Edge of Real-Time – considerazioni sulla programmazione real-time – pt.1

William Cage: *Lei è americano?*
Sergente Farrell: *No, sono del Kentucky.*

Il bel **Edge of Tomorrow** è un film sul tempo, dove passato, presente e futuro si accavallano ed entrano in un loop infinito (*ma quasi senza paradossi, notevole…*), e mi offre un bello spunto per parlare della programmazione **real-time**, un tipo di programmazione dove il tempo gioca un ruolo fondamentale (*N.B.: si parla di tempo di esecuzione, non del tempo necessario allo sviluppo di una applicazione: quello è la croce di molti programmatori, ma è tutta un'altra storia… ah ah ah*).



…grazie al real-time riesco a fare anche questo…

Veniamo al dunque: cosa è una applicazione **real-time**? Prima di

dirlo farò una piccola premessa: in questa parte sto per ripetere concetti stra-conosciuti a chi mastica già l'argomento, ripetendo spiegazioni che usano quasi le stesse frasi (al limite del plagio) e che potete trovare in mille altre pagine sull'argomento: il fatto è che non posso parlare di **real-time** senza introdurlo, ed è impossibile farlo senza che sembri una copia di una descrizione già vista (giuro: ho trovato articoli sul real-time che sembrano fatti col copia-e-incolla, ma è inevitabile: sono quatto argomenti ben precisi e sintetici, ed è difficile descriverli in maniera molto originale). *Indi per cui: chi vuole può saltare tutta l'introduzione, anzi può andare direttamente al "Ciao, e al prossimo post!". Saluti e baci.*

E torniamo al dunque:

"Una applicazione real-time è una applicazione che esegue le sue attività con dei tempi garantiti"

Questa frase è del sottoscritto, l'ho sfornata proprio ora, ma magari l'ho copiata: il subconscio gioca brutti scherzi e ho letto molta (forse troppa) roba su questo argomento. La frase qua sopra nasconde già una insidia: *"ma allora un sistema veloce è un sistema real-time!"*. La risposta è **NO**: una applicazione veloce, ben scritta con un linguaggio adatto (*in C, no? Se no che ci stiamo a fare qui?*) e che gira su un computer molto veloce dotato di un buon sistema operativo (*Linux, ovviamente…*) potrebbe dare l'impressione di essere **real-time** con la sua grande velocità di risposta, ma in realtà non garantisce **SEMPRE** la velocità che ti aspetti, quindi non è veramente **real-time**. La parte importante della frase qui sopra è **"tempi garantiti"**, magari lunghi, ma garantiti.

Il caso che ho appena descritto come sbagliato offre lo spunto per il passo successivo: una applicazione **real-time** non è *solo* una applicazione, ma è un vero e proprio sistema composto da Sistema Operativo + Software + Hardware:

1. **Sistema Operativo**: deve essere di tipo real-time (RTOS

per gli amici).
2. **Software**: deve essere scritto rispettando lo "*stile real-time*" previsto dal RTOS prescelto, usando un linguaggio adatto (tipicamente il C, ma se ne usano anche altri).
3. **Hardware**: deve essere adatto alle esigenze del RTOS prescelto.

Devo evidenziare una caratteristica della tabellina qua sopra: tutti i punti elencati sono condizioni necessarie ma non sufficienti, nel senso che o sono rispettati tutti e tre o non potremo ottenere prestazioni **real-time**.

(*…ho saltato un caso particolare che non è argomento di questa trattazione: è possibile realizzare sistemi con prestazioni real-time anche programmando Hardware senza sistema operativo (programmazione "bare-metal"), in C o addirittura in Assembler, ma anche questa è un altra storia…*)

Ci manca solo un ultimo punto da descrivere: esiste un solo tipo di **real-time**? No, ce ne sono due, **Hard** e **Soft** (più un terzo, il **Firm**, che praticamente corrisponde col **Soft**, quindi lo saltiamo):

- **Hard real-time**: è, in realtà, l'unico che può fregiarsi del titolo **real-time**: un sistema di questo tipo garantisce i tempi previsti **SEMPRE**. In gergo del settore: *"non tollera il fallimento di nessuna deadline"* dove per *deadline* si intende il limite temporale massimo oltre il quale una attività **DEVE** essere completata, pena il degrado irrimediabile del sistema controllato (*provate a pensare a una centrale nucleare controllata da un sistema non real-time… bum!*).
- **Soft real-time**: si comporta, **NORMALMENTE**, come un vero sistema **real-time**, ma tollera il fallimento di **QUALCHE** *deadline* sporadica, con un degrado di funzionamento statisticamente accettabile.

E adesso non ci sta male una breve panoramica (non esaustiva e, perlopiù, soggettiva: metterò solo quelli che mi piacciono) dei sistemi operativi, di tipo RTOS, disponibili e raccomandabili per realizzare sistemi **Hard real-time**:

- **QNX** – Proprietario, POSIX-compliant. È un vero e proprio UNIX real-time, basato su un microkernel real-time.
- **LynxOS** – Proprietario, POSIX-compliant. Anche questo è un vero e proprio UNIX real-time, basato su un kernel monolitico real-time.
- **VxWorks** – Proprietario, POSIX-compliant. Molto completo, basato su un kernel monolitico real-time.
- **RT-Linux** – Open source, POSIX-compliant. È, in parole povere, un microkernel real-time su cui gira un Linux standard come processo a bassa priorità.
- **RTAI-Linux** – Open source, POSIX-compliant. È, come RT-Linux, un microkernel real-time su cui gira un Linux standard come processo a bassa priorità.
- **NuttX** – Open source, POSIX-compliant. Molto compatto e, quindi, adatto anche a sistemi *embedded* semplici. Basato su un microkernel real-time.

E, fuori dalla lista (perché non è un sistema operativo completo), aggiungerei anche il buon **FreeRTOS**, un RTOS compattissimo e leggerissimo (è quasi solo uno *scheduler*) studiato ad-hoc per sistemi *embedded* semplici che non necessitano di tutte le funzionalità avanzate fornite dai sistemi POSIX. È una ottima alternativa (ma non è l'unica) alla programmazione embedded *"bare-metal"*, quella senza sistema operativo.

Una parentesi a parte la merita il Linux standard, che, dal kernel 2.6 in avanti, ha delle **estensioni real-time** che si possono attivare ed usare congiuntamente a una *scheduling-policy* dello *scheduler* di tipo SCHED_FIFO o SCHED_RR (al posto di quella di *default*, che è SCHED_OTHER, come già citato **nel mio articolo sulla sched_yield(2)**). In questa maniera Linux standard può eseguire egregiamente attività di tipo **Soft real-**

time. Meglio che niente, no?

E un’altra parentesi bisogna spenderla sulle prestazioni assolute di questi RTOS: anche se nella introduzione dell’articolo ho ricalcato sul fatto che il **real-time** è *“rispetto dei tempi”* e non necessariamente velocità, tutti gli RTOS descritti sopra sono anche velocissimi, perché, essendo progettati per adempiere ai compiti più svariati, devono per forza tentare di fornire *“tempi garantiti”* anche quando si ha bisogno di risposte velocissime (avionica, centrali nucleari, ecc.). Quindi, ad esempio, **QNX** e compagni hanno tempi di latenza per le interruzioni e tempi di *context-switch* dell’ordine dei microsecondi (!). Invece Linux standard in modo **soft real-time** può fornire prestazioni dell’ordine dei millisecondi (*e anche in questo caso: meglio che niente, no?*).

Per oggi può bastare, abbiamo analizzato il punto 1 della prima tabella (*lo ammetto, ho descritto cose abbastanza semplici da reperire qui e là, ma almeno qua le trovate tutte insieme…*). Nella seconda parte (*che non necessariamente sarà il prossimo articolo…*) parleremo del punto 2 della tabella, **il Software in stile real-time**, che è un argomento un pelino più complicato. *E, ancora una volta, non trattenete il respiro nell’attesa!*

Ciao, e al prossimo post!

https://italiancoders.it/edge-of-real-time-considerazioni-sulla-programmazione-real-time-pt-1/