Lezione R13

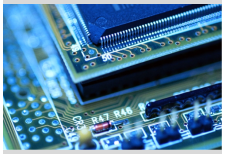
Sistemi operativi real-time – II

Sistemi embedded e real-time

15 gennaio 2013

Marco Cesati

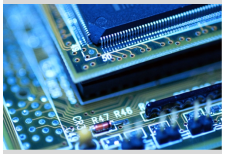
Dipartimento di Ingegneria Civile e Ingegneria Informatica
Università degli Studi di Roma Tor Vergata



Di cosa parliamo in questa lezione?

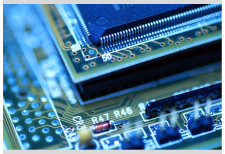
In questa lezione descriviamo brevemente alcuni dei più diffusi sistemi operativi real-time

1. Caratteristiche comuni degli RTOS
2. VxWorks
3. LynxOS
4. QNX Neutrino
5. eCos
6. Windows Embedded CE
7. Linux come RTOS



Caratteristiche comuni dei principali RTOS

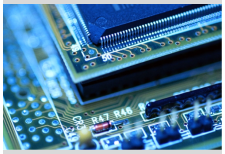
- Corrispondenza agli standard: generalmente le API sono proprietarie, ma gli RTOS offrono anche compatibilità (*compliancy*) o conformità (*conformancy*) allo standard Real-Time POSIX
- Modularità e Scalabilità: il kernel ha una dimensione (*footprint*) ridotta e le sue funzionalità sono configurabili
- Dimensione del codice: spesso basati su microkernel
- Velocità e Efficienza: basso overhead per cambi di contesto, latenza delle interruzioni e primitive di sincronizzazione
- Porzioni di codice non interrompibile: generalmente molto corte e di durata predicibile
- Gestione delle interruzioni "separata": interrupt handler corto e predicibile, ISR lunga e di durata variabile
- Gestione della memoria: possibilità di utilizzare memoria virtuale e protezione dello spazio di indirizzi kernel (disabilitabili); non esiste in genere la paginazione



La schedulazione nei principali RTOS

- Almeno 32 livelli di priorità differenti
- Possibilità di scelta fra FIFO e Round Robin per la gestione dei task nello stesso livello di priorità
- Possibilità di cambiare la priorità a run-time
- Generalmente non supportano in maniera nativa politiche di scheduling a priorità dinamica (es. EDF) o la possibilità di integrare server a conservazione di banda, ecc.
- Offrono meccanismi di controllo dell'inversione di priorità: tipicamente priority inheritance, alcuni RTOS anche priority ceiling
- Risoluzione nominale di timer e orologi di un nanosecondo, ma l'accuratezza non è mai sotto a 100 ns a causa della latenza di gestione delle interruzioni

Alcuni meccanismi, come quelli di controllo dell'inversione di priorità, possono essere abilitati o disabilitati in fase di progetto

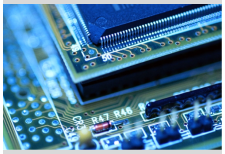


Quanti sono gli RTOS?

Esistono molti sistemi operativi RTOS, in particolare per il segmento embedded

Ad esempio:

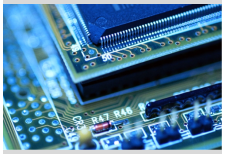
- VxWorks
- LynxOS
- QNX
- Windows CE
- Linux
- eCos
- PikeOS
- ThreadX
- RTEMS
- Operating System Embedded
- OS-9
- Nucleus OS



Quanti sono gli RTOS? (2)

Abassi, AMOS, AMX RTOS, ARTOS (Locamation), ARTOS (Robotu), Atomthreads, AVIX, BeRTOS, BRTOS, CapROS, ChibiOS/RT, ChorusOS, ChronOS, CMX RTOS, cocoOS, Concurrent CP/M, Concurrent DOS, Contiki, COS, CooCox CoOS, Deos, DioneOS, DNIX, GEC DOS, DrRtos, DSPnano RTOS, DSOS, eCos, eCosPro, embOS, Embox, ERIKA Enterprise, EROS, Femto OS, FlexOS, FreeRTOS, FunkOS, Fusion RTOS, Helium, HP-1000/RTE, Hybridthreads, IBM 4680 OS, IBM 4690 OS, INTEGRITY, IntervalZero RTX, ITRON, μITRON, ioRTOS, iRTOS, KolibriOS, LynxOS, MaRTE OS, MAX II,IV, MenuetOS, Micrium μC/OS-II, Micrium μC/OS-III, Milos, Microsoft Invisible Computing (MMLite), MP/M, MERT, Multiuser DOS, Nano-RK, Neutrino, Nokia OS, Nucleus OS, NuttX, On Time RTOS-32, OS20, OS21, OS4000, OPENRTOS, OSA, OSE, OS-9, OSEK, Phar Lap ETS, PaulOS, PICOS18, picoOS, Phoenix-RTOS, PikeOS, Portos,POK, PowerTV, Prex, Protothreads, pSOS, QNX, Q-Kernel, QP, RDOS, REAL/32, Real-time Linux (CONFIG_RT_PREEMPT), REX OS, RSX-11, RT-11, RTAI, RTEMS, rt-kernel, RTLinux, RT-Thread, RTXC Quadros, SafeRTOS, Salvo, SCIOPTA, scmRTOS, SDPOS, SHaRK, SimpleAVROS, SINTRAN III, Sirius RTOS, SMX RTOS, SOOS Project, Symbian OS, SYS/BIOS, Talon DSP RTOS, TargetOS, T-Kernel, THEOS, ThreadX, Trampoline Operating System (OSEK and AUTOSAR), TNKernel, Transaction Processing Facility, TRON project, TUD:OS, Unison RTOS, UNIX-RTR, μTasker, u-velOSity, VRTX, Windows CE, Xenomai, xPC Target, Y@SOS, MontaVista Linux, μnOS, uOS

Sistemi operativi real-time
Marco Cesati

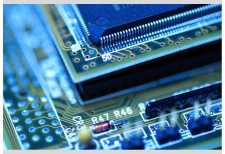


Schema della lezione
Caratteristiche comuni
VxWorks
LynxOS
QNX
eCos
Windows
Linux come RTOS

VxWorks

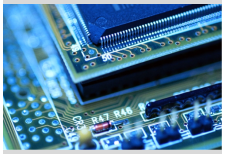
VxWorks è un RTOS commerciale prodotto dalla WindRiver (`www.windriver.com`)

- Installato su oltre 500 milioni di dispositivi
- Basato sul microkernel Wind
- È utilizzato in sistemi ad elevata affidabilità con certificazione DO-178B livello A, Arinc 653, IEC 61508
- Ha un ambiente di sviluppo basato su Eclipse (WorkBench)
- Supporta lo sviluppo cross-platform e semplifica la creazione delle toolchain di compilazione e collegamento
- Adatto sia per sistemi embedded che per sistemi con molte risorse
- Le API (interfaccie per le applicazioni) sono proprietarie
- Fornisce anche un'interfaccia di compatibilità per le API POSIX Real-Time



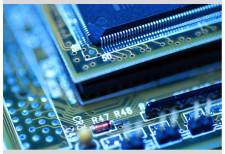
Gestione delle interruzioni in VxWorks

- Le Interrupt Service Routine (ISR) eseguono in un contesto differente rispetto a quello di tutti i task ed a priorità più elevata di quelle dei task
- È possibile aggiustare la priorità di interruzioni provenienti da particolari periferiche
- Ciascuna ISR può essere interrotta da una ISR di priorità superiore
- Le ISR condividono un unico stack (che deve essere abbastanza grande in rapporto al massimo livello di annidamento di interruzioni richiesto dal progetto)
- Le interruzioni possono essere disabilitate o abilitate mediante le API proprietarie `intLock()` ed `intUnLock()`



Schedulazione in VxWorks

- I task del sistema sono in generale interrompibili
- Lo scheduler è a priorità fissa a livello di task
- Task di pari priorità possono essere gestiti con politiche FIFO oppure Round Robin
- È possibile disabilitare l'interrompibilità di un task (API proprietarie `taskLock()` e `taskUnLock()`)
- Supporta il meccanismo di priority inheritance
- Supporta architetture SMP (multiprocessore simmetrico) ed architetture AMP (multiprocessore asimmetrico)
- Consente di specificare la CPU preferenziale di esecuzione di un task (`taskCpuAffinitySet()`)



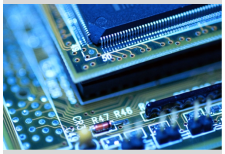
Gestione della memoria e comunicazione in VxWorks

Se presente, può utilizzare un dispositivo MMU (Memory Management Unit)

- per realizzare memoria virtuale
- per proteggere lo spazio di indirizzamento
- sono comunque meccanismi disabilitabili

Supporta i principali meccanismi di comunicazione fra task:

- Memoria condivisa
- Semafori e mutex
- Code di messaggi e pipe
- Segnali



LynxOS

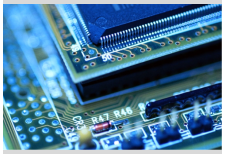
LynxOS è una linea di RTOS commerciali prodotti dalla LynuxWorks (`www.lynuxworks.com`)

Esistono diverse versioni, tra le quali:

- LynxOS RTOS: per sistemi embedded
- LynxOS-SE RTOS: per sistemi partizionati con applicazioni POSIX, Linux e conformi a Arinc 653
- LynxOS-178 RTOS: per sistemi *safety-critical*, certificabili DO-178B livello A e Arinc 653

Il cuore di tutti questi SO è LynxOS RTOS:

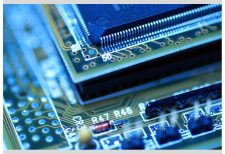
- Utilizzato in milioni di dispositivi
- Compatibile con l'ABI (Application Binary Interface) di Linux
- Utilizza un kernel monolitico
- L'ambiente di sviluppo è tipicamente "cross-platform" e basato su Eclipse



Gestione delle interruzioni in LynxOS

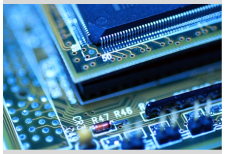
La gestione delle interruzioni è di tipo "separata":

- La prima parte della gestione è effettuata a priorità più elevata rispetto a quella dei task
- Nel caso la prima parte non completi la gestione dell'interruzione, viene impostata una interruzione (software) asincrona per interrompere il kernel
- Non appena il kernel è in esecuzione in modalità interrompibile, viene schedulato un thread per la gestione secondaria dell'interruzione
- Il thread esegue alla priorità del task che ha effettuato l'"apertura" del device che ha generato l'interruzione



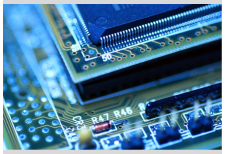
Lo scheduler di LynxOS

- I task di LynxOS sono in generale interrompibili
- Lo scheduler è a priorità fissa, con 256 livelli di priorità
- Task di uguale priorità sono gestiti come FIFO o Round Robin
- È disponibile anche uno scheduler “non-preemptive”, in cui i task non vengono interrotti da altri task
- Implementa il protocollo di priority inheritance sui semafori



Altri dettagli su LynxOS

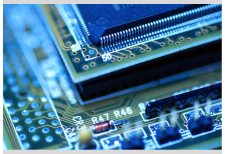
- Ha supporto completo (ma disabilitabile) per i meccanismi di memoria virtuale e protezione della memoria
- Ha supporto per il meccanismo di allocazione della memoria “su richiesta” (*demand paging*)
- Supporta architetture SMP
- È conforme allo standard POSIX, quindi possiede tutti i meccanismi di comunicazione e sincronizzazione fra task previsti dallo standard



QNX Neutrino

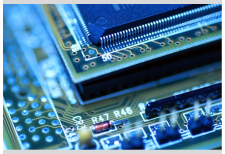
QNX Neutrino è un RTOS commerciale prodotto da QNX Software Systems (`www.qnx.com`)

- Basato su microkernel di dimensione estremamente ridotta
- Molto utilizzato soprattutto nel settore automotive
- Conforme allo standard POSIX
- È progettato attorno ad un meccanismo di scambio di messaggi e segnali, implementato nel microkernel
- Tutti i servizi non essenziali sono forniti da applicazioni (server) in spazio utente
- Gestione delle interruzioni di tipo “separata”
- Le interruzioni possono essere annidate
- Le interruzioni possono essere abilitate o disabilitate



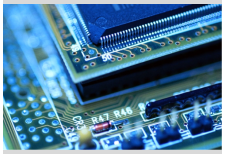
Lo scheduler di QNX Neutrino

- I task sono interrompibili
- Lo scheduler principale è a priorità fissa
- Task di pari priorità sono gestibili tramite FIFO o Round Robin
- Utilizza 256 livelli di priorità
- Implementa il protocollo di priority inheritance sui mutex POSIX Thread
- È supportato anche uno *sporadic scheduler* che consente di implementare server sporadici per la gestione di task aperiodici
- Il server sporadico è implementato per mezzo di un meccanismo che fa variare la priorità del task tra un valore alto (budget non esaurito) ed un valore basso (budget esaurito)



Architettura di QNX Neutrino

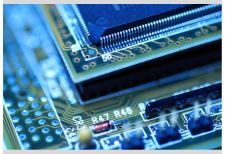
- Possiede tutti i meccanismi di comunicazione e sincronizzazione previsti dallo standard POSIX
- Lo scambio di messaggi avviene internamente in modalità sincrona (si evitano le code di messaggi)
- L'architettura basata su microkernel consente di fornire una protezione completa della memoria, sia agli applicativi "user space" sia ai driver ed alle estensioni modulari del kernel
- Supporta architetture SMP e AMP
- È possibile vincolare l'esecuzione di specifici thread su particolari processori (nella terminologia di QNX, BMP o Bound MultiProcessing)



eCos

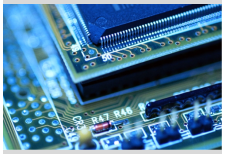
eCos (`ecos.sourceware.org`) è un RTOS *open-source*

- Progettato per sistemi embedded
- È distribuito con una licenza GPL compatibile
- È fortemente modulare ed altamente configurabile
- Offre sia API proprietarie che funzioni compatibili con POSIX e μITRON
- Il kernel è solo uno dei package del sistema, e non è indispensabile per lo sviluppo di applicazioni in eCos
- Il kernel offre il supporto alla gestione delle interruzioni, alla schedulazione ed alla sincronizzazione fra thread
- È scritto in C++ ed utilizza il compilatore GNU gcc
- I dettagli legati all'architettura sono incapsulati in un programma di interfaccia (HAL – Hardware Abstraction Layer) che supporta numerose architetture ed è scritto in C ed assembler



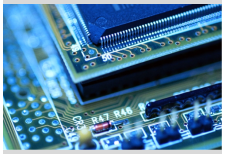
Gestione delle interruzioni in eCos

- La gestione delle interruzioni è implementata mediante l'interazione fra kernel e HAL
- Le API utilizzate dai driver delle periferiche per la gestione delle interruzioni utilizzano (in maniera trasparente per lo sviluppatore) l'interfaccia del kernel o quella di HAL
- La gestione è "separata"
- La prima fase è realizzata da HAL e dal kernel (priorità elevata, non interrompibile, non in contesto thread)
- La gestione secondaria è interrompibile ed è realizzata in thread a priorità più elevata dei normali thread



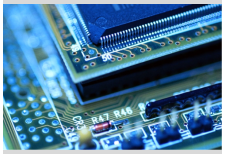
Lo scheduler di eCos

- I task sono interrompibili
- Lo scheduler è a priorità fissa
- Nella variante *bitmap* lo scheduler ammette al massimo 32 thread, quindi esistono solo 32 livelli di priorità!
- Non vi sono limiti sul numero di thread nella variante *multi-level queue* (MLQ) dello scheduler, ed il numero di livelli di priorità è configurabile
- Task di pari livello sono gestibili con Round Robin
- I task possono disabilitare l'interrompibilità
- Nella variante MLQ supporta architetture SMP
- Non è possibile legare un task ad un processore



Altri dettagli su eCos

- La configurazione è uno dei punti di forza: consente la selezione dei soli programmi necessari (footprint estremamente ridotto)
- Il meccanismo di configurazione è realizzato mediante compilazione condizionale e direttive al preprocessore
- I meccanismi di gestione della memoria sono delegati allo strato HAL
- Se è presente un dispositivo MMU, supporta memoria virtuale e protezione dello spazio di indirizzi
- I meccanismi di sincronizzazione e comunicazione sono compatibili con POSIX ed implementati nel kernel



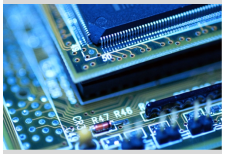
Windows Embedded

Microsoft offre una serie di RTOS embedded con footprint ridotto, ad esempio:

- Windows Embedded CE 6.0 e Windows Embedded Compact 7: le versioni più recenti e con maggiori funzionalità
- Windows Mobile: orientata alle applicazioni su telefoni cellulari, offre un sottoinsieme delle caratteristiche dell'Embedded CE 5.0
- Windows Phone 7: evoluzione di Windows Mobile, commercializzato nel 2011

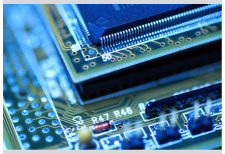
Riguardo a Windows Embedded CE:

- **Non** è una versione "leggera" di Windows!
- Ha un kernel monolitico
- Non è compatibile con lo standard POSIX
- Il codice del kernel è distribuito con licenza "shared-source" (simile alla licenza BSD)
- Supporta solo architetture monoprocessore



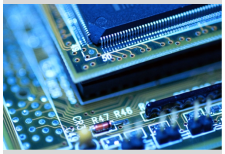
Gestione delle interruzioni su Windows Embedded

- La gestione delle interruzioni è “separata”
- La gestione immediata è effettuata dal kernel in contesto non interrompibile
- La gestione secondaria è delegata ad un *Interrupt Service Thread* (IST) opportuno
- Poiché l’IST non è rientrante, durante la sua esecuzione tutte le interruzioni sono abilitate tranne quella gestita dall’IST stesso
- Supporta l’abilitazione e disabilitazione delle interruzioni
- Le interruzioni possono essere annidate



Altri dettagli su Windows Embedded

- I task sono interrompibili
- Lo scheduler è a priorità fissa con 256 livelli di priorità
- Task di pari priorità sono gestibili con FIFO o Round Robin
- La priorità di un task è prefissata nel momento della creazione, e deve essere modificata successivamente
- Supporta il protocollo priority inheritance
- Sono supportati i principali meccanismi di comunicazione e sincronizzazione fra task: semafori e mutex, code di messaggi, eventi
- È supportato il meccanismo di memoria virtuale (non disabilitabile)
- È opzionale il meccanismo di paginazione su richiesta (demand paging)
- Sono disponibili un gran numero di driver già sviluppati



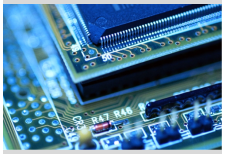
Linux come RTOS

È possibile considerare Linux come un sistema operativo real-time? ***Dipende. . .***

Linux **non** è un sistema **hard** real-time: è progettato per massimizzare il throughput delle applicazioni e minimizzare i tempi medi di risposta

Comunque Linux ha numerose caratteristiche che lo rendono conveniente per un utilizzo in ambienti real-time:

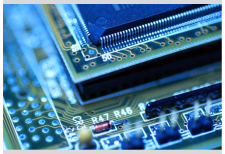
- Buona gestione di tempo e timer (*high resolution timer*)
- Gestione “separata” delle interruzioni
- Schedulazione e gestione delle priorità dei task
- Supporto alle politiche di accesso alle risorse condivise
- Interrompibilità dei task anche in kernel mode



Linux come RTOS (2)

- Possibilità di creare facilmente task in kernel mode, con cambi di contesto rapidi e gestione della memoria con un unico spazio di indirizzamento
- Supporto alla memoria virtuale e protezione della memoria per i task user mode (cambio di contesto lento)
- Possibilità di configurazione a grana estremamente fine
- Kernel monolitico, ma codice completamente disponibile, modificabile e modulare per ottenere occupazione della memoria (footprint) estremamente ridotta
- È il sistema operativo con il maggior numero di architetture supportate
- È il sistema operativo con il maggior numero di driver di periferiche disponibili
- Comunità di sviluppo, supporto e testing molto attive

Sistemi operativi real-time
Marco Cesati



Schema della lezione
Caratteristiche comuni
VxWorks
LynxOS
QNX
eCos
Windows
Linux come RTOS

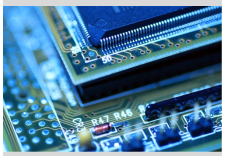
Limiti di Linux come hard RTOS

Il limite principale di Linux per il suo uso in ambito hard real-time è nella predicibilità temporale di kernel e applicazioni:

- Il kernel non è completamente interrompibile
- Le interruzioni sono disabilitate nelle sezioni critiche della gestione secondaria (ISR) di numerose interruzioni
- Le ISR possono avere una durata non predicibile
- La gestione delle interruzioni non fa uso di meccanismi di priorità
- La latenza di schedulazione dei task è elevata, soprattutto a causa del costo dei cambi di contesto

Si può rimediare a tutti questi limiti?

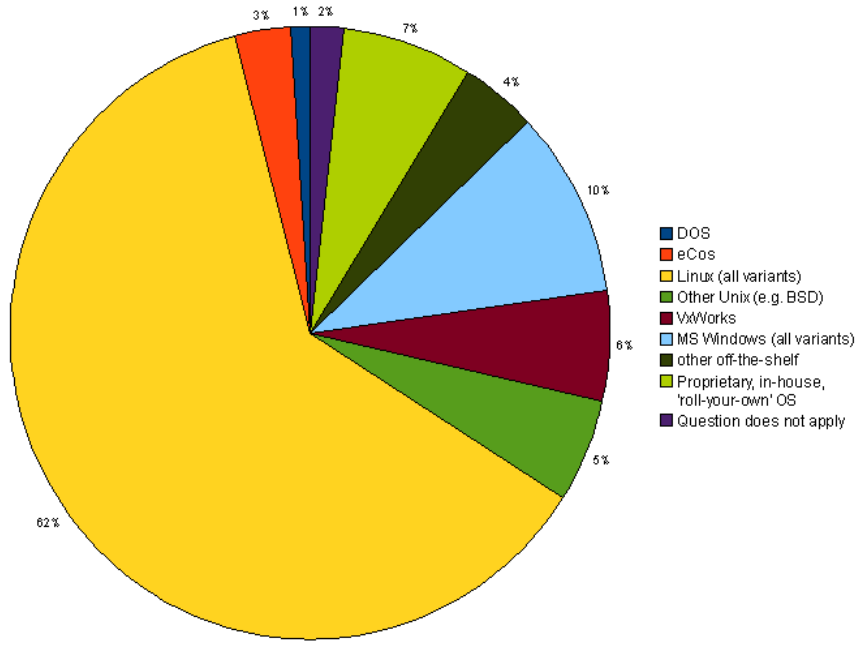
Non è semplice, ma effettivamente è possibile modificare il kernel Linux in modo da renderlo sostanzialmente predicibile



Linux come embedded RTOS

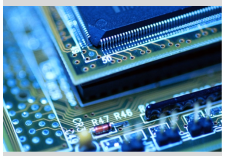
Il kernel Linux è monolitico, ha senso considerarlo per i sistemi embedded?

Sì, dato che è possibile configurare il kernel in modo da ridurre drasticamente la sua dimensione



Fonte: linuxfordevices.com

Sistemi operativi real-time
Marco Cesati



Schema della lezione
Caratteristiche comuni
VxWorks
LynxOS
QNX
eCos
Windows
Linux come RTOS