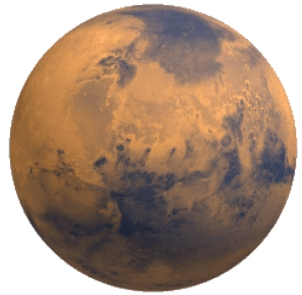
Novedades en MaRTE OS: Librería de Pthreads para Linux[1] y Java de Tiempo Real[2]

Mario Aldea Rivas (aldeam@unican.es)[1][2]
Michael González Harbour (mgh@unican.es)[1]
Dpto. Electrónica y Computadores, Universidad de Cantabria

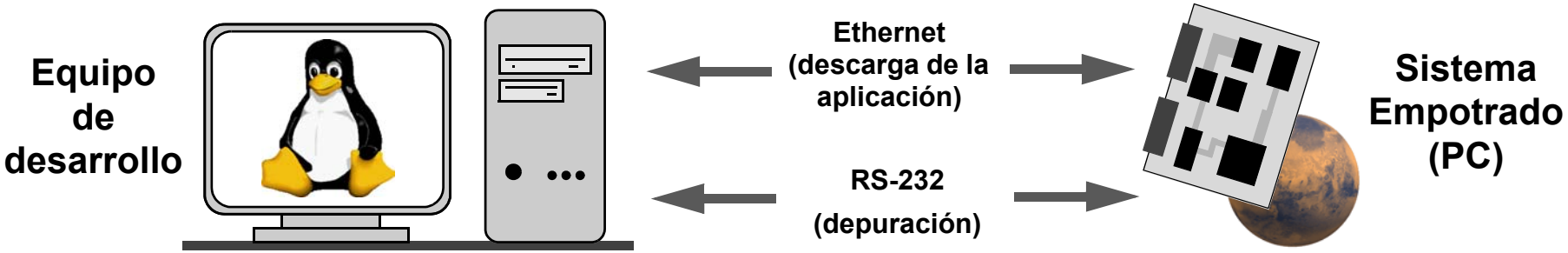
Andy Wellings (andy@cs.york.ac.uk)[2]
Department of Computer Science, University of York

VIII Jornadas de Tiempo Real. Bilbao, febrero de 2005.

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

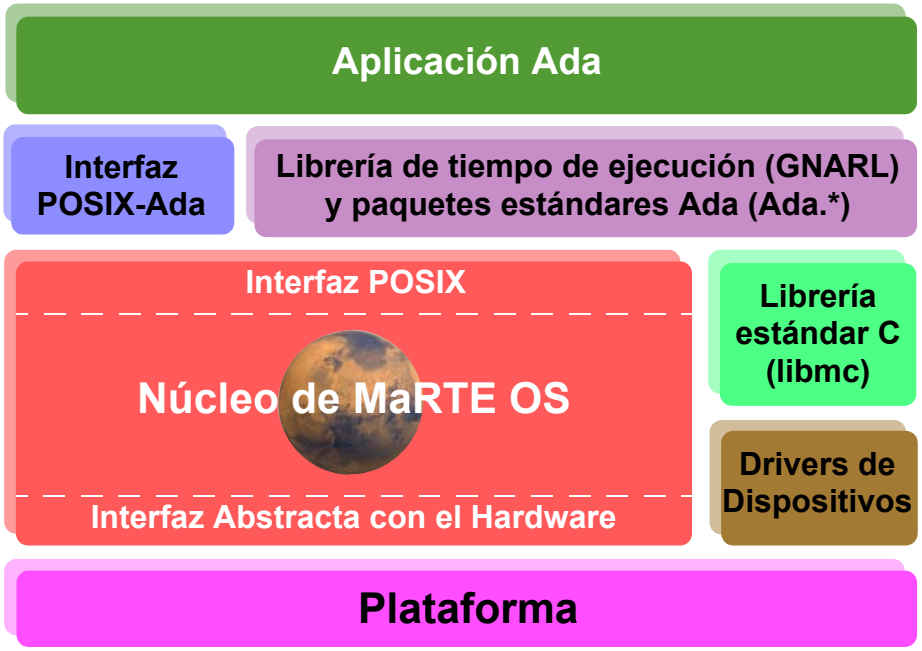


- Sigue el perfil "sistema mínimo de tiempo real" (POSIX.13). Concurrencia a nivel de threads:
  - Threads (FIFO, Round-Robin y Servidor Esporádico)
  - Mutexes, Variables Condicionales, Semáforos, Señales
  - Relojes y Temporizadores (también de tiempo de CPU)
  - Ficheros de dispositivo (`open()`, `read()`, `write()`, ...)
- Entorno de desarrollo cruzado (*ahora hay otros entornos)*

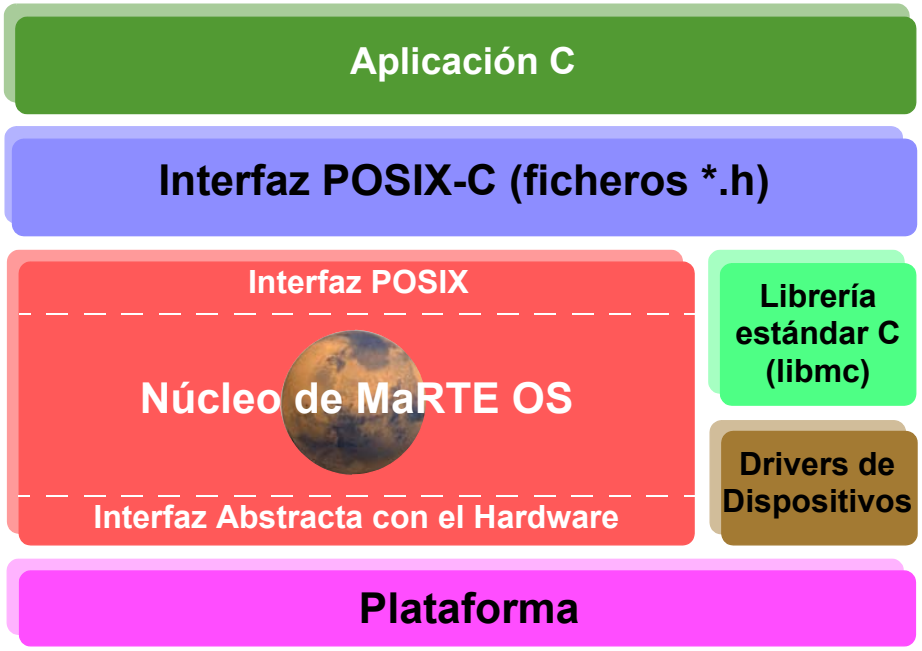


- Código libre (GPL). Descarga: *http://marte.unican.es*

Arquitectura



Aplicación Ada ejecutando sobre MaRTE OS



Aplicación C ejecutando sobre MaRTE OS

MaRTE OS como librería de POSIX-threads para Linux

Tres configuraciones diferentes (desde MaRTE OS V1.56):

- Arquitectura x86_PC
  - Las aplicaciones se cargan y ejecutan en un PC desnudo
- Arquitecturas Linux and Linux_lib
  - MaRTE OS se comporta como una librería de Pthreads
    - concurrencia a nivel de librería (sin usar LinuxThreads)
    - posible asignar prioridad a threads sin ser superusuario
  - Los programas MaRTE OS se ejecutan como cualquier otro proceso Linux de usuario

En DISCA (UPV) surgió la idea y la primera implementación de MaRTE OS como proceso Linux (superusuario) [1]

Interfaz Abstracta con el Hardware para Linux y Linux_lib

Señales Linux en el papel de las interrupciones hardware

- instalar manejador de señal (= inst. manejador de interrup.)
- modificar máscara señales (= habilitar o deshab. interrup.)

Temporizador Linux en lugar de temporizador hardware

- envía señal al proceso cuando expira

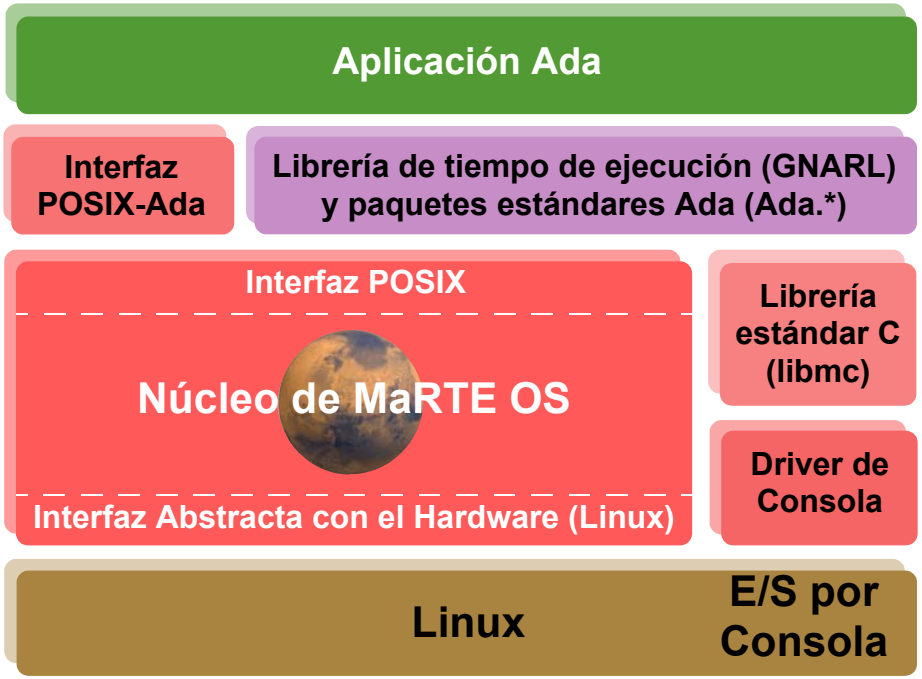
Reloj

- Time Stamp Counter (arquitectura Linux)
- Linux `gettimeofday()` (arquitectura Linux_lib)

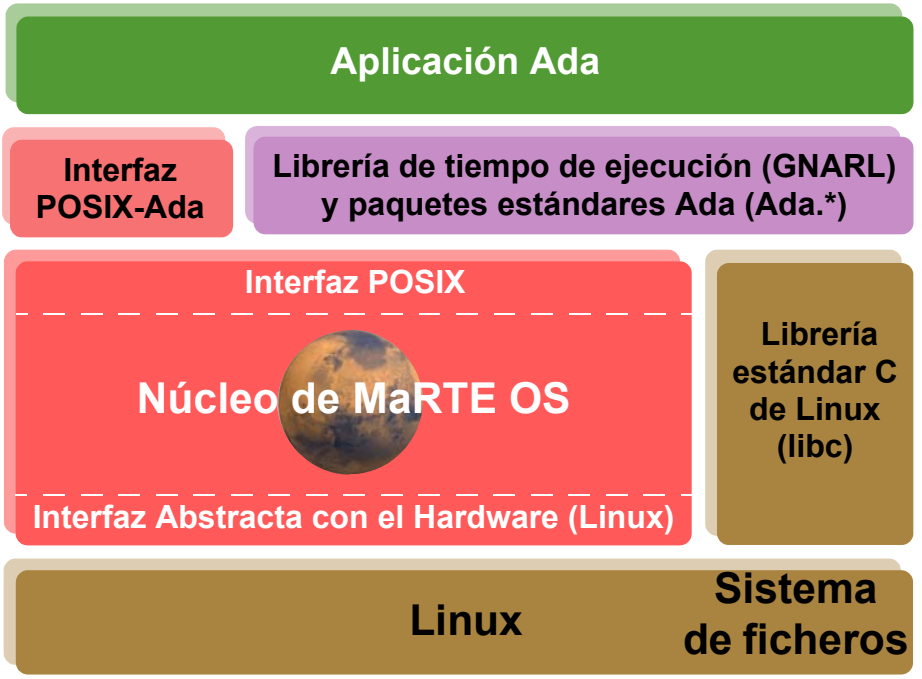
Rutina de cambio de contexto

- la misma que se usa en la arquitectura x86

Diferencias entre las Arquitecturas Linux y Linux_lib

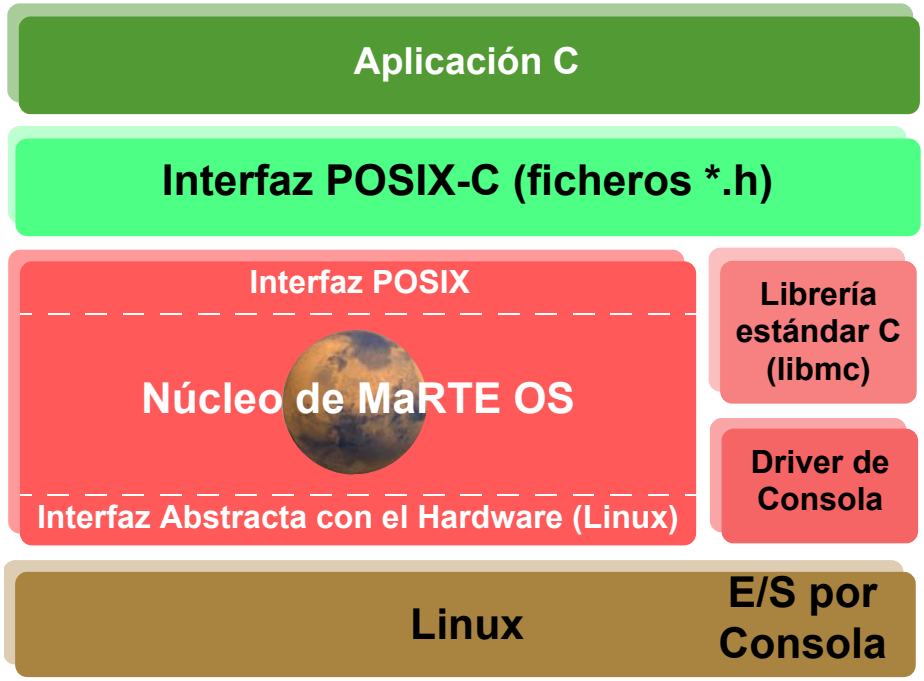


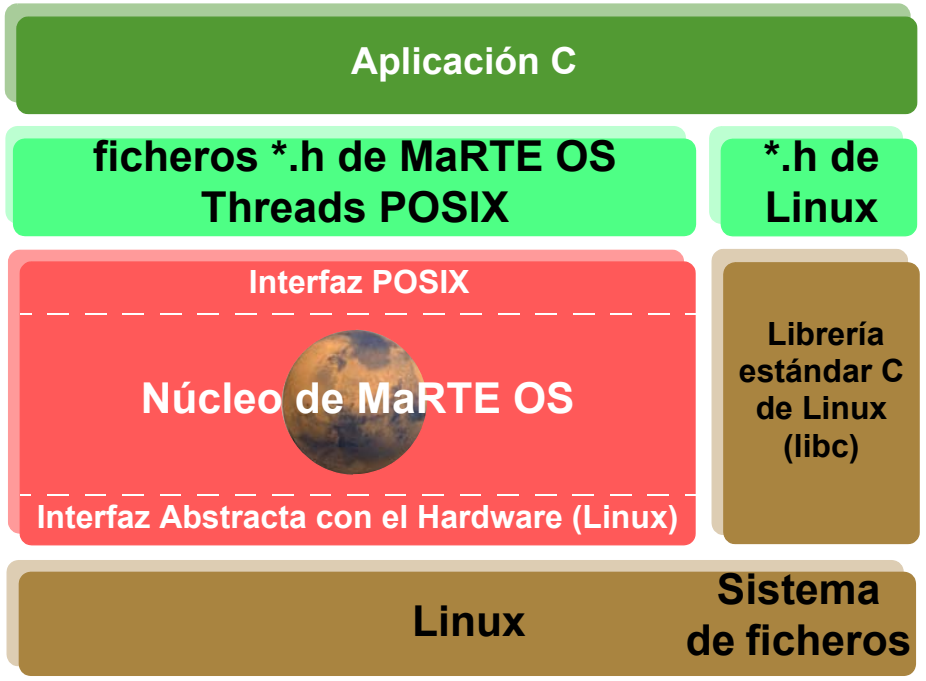
- Arquitectura Linux
  - Usa `libc` y "sistema de ficheros" de MaRTE OS
  - Manejador de consola usa `stdin` y `stdout` del proceso



- Arquitectura Linux_lib
  - Usa librerías Linux estándares
  - Es posible acceder al sistema de ficheros Linux

Diferencias entre las Arquitecturas Linux y Linux_lib (cont.)

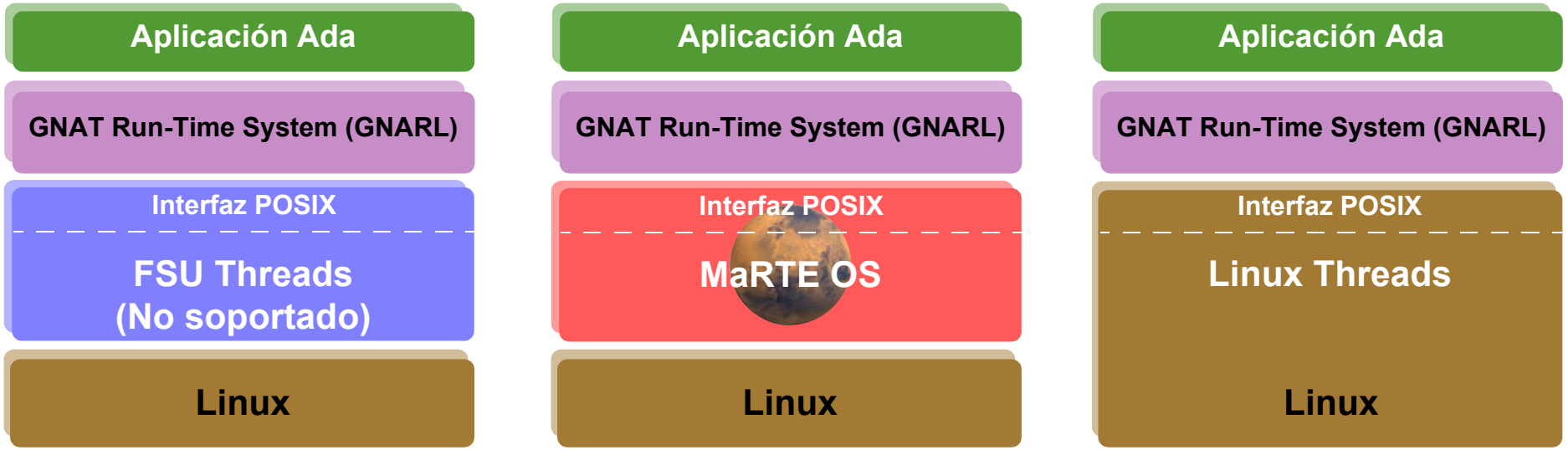




- Arquitectura Linux
  - Usa `libc` y "sistema de ficheros" de MaRTE OS
  - Manejador de consola usa `stdin` y `stdout` del proceso
- Arquitectura Linux_lib
  - Usa librerías Linux estándares
  - Es posible acceder al sistema de ficheros Linux

La arquitectura Linux_lib y el compilador GNAT

- GNAT puede usar diferentes sistemas de tiempo de ejecución
  - rts-fsu: basado en FSU Threads (*¡ya no soportado!*)
  - rts-native: basado en Linux threads
- AdaCore podría estar interesado en un rts-marte
  - substituto de rts-fsu para enseñanza (corto plazo)
  - como RTS para PC desnudo (medio-largo plazo)



Utilidad de las arquitecturas Linux y Linux_lib

Buena elección para cursos de programación en tiempo real:

- Posible usar prioridades y políticas de planificación de tiempo real (POSIX y Ada) en un sistema Linux convencional
  - no se requieren privilegios de superusuario
- Posible usar servicios avanzados en cursos de POSIX de TR
  - relojes y tempor. de tiempo de CPU, políticas RR y SS, ...

Test

- Método simple y rápido para realizar test funcional de aplicaciones sin necesidad de utilizar la plataforma final

NO se puede lograr comportamiento de tiempo real estricto:

- Aplicaciones MaRTE bajo el control del planificador Linux
- Afectadas por memoria virtual, actividades del núcleo, ...

Trabajo Futuro

- Convertir MaRTE Linux_lib en un nuevo sistema de tiempo de ejecución para GNAT
  - MaRTE OS será más fácil de instalar y usar
  - necesario para que AdaCore incluya MaRTE OS en la distribución estándar de GNAT
  - parece fácil: mover ficheros y crear librerías (casi hecho)

- Portar MaRTE OS a Windows y Solaris
  - GNAT también se distribuye para esos sistemas operativos
  - no sabemos lo difícil que puede resultar

Limitaciones conocidas (¿importantes para docencia?)

- Problemas al usar a la vez '`*.h`' de MaRTE y de Linux
  - redefinición de tipos (`pthread_t`, `pthread_attr_t`, `pthread_mutex_t`, etc.)
  - se puede resolver fácilmente en la mayoría de los casos
- Relojes y temporizadores de tiempo de ejecución
  - contabilizan el tiempo durante el que están ejecutando otros procesos del sistema
- No es posible utilizar de forma directa las señales Linux
  - `sigaction()`, `sigprocmask()`, ... actúan sobre las señales de MaRTE OS
- Operaciones de E/S bloqueantes
  - Cuando una tarea se bloquea, se bloquea todo el proceso

Java de Tiempo Real en MaRTE OS: RTSJ

- "Write Once, Run Anywhere" es una característica de Java muy interesante para sistemas empotrados
  - Plataformas muy diversas
  - Sistemas abiertos
- Java es "intencionadamente" inapropiado para Tiempo Real
  - Planificación, sincronización, interrupciones, gestión de memoria, E/S, no bien definidas para lograr portabilidad
- Real-Time Specification for Java (RTSJ) [2]
  - Define nuevas clases (p.e. `RealtimeThread`)
    - No implica cambios en el lenguaje
  - Sacrifica portabilidad por Tiempo Real:
    - "Write Once Carefully, Run Anywhere Conditionally"

Implementación de la RTSJ elegida: jRate

GCJ (Compilador de GNU para Java) [3]

- "ahead-of-time compiler". Puede compilar:
  - Código fuente Java a código máquina
  - Código fuente Java a "bytecode" (ficheros `*.class`)
  - "bytecode" a código máquina
- Las aplicaciones se enlazan con la librería de tiempo de ejecución (libgcj)

jRate (Java Real-Time Extension) [4]

- extensión del compilador GCJ y de su librería de tiempo de ejecución
- Añade mucha de la funcionalidad definida en la RTSJ
- Escrita en C++ y Java

Adaptación de jRate a MaRTE OS

jRate tiene un entorno de desarrollo complejo

- compilación completa del GCC (es delicado hacer cambios)

jRate se basa en LinuxThreads (interfaz POSIX)

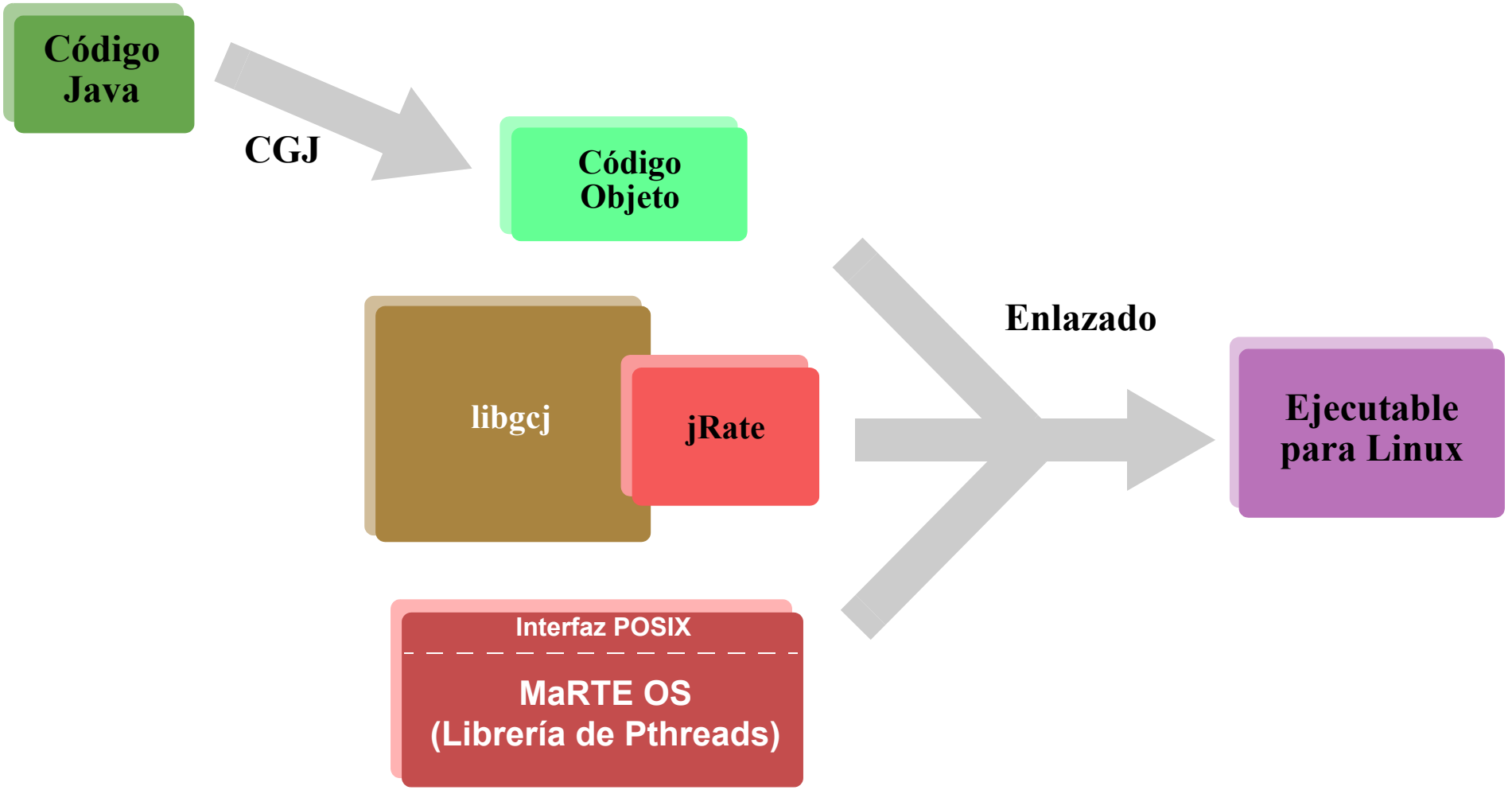
- adaptación: compilar con los ficheros '`*.h`' de MaRTE OS

Problemas encontrados

- Versiones de compiladores y utilidades
- Enlazado: librerías, versiones, orden de las librerías
- jRate no comprueba valor de retorno de las funciones POSIX
- Constructores de objetos estáticos de C++ se ejecutan antes de la inicialización de MaRTE OS

Realizada en el grupo RTS de York (Prof. Andy Wellings)

Generación de aplicaciones



Ampliación de la funcionalidad proporcionada por jRate

Cada objeto `RealtimeThread` puede tener definidos:

- Plazo (*jRate sólo considera plazos iguales a periodos*)
- Tiempo de ejecución de peor caso (*no utilizado en jRate*)
- Manejadores para ejecutar cuando se supere alguno de los parámetros anteriores

Los manejadores pueden a su vez tener asociados los mismos parámetros (incluyendo otros manejadores) (*no en jRate*)

Toda esta funcionalidad está implementada en la adaptación de jRate para MaRTE OS

- Temporizadores POSIX de tiempo de CPU para detectar sobrepasos del WCET (no disponibles en Linux)

Estado actual de jRate para MaRTE OS

Funcionalidad implementada

- Threads de tiempo real (`RealtimeThread`)
- Manejadores de eventos asíncronos (`AsyncEventHandler`)
- Temporizadores (`Timer`)

Algunas limitaciones

- Sólo threads periódicos (no esporádicos ni aperiódicos)
- No sincronización utilizando protocolos de tiempo real

Disponible en la web de MaRTE OS (sólo para Linux_lib):

- Precompilado: fichero “tar” con las librerías, clases Java y GCJ (muy fácil de instalar)
- Parche para jRate

Generalización del modelo de la RTSJ 1.0.1

Cuando `RealtimeThread` (periódica) pierde su plazo:

- permanece activa
- si hay un manejador asociado con la pérdida de plazo:
  - se activa el manejador
  - no será activada cuando llegue su próxima activación salvo que el manejador lo indique
- si no hay manejador: se le notifica la perdida del plazo cuando llame a `waitForNextPeriod`

Cuando `RealtimeThread` (periódica) sobrepasa su WCET:

- es suspendida y se activa manejador (en caso que le haya)
- el manejador puede aumentar su capacidad (es reactivada)
- la capacidad se recarga cuando llega su próxima activación

Generalización del modelo de la RTSJ 1.0.1 (cont.)

Comportamiento diferente para otros objetos planificables:

- `RealtimeThread` no periódicas y `AsyncEventHandler`

En una nueva revisión de la RTSJ se pretende plantear un modelo general para todos los objetos planificables [5]:

- interfaz `Schedulable`: nuevos métodos que permitan ignorar las próximas activaciones de un objeto planificable
- clase `RealtimeThread`: mecanismo de activación general
  - válido también para aperiódicas y esporádicas
  - `waitForNextRelease` frente a `waitForNextPeriod`
- clase `AsyncEventHandler`: mecanismo para gestionar sobrepasos de plazo cuando no hay manejador

La implementación del modelo general en jRate para MaRTE permitirá evaluar su validez y complejidad

Trabajo futuro

- Completar la implementación de la RTSJ
  - Sincronización utilizando protocolos de tiempo real
  - `ProcessingGroupParameters`: aislamiento temporal (C cada P) de uno (o varios) objetos planificables
  - Añadir otros planificadores (subclases de `Scheduler`)
- Ampliaciones a la RTSJ 1.0.1
  - Plazos mayores que los periodos
- Portado a MaRTE OS para máquina desnuda
  - Resolver posible problema con la carga dinámica de las clases

Referencias

[1] Miguel Masmano, Jorge Real, Ismael Ripoll, and Alfons Crespo. “Extending the Capabilities of Real-Time Applications by Combining MaRTE-OS and Linux”. 9th International Conference on Reliable Software Technologies Ada-Europe 2004. Palma de Mallorca, 14-18 Junio 2004.

[2] “The Real-Time Specification for Java”, versión 1.0. Addison Wesley, 2000. Disponible en http://www.rtj.orj

[3] The GNU Compiler for the Java Programming Language. http://gcc.gnu.org/java/

[4] jRate (Java Real-Time Extension). Angelo Corsaro. http://jrate.sourceforge.net/

[5] Andy Wellings, Greg Bollella, Peter Dibble, David Holmes. “Cost Enforcement and Deadline Monitoring in the Real-Time Specification for Java”. Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'04). May 12 - 14, 2004. Vienna, Austria