Miosix: una introduzione

Federico Terraneo

7 aprile 2011

Introduzione

- Cos'è un microcontrollore?
- Perchè un OS per micorocontrollori?
- Perchè non portare un OS desktop già esistente?
- Architettura di Miosix
- Librerie per Miosix
- Case study: scheduling control-based

Cos'è un microcontrollore?

Un microcontrollore è un circuito integrato che contiene una CPU, della RAM, una memoria nonvolatile (di solito FLASH) e delle interfacce per comunicare con il mondo esterno.
Tutti i microcontrollori hanno dei GPIO, piedini il cui livello logico è controllabile via software. E' possibile configurarli come uscite e impostare un livello logico, o come ingressi e leggere il livello logico.
La maggior parte dei microconttrollori hanno anche delle interfacce specializzate, come ADC (per leggere segnali analogici), SPI, I2C, UART, USB (per comunicare con dei protocolli standard)

Cos'è un microcontrollore?

Guardando al mondo dei microcontrollori, si nota che possono essere divisi all'incirca in due categorie:

- Microcontrollori a 8bit, con architetture propietarie del produttore e dimensioni della RAM da un centinaio di byte a qualche KB, e dimensione della FLASH da qualche KB a qualche decina di KB. Il clock è dell'ordine delle decine di MHz
- Microcontrollori a 32bit, con architetture comuni come ARM, e quantità più consistenti di RAM (da qualche KB a oltre 100KB), e di FLASH (da qualche decina di KB a oltre il MB). Il clock varia da qualche decina di MHz a oltre 100MHz

Perchè un OS per micorocontrollori?

Come abbiamo visto, i microcontrollori hanno poche risorse, perchè usare un OS invece di programmarli «bare metal»?

Risposta 1: I microcontrollori stanno cambiando. Negli anni '90 il PIC16F84 aveva 68 Byte di RAM e 1KByte di FLASH. Oggi con la diffusione dei microcontrollori ARM a 32bit le risorse a disposizione stanno aumentando, rendendo fattibile l'uso di un OS.

Risposta 2: Multithreading. Spesso un sistema embedded deve poter eseguire più task contemporaneamente, e le soluzioni classiche come polling e uso di interrupt non scalano per progetti complessi:

- Polling: si finisce per scrivere il codice come una serie di FSM, che rendono il codice poco leggibile.
- Interrupt: tutte le routine di interrupt vanno comunque scritte come FSM.

Perchè un OS per micorocontrollori?

Risposta 3: Astrazione dell'hardware. Su un PC il programma più semplice è un printf(Hello world);
Su un microcontrollore questo è un programma già molto complesso perchè bisogna decidere dove redirigere l'output di printf, ad esempio su una porta seriale o un display, scrivere il driver per quella periferica, e integrarlo con la libreria del C.
Se una printf è difficile, immaginatevi aprire un file con una fopen() o gestire un dispositivo USB...

Perchè non portare un OS desktop già esistente?

I sistemi operativi desktop non sono progettati per funzionare su microcontrollori. Sebbene ci sia uClinux che funziona anche senza una MMU, il limite è la quantità di memoria richiesta.
Per Linux serve minimo 1MB di RAM, mentre i microcontrollori hanno solo qualche decina di KB.

Non è solo una limitazione temporanea che sarà superata con il passare del tempo. Spesso nei sistemi embedded per ragioni di costo si cerca di usare il minimo di risorse necessarie. Per esempio, micorocontrollori con solo un centinaio di byte di RAM sono ancora molto comuni.

Architettura di Miosix

I microcontrollori, come abbiamo detto, hanno due memorie: una RAM e una FLASH. La FLASH è in genere più grande della RAM, questo perchè i microcontrollori sono pensati per *eseguire codice direttamente dalla FLASH*. Questo è in contrasto con quanto viene fatto da un OS desktop, che deve caricare il codice di una applicazione in RAM prima di eseguirlo.

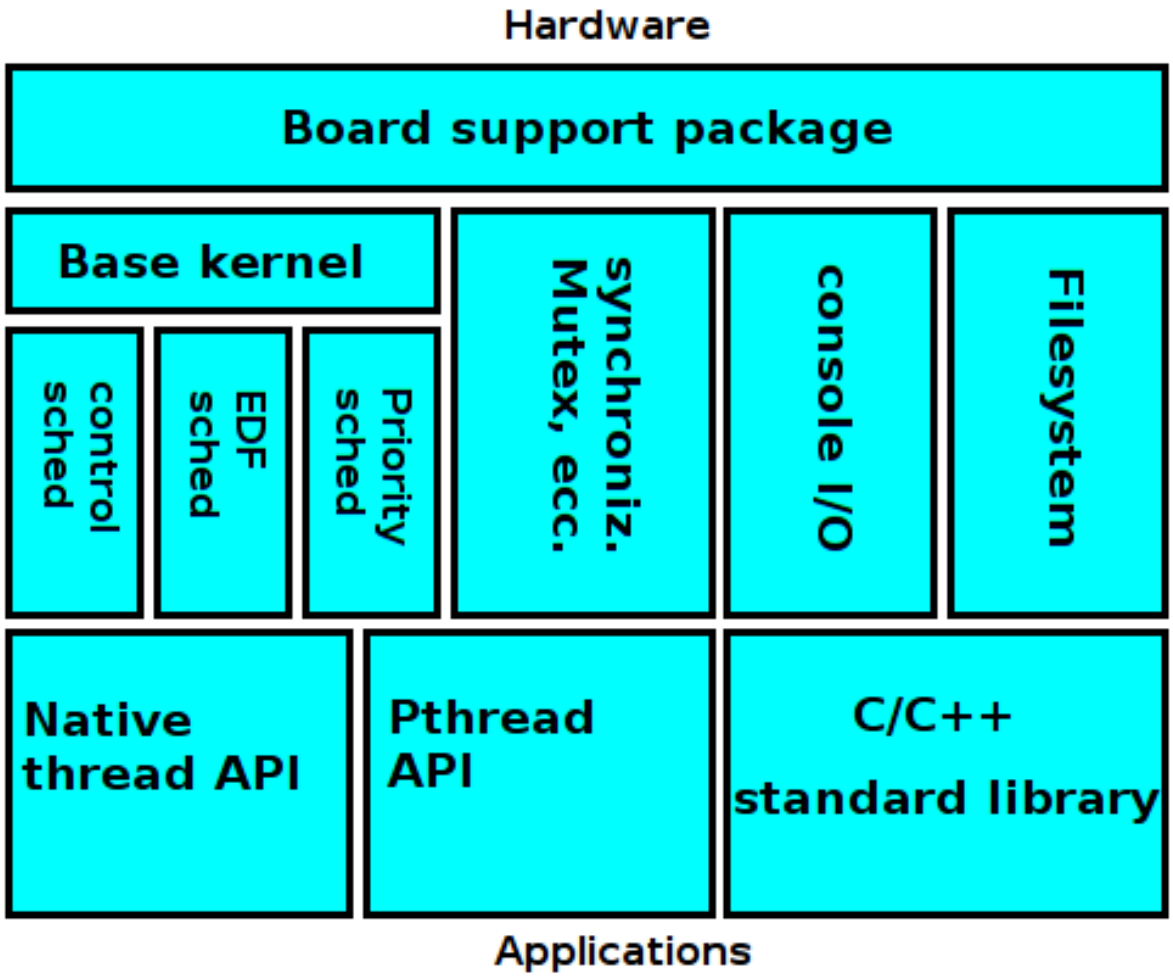
Miosix non ha un loader, il codice del kernel e delle applicazioni viene linkato staticamente in un binario che viene trasferito nella FLASH del microcontrollore, ed eseguito in FLASH.

Architettura di Miosix

- Vantaggio: si risparmia RAM, che viene usata solo per stack e heap, e non anche per il codice dei programmi. E' questo il «trucco» che consente a Miosix di stare nei requisiti di memoria dei microcontrollori.
- Svantaggio: non si può caricare codice dinamicamente.
- Vantaggio: il fatto che non si può caricare codice dinamicamente consente al linker di strippare codice che non viene usato. Si risparmia anche FLASH.

Miosix può essere considerato un kernel monoprocesso, multithread. Al termine del boot viene chiamato il main() che è l'entry point dell'unico processo nel sistema. E' però possibile creare thread.

Architettura di Miosix



More info here: http://www.webalice.it/fede.tft/miosix/index.html

Architettura di Miosix

Portabilità:
Il kernel è attualmente disponibile per due architetture di microcontrollori:

- STM32: Prodotti da ST e basati su un core ARM Cortex-M3
- LPC2000: Prodotti da NXP e basati su un core ARM7

Librerie per Miosix

Recentemente, Miosix si è esteso con dei «companion projects», librerie che forniscono una astrazione indipendente dalla piattaforma per gestire specifiche periferiche:

- *mxusb*: Una librera USB device integrata con Miosix. Già disponibile qui: http://gitorious.org/mxusb
- *mxgui*: Gestione di display LCD/OLED grafici. Work in progress.
- In futuro, forse, anche uno stack TCP/IP...