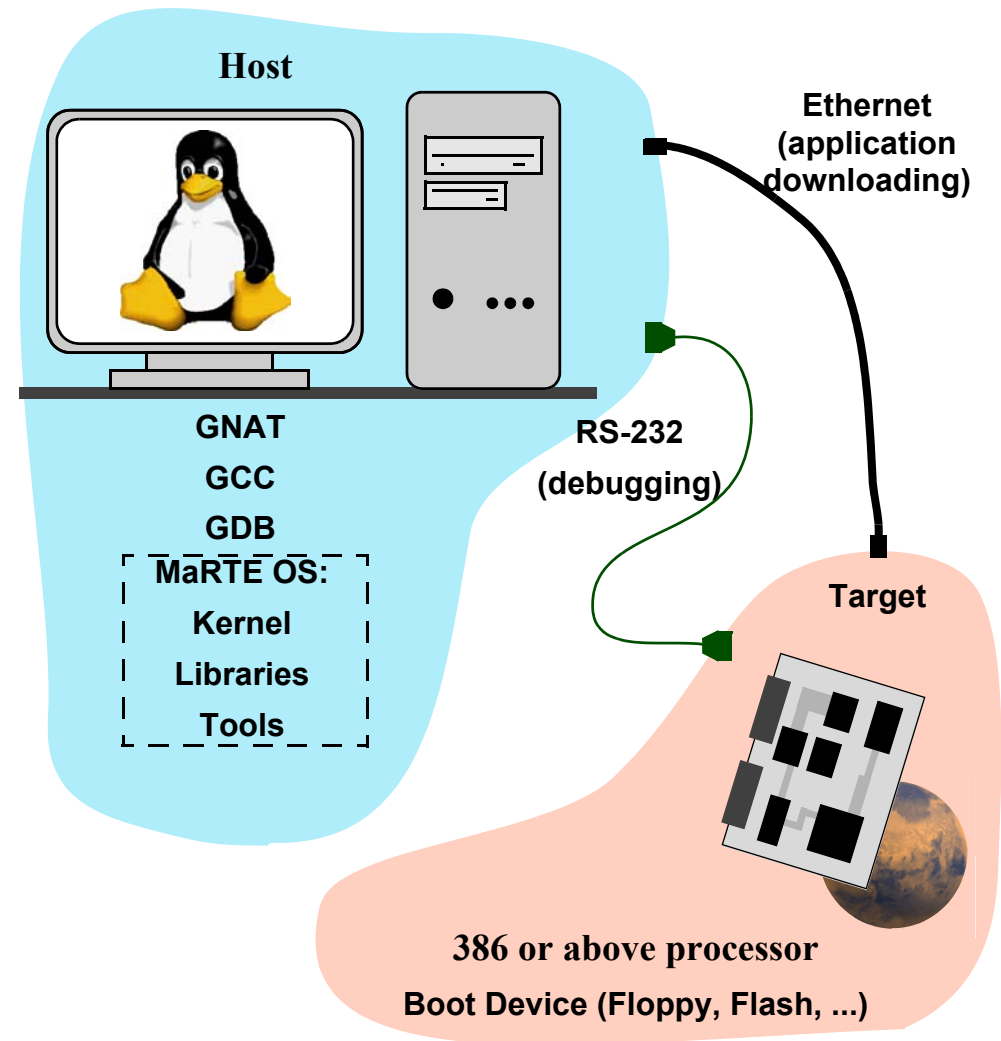
- Host: PC running Linux
- Target: bare 386 PC

Compiling and linking:

- Based on GCC and GNAT
- **mgcc** and **mgnatmake**
`$ mgcc -g any.o prog.c`
`$ mgnatmake prog.adb`

Ethernet used for application downloading

Serial line for debugging



Future Improvements / Limitations

- **Support for real-time distributed applications with RT-GLADE**
 - **Modification of GLADE (GNAT implementation of the Ada 95 DSA) to support real-time requirements**
- **Add Ada'0Y new real-time services (MaRTE OS provides support for some of them)**
 - **Execution-time clocks and group budget control**
 - **Round Robin and EDF scheduling policies**
- **Trace mechanism**
 - **Debugging**
 - **WCET analysis**
- **Hardware interrupt handlers for Ada**
 - **Using protected procedures**

2. New Architectures: MaRTE as a POSIX-threads library for Linux

Three configurations for MaRTE OS

- Architecture **x86_PC**
 - MaRTE OS applications are stand-alone programs to be executed in a bare PC
- Architectures **Linux** and **Linux_lib**
 - MaRTE OS behaves as a Pthread library for Linux
 - concurrence at library level (not using Linux Threads)
 - no super-user privileges to assign priorities to threads
 - Applications are executed as a standard Linux user process
 - Abstract Hardware Interface based in Linux system calls
 - Main idea and first version (super-user) by Miguel Ángel Masmano Tello (Polytechnic University of Valencia, Spain)

Abstract Hardware Interface for Linux and Linux_lib

Linux signals play the role of hardware interrupts

- Install signal handler (install an interrupt handler)
- Modify signal mask (enable or disable interrupts)

Linux Timer instead of a hardware timer

- Linux timer sends a signal to the process when expires

Clock

- Time stamp Counter register (**Linux** architecture)
- Linux system call `gettimeofday()` (**Linux_lib** architecture)

Context Switch routine

- The same as used in **x86** architecture

Architectures Linux and Linux_lib: Linux System Calls

Signals

- `linux_sigaction()`, `linux_sigprocmask()`

Timer

- `linux_setitimer()`, `linux_timer_create()`

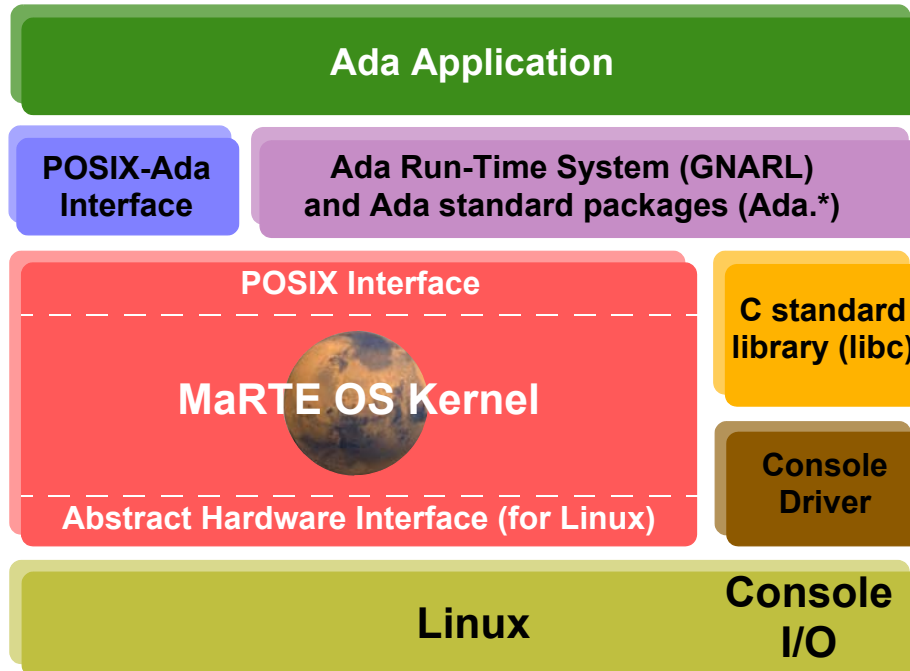
System calls accessible using macros provided by Linux

```
_syscall3(int, sigaction, int, signum,  
          const struct sigaction *, act, struct sigaction *, oldact);  
int linux_sigaction (int signum,  
                    const struct sigaction * act, struct sigaction * oldact);
```

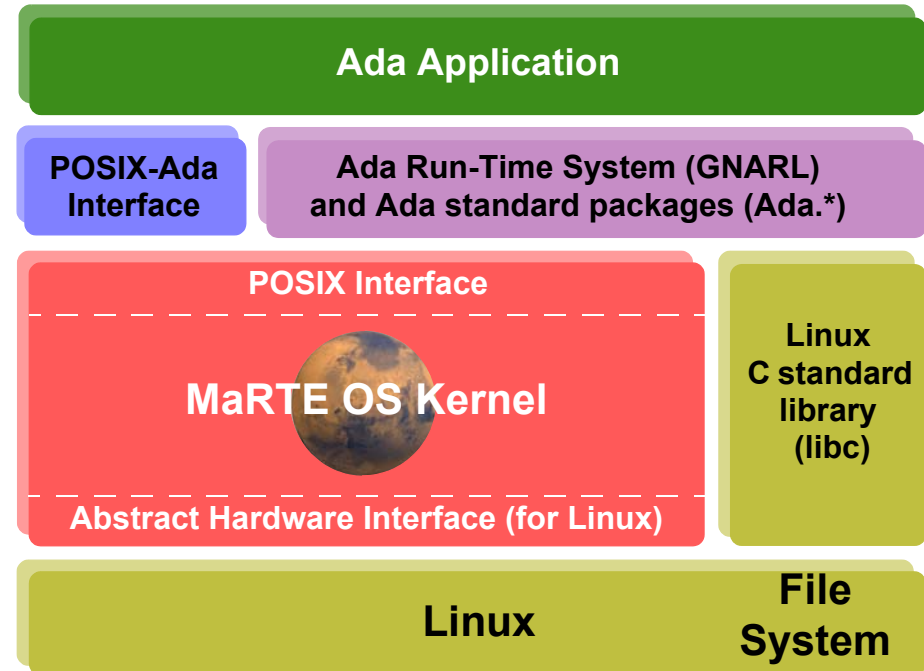
Avoid linking problems

- `sigaction()` is also defined in MaRTE for internal signals (the same for other functions)

Architectures Linux and Linux_lib: Differences

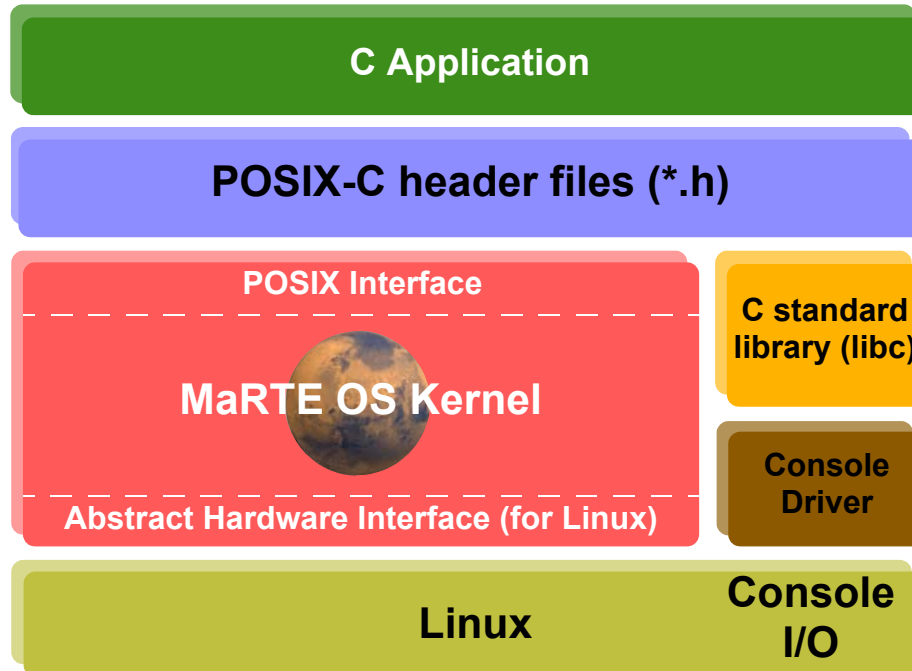


- **Linux Architecture**
 - Uses MaRTE libc and file system
 - Console driver modified to use Linux stdin and stdout

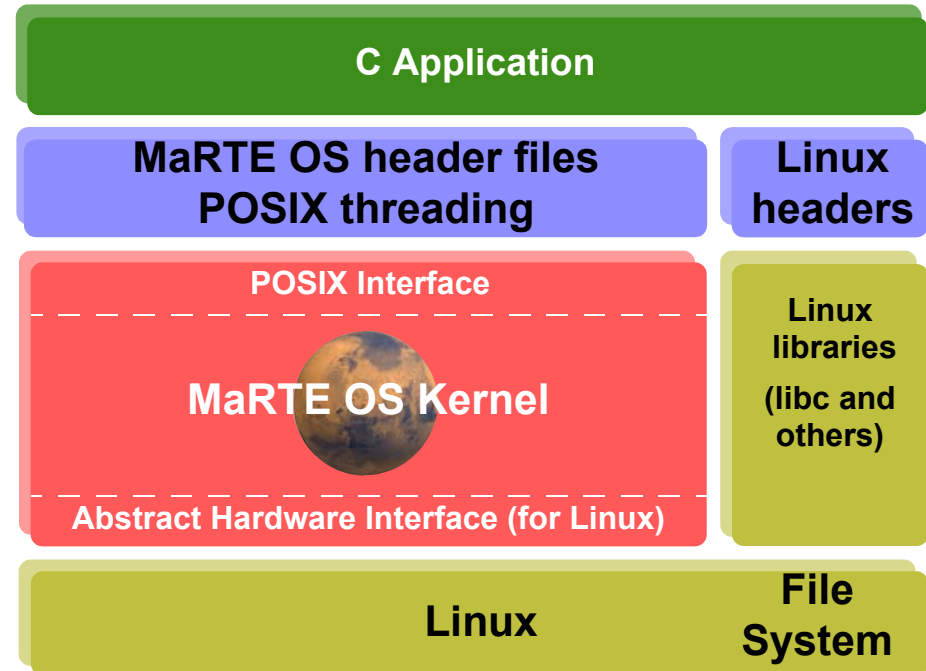


- **Linux_lib Architecture**
 - Uses standard Linux libraries
 - It is possible to access Linux file system

Architectures Linux and Linux_lib: Differences



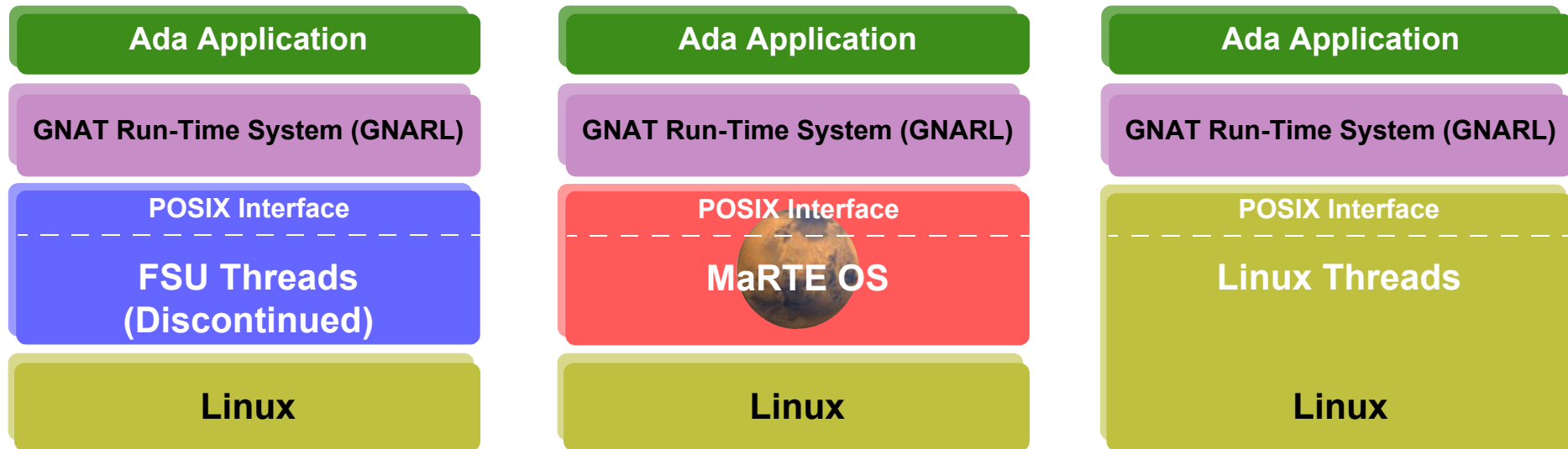
- **Linux Architecture**
 - Uses MaRTE libc and file system
 - Console driver modified to use Linux stdin and stdout



- **Linux_lib Architecture**
 - Uses standard Linux libraries
 - It is possible to access Linux file system

GNAT Specific Issues

- **GNAT can use different Run-Time Systems**
 - **rts-fsu**: based on FSU Threads (discontinued!)
 - **rts-native**: based on Linux threads (super-user privileges)
- **ACT could be interested in a rts-marte**
 - **substitute of rts-fsu for teaching (short term)**
 - **also for bare PC (medium-long term)**



Architectures Linux and Linux_lib

Good choice to teach real-time programming courses:

- **Possible to use priorities and POSIX and Ada real-time scheduling policies in a standard Linux box**
 - **no super-user privileges required**
- **Possible to use advanced features in RT POSIX courses**
 - **CPU-time clocks and timers, RR and SS policies, ...**

Testing

- **Simple and fast mechanism to perform functional testing of applications without using the final embedded system**

Hard real-time behaviour cannot be achieved

- **MaRTE applications are scheduled by the Linux scheduler**
- **Affected by memory swapping, Linux kernel activities, etc.**

Future Improvements

- Convert **MaRTE Linux_lib** in a new GNAT run-time system
 - More easy to install and use
 - Required for ACT in order to distribute it along with GNAT
 - Easy: just move files and create libraries (almost done)
- Port MaRTE to Windows and Solaris
 - GNAT is also distributed for that operating systems
 - No experience

Known Limitations (are they important for teaching?)

- **Problems using MaRTE and Linux header files together**
 - Types (`pthread_t`, `pthread_attr_t`, `pthread_mutex_t`, etc.) could be redefined
 - Can be solved in the most common cases
- **CPU-time**
 - Context switches between processes are not considered
- **Not possible to use Linux signals directly**
 - `sigaction()`, `sigprocmask()`, etc. act on “internal” MaRTE signals
- **Blocking I/O operations**
 - When a task blocks in an I/O operation the whole application blocks