Presentazione

Seminari ATMEL/EBV/Acmesystem

NerInformatica

Relatore : Luciano Neri

Libero professionista
Ordine degli Ingegneri di Vicenza

Profilo personale e professionale

Luciano Neri, Bolognese, classe 1965

- Inizio dell'attività lavorativa negli anni '80
- Istituto Tecnico Industriale Statale di Bologna, diploma di perito elettronico
- Università di Padova, Ingegneria Informatica e Automatica
- Sposato, tre figli … maschi
- Metà anni '80 : progettazione schede elettroniche senza uC e programmazione software PC
- Inizio anni '90 : progettazione hardware e software per PC e schede con uC 8bit
- Metà anni '90 : primi software con Linux
- …
- 2005 : inizio della libera professione

http://www.nerinformatica.it/competenze - Profilo Linkedin: http://it.linkedin.com/in/lucianoneri

Cosa fà il nostro studio ??

… è uno studio di Ingegneria che opera nel mondo del software per l'industria e l’automazione, con un’offerta completa di servizi di progettazione, consulenza e formazione personalizzata,

… per ogni progetto seleziona, nel proprio ecosistema di aziende e professionisti, gli attori che coprono con il massimo della competenza ed esperienza, ogni necessità: gestionale, di progetto, di supporto alla produzione, di collaudo e verifica, di industrializzazione, di produzione,

… riduce sensibilmente il tempo di sviluppo del progetto, intervenendo per eliminare o ridurre in anticipo eventuali problemi,

… forma i tecnici per l'utilizzo delle tecnologie software e ICT nel campo embeded.

I nostri servizi

Formazione personalizzata

Consulenza

Studio di fattibilità

Progettazione e sviluppo SW

Prog.Elettronica

Sbroglio PCB

Industrializzazione

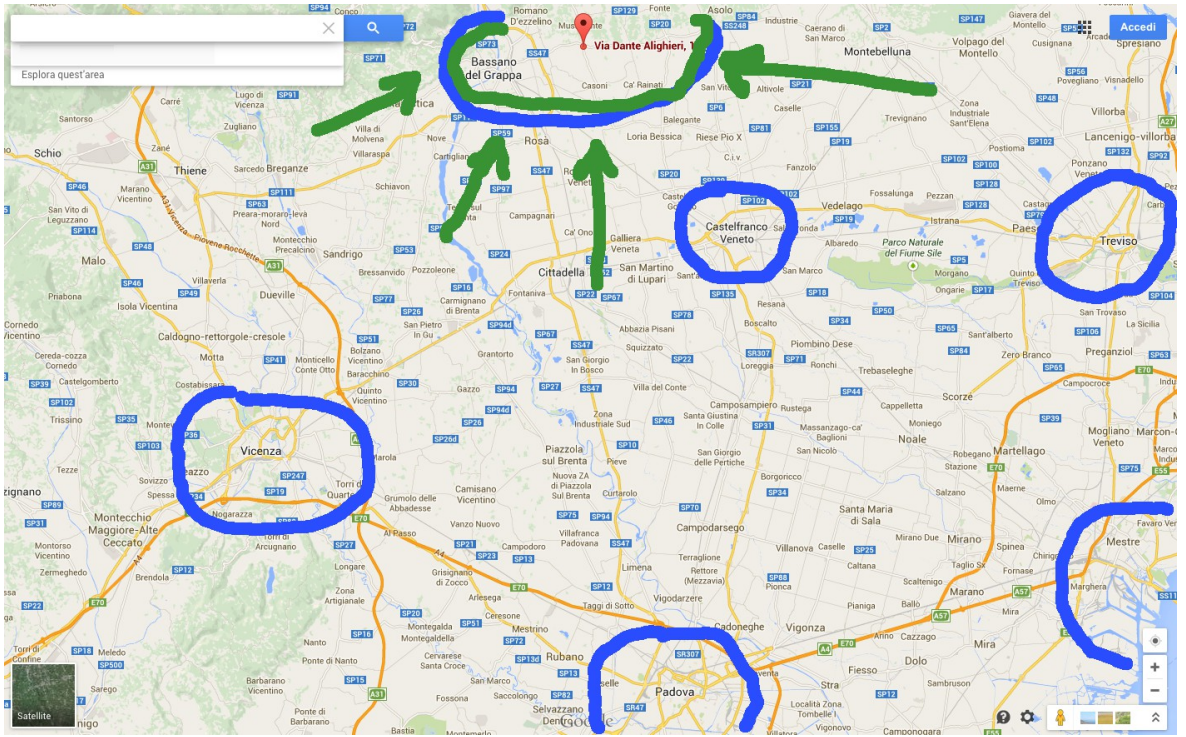
Test e qualificazione

Procedure di collaudo

Produzione pre-serie

Supporto tecnico

Dove siamo



Piazza della Vittoria, 6

36065 – Mussolente (VI)

tel: 0444 1835569

email : info@nerinformatica.it

5Km da Bassano del Grappa

45Km da Vicenza / Treviso

55Km da Padova

… e ora iniziamo a parlare di Linux

La programmazione in Linux

Seminari ATMEL/EBV/Acmesystems

NerInformatica

Relatore : Luciano Neri

Libero professionista
Ordine degli Ingegneri di Vicenza

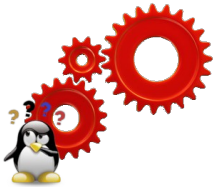
Agenda (45')

- 1. Programmazione da C embedded con Linux
  - Da ANSI C a POSIX C
  - Lo stile di codifica
  - Differenza di stili di programmazione con Linux
- 2. I componenti del kernel
  - Kernel space vs User space
  - La gestione della memoria virtuale
  - Il caricamento dei processi in memoria
  - La gestione degli ISR nel kernel
  - I device driver
- 3. Il debug del kernel e degli applicativi
- 4. Tools per il controllo formale dei sorgenti
- 5. L'uso degli script

Programmare su Linux embedded

- Non è „programmare un applicativo per Linux Desktop“
- Neanche „programmare un firmware per uC“
- Include molti moduli software del sistema, diversi tra loro
- Ogni modulo software è scritto con linguaggi e framework differenti
- Ogni modulo software richiede delle specifiche competenze
- Le funzionalità e la stabilità del prodotto non dipendono solo dall'applicativo;

L'esecuzione del codice



- In un sistema a uC
  - il codice viene eseguito direttamente in FLASH (NOR)
  - Solo i dati si trovano in RAM
  - Lo spazio di indirizzamento è diretto (solo indirizzi fisici)
- In un sistema Linux a uC
  - il codice si trova in FLASH e viene copiato in RAM per essere eseguito
  - Dati e codice si trovano in RAM e vengono protetti dalla MMU
  - Lo spazio di indirizzamento è virtuale, quindi mappato dalla MMU; gli indirizzi fisici di solito non si utilizzano

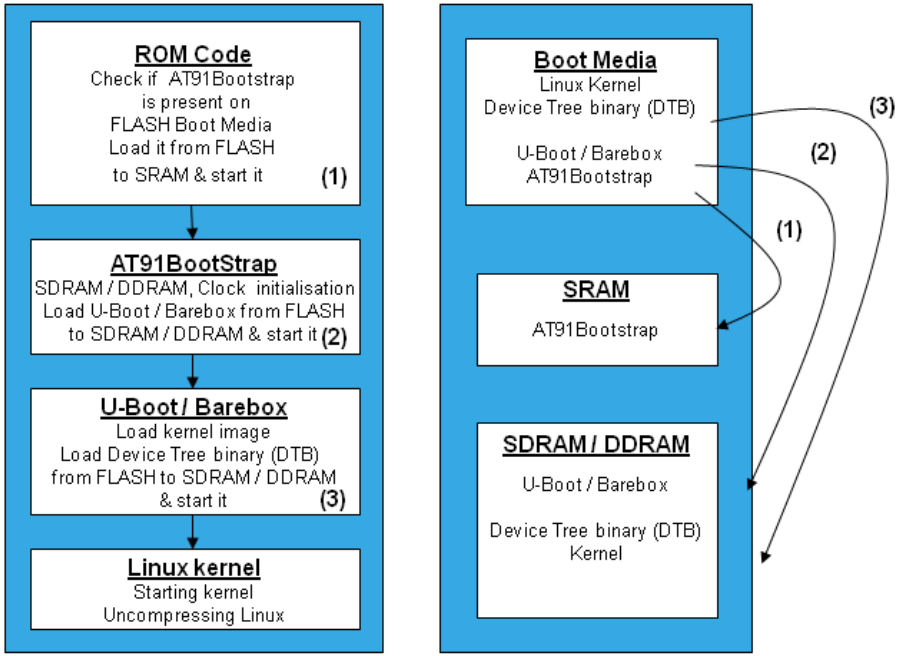
Come parte un sistema Linux ?

The boot sequence of linux4SAM is done in several steps :

1. Boot Program - ROM Code
Checks if a valid application is present in FLASH and if it is the case downloads it into internal SRAM.

2. AT91Bootstrap
In charge of hardware configuration, downloads U-Boot / Barebox binary from FLASH to SDRAM / DDRAM, starts the bootloader

3. U-Boot or Barebox
The bootloader, in charge of downloading kernel binaries from FLASH, network, SD card, etc. It then loads the Device Tree Binary and starts the Linux kernel.



http://www.at91.com/linux4sam/bin/view/Linux4SAM/GettingStarted

I moduli di un sistema Linux

Bootloader	Kernel e Device driver	Applicativi e Librererie
• Primo codice esterno eseguito dal uC • Init UART/RAM	• Astrazione completa dell'hardware verso gli applicativi	• Implementano le funzionalità del sistema embedded

Il C embedded con Linux

ANSI C e POSIX C

- Differenti API e strutture di controllo a seconda del modulo sw su cui si lavora

Lo stile di codifica

- Dipende fortemente dal software utilizzato
- Non viene forzato nei progetti open source
- es.www.kernel.org/doc/Documentation/CodingStyle

Differenti stili di programmazione

- Dipende dallo sviluppatore
- Scarso utilizzo di design pattern
- Scarsa documentazione sulle SM

Bootloader : ANSI C = firmware

- Programmazione in C/ASM
- Macchine a stati legate a ISR e timer HW
- Controllo diretto dell'HW
- API e gestione delle personalizzazioni strettamente legato allo specifico bootloader ed alle funzionalità

Il Kernel: Un mondo a parte … :-)

- Scritto in ANSI C + ASM
- Per la modifica o aggiunta di driver è necessario integrarsi con il kernel utilizzando propriamente tutte le API specifiche
- Gli Interrupt sono disponibili anche come Threaded Interrupt
- Si possono utilizzare i tasklet (simili ai thread) per la gestione di macchine a stati asincrone
- Lo spazio di indirizzamento del kernel è condiviso tra tutti i moduli e device driver e lo stack è di dimensione fissa
- L'area heap del kernel è di dimensione fissa e bloccata

Applicativo: Da ANSI C a POSIX C

- Lo spazio utente è compatibile POSIX, quindi tutti gli applicativi devono conformarsi a questi standard per poter essere eseguiti su un sistema Linux, questo sia dal punto di vista delle API libc che per l'accesso alle periferiche e alle risorse del sistema
- In generale si programma esattamente come su un sistema Linux desktop, tenendo conto delle limitazioni del sistema embedded (es. no swap) e delle differenti tipologie di allocazione della memoria

Lo stile di codifica

ESEMPIO APPLICAZIONE LINUX :

```
char *portname = "/dev/ttyUSB1"
 ...
int fd = open (portname, O_RDWR | O_NOCTTY | O_SYNC);
if (fd < 0)
{
        error_message ("error %d opening %s: %s", errno, portname, strerror (errno));
        return;
}
set_interface_attribs (fd, B115200, 0);  // set speed to 115,200 bps, 8n1 (no parity)
set_blocking (fd, 0);                // set no blocking
write (fd, "hello!\n", 7);           // send 7 character greeting
usleep ((7 + 25) * 100);             // sleep enough to transmit the 7 plusreceive 25:  approx 100 uS per char transmit
char buf [100];
int n = read (fd, buf, sizeof buf);  // read up to 100 characters if ready to read
...
```

Tecniche di programmazione

Bootloader 80KB	Kernel 2-4MB	Librerie e Applicativi
• C / ASM • API specifiche • Librerie statiche • Codice statico, su HW • SM : ISR • Esecuzione lineare	• C / ASM • API specifiche • Librerie statiche • Codice statico, su HW • SM: ISR / TASK • Esecuzione concorrente	• C/C ++ (C#, JAVA, script,..) • API POSIX • Librerie dinamiche • Codice Dinamico / Astratto • SM: Processi / Thread • Esecuzione concorrente

I componenti del Kernel

Task Scheduler	Memoria Virtuale	File system
Device driver	Gestione degli ISR	Codice per architettura

Kernel space vs user space

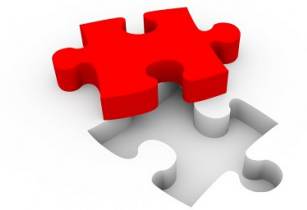
USER SPACE
LibC
KERNEL SPACE
HARDWARE

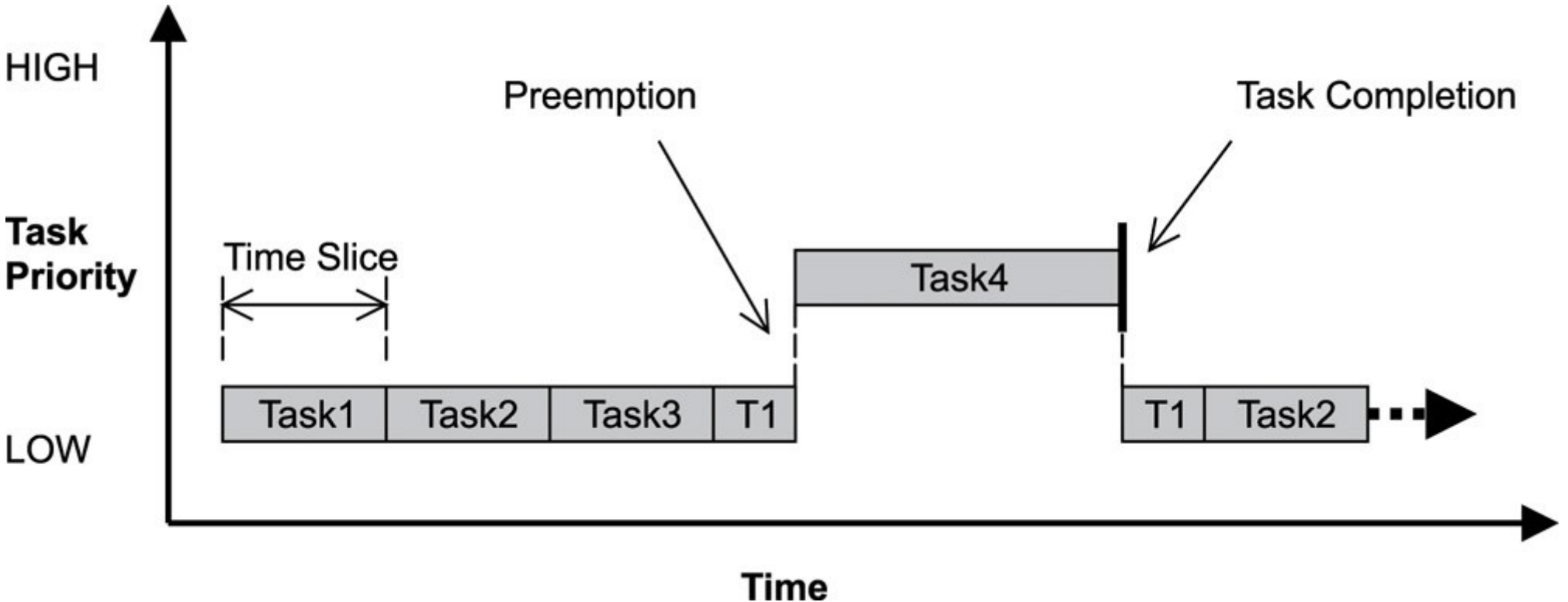
Task scheduler : punti importanti

- Ogni applicativo in esecuzione (processo) crede di avere la CPU in esclusiva
- Diverse code con priorità : da FIFO a ROUND ROBIN con e senza interruzione forzata (Pre-emption)
- Coda standard con priorità dinamica decisa dal SO
- „Time Slice“ = 100mS

*sched.c:118 #define DEF_TIMESLICE (100 * HZ / 1000)*

Task scheduler : come si comporta

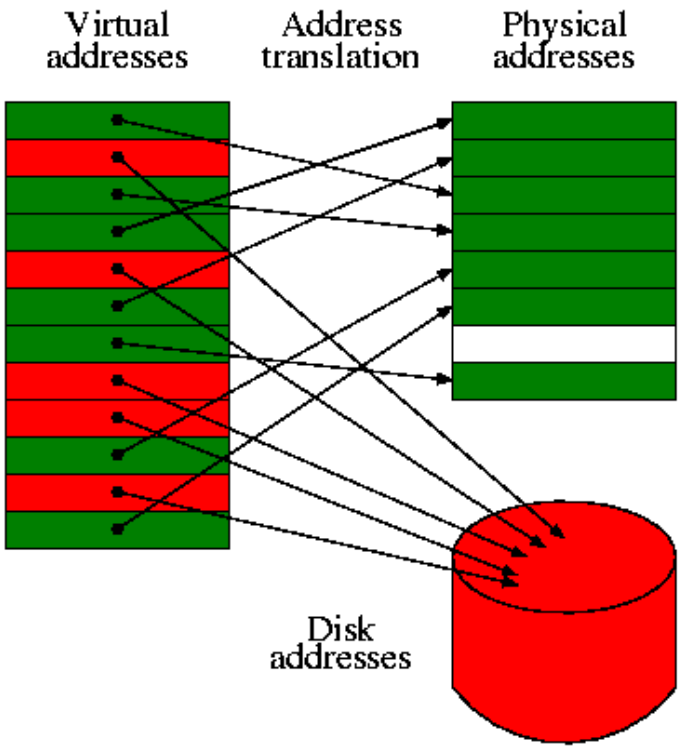




Memoria virtuale : punti importanti

- Solo su uC con MMU;
- Il sistema gestisce uno spazio di memoria virtuale maggiore di quella fisica;
- Ogni applicativo (processo + librerie) crede di avere uno spazio di memoria lineare esclusivo;
- La memoria fisica viene allocata solo se utilizzata, non quando viene richiesta con malloc;
- La maggior parte degli applicativi e librerie disponibili per Linux, quando eseguiti su un sistema embedded, allocano più memoria virtuale della memoria fisica disponibile

Memoria : fisica vs virtuale



- Il kernel linux sui sistemi embedded è lo stesso che gira sui sistemi desktop e server, quindi i meccanismi di gestione della memoria vituale sono gli stessi
- Sui sistemi embedded di norma non viene abilitata la partizione di swap

Memoria virtuale : overcommit

ATTENZIONE !!!

malloc(*mem_req*) != NULL

può essere vero anche per mem_req >> memoria fisica libera

Overcommit : un esempio

#sysctl -a | grep vm.overcommit

vm.overcommit_kbytes = 0

vm.overcommit_memory = 0

vm.overcommit_ratio = 50

Con queste impostazioni possiamo allocare al massimo 62.340kB (CommitLimit) per ogni malloc (_ratio=50%), ma possiamo allocare complessivamente più memoria virtuale di quella fisica libera nel sistema (_memory=0).

Con _memory = 2 impediamo di fatto l'overcommit

#cat /proc/meminfo

MemTotal:	124684 kB
MemFree:	117364 kB
MemAvailable:	118072 kB
Buffers:	1032 kB
Cached:	3132 kB
…	
CommitLimit:	62340 kB
Committed_AS:	2008 kB
VmallocTotal:	892928 kB
VmallocUsed:	2524 kB
VmallocChunk:	887228 kB

Memoria virtuale : OOM Killer

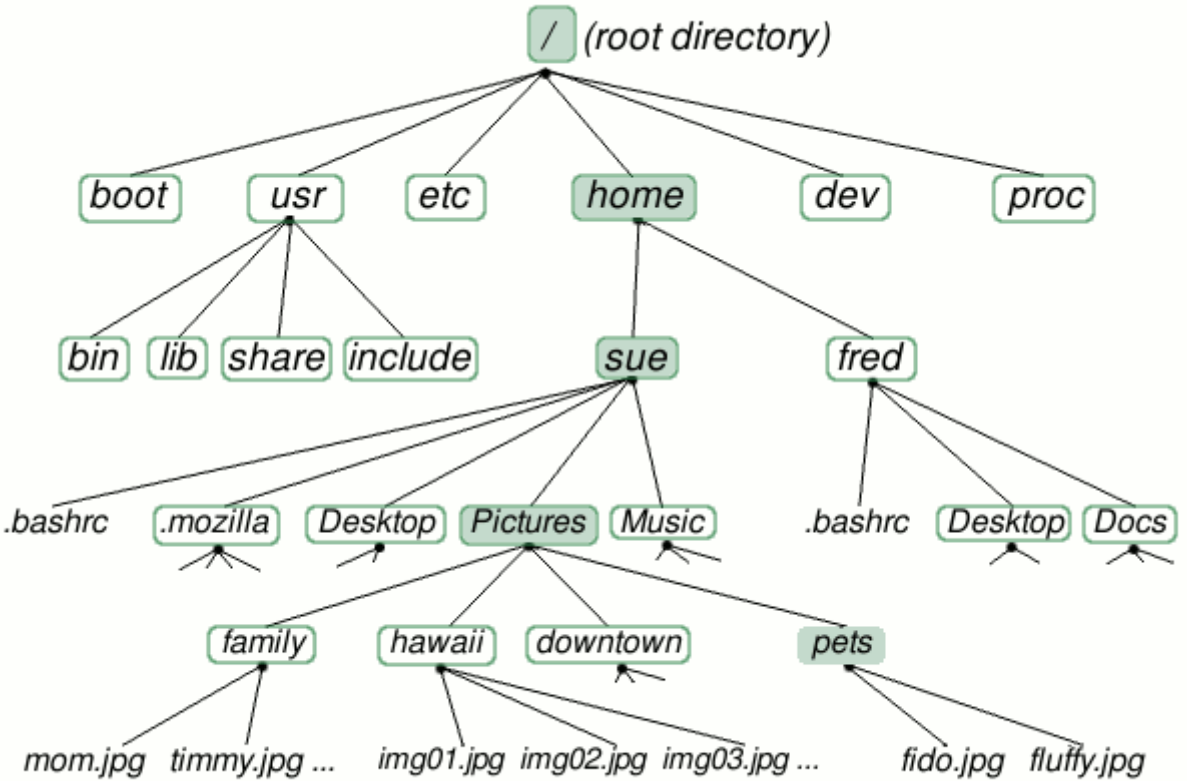
Cosa fà il sistema quando non ha più memoria fisica disponibile ?

- SWAP su disco delle pagine poco usate (NON su embedded)
- scarica pagine di codice eseguibile di processi
- Esegue il processo di sistema „OOM killer“ (Out Of memory) che termina uno o più processi con una logica „primitiva“

File System : punti importanti

- Virtualizza ogni informazione del sistema operativo;
- Non solo file con dati su disco, ma anche comunicazione tra processi (named pipes) e informazioni del kernel (es /proc);
- L'accesso concorrente ai file è gestito dal sistema operativo;
- Consente di vedere dischi esterni o filesystem di rete in modo omogeneo;

File System : vista generale



File System : alcune particolarità

- I file nascosti iniziano con il carattere “.” (es: .config);
- I file possono essere assegnati ad un solo proprietario e ad un solo gruppo
- I file possono avere la proprietà “eseguibile” che segnala al SO che i file possono essere eseguiti oppure, per le directory, che è possibile accedervi;
- I permessi di ogni elemento del filesystem sono questi :

rwxrwxrwx<proprietario>:<gruppo>

i permessi in rosso sono per “tutti gli altri” che non fanno parte del gruppo
e non sono il proprietario (es. rw-r---- cosa significa ??)

File System : il Load on Demand

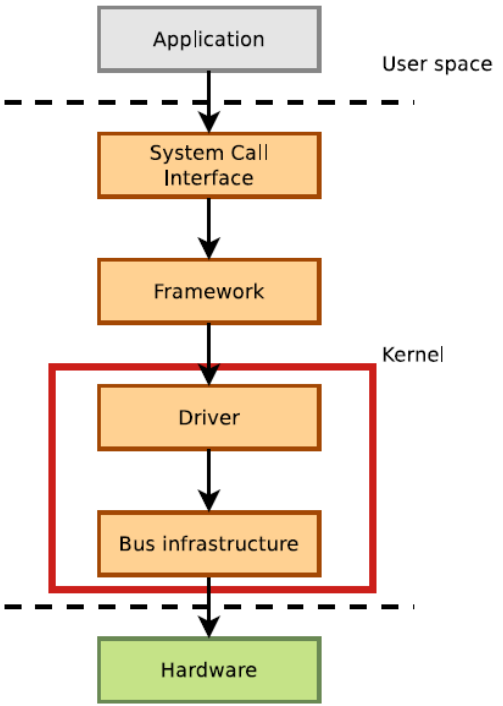
- Il Kernel di Linux carica i processi in memoria usando la VM e implementando il „Load on Demand“;
- Quando un processo viene eseguito, solo le porzioni di codice effettivamente percorso vengono caricate dal file system alla memoria virtuale;
- Questo permette una gestione ottimizzata della memoria anche se si utilizzando librerie generiche di grandi dimensioni;

Device driver

- Il kernel di linux viene eseguito su un'ampia selezione di architetture diffrenti, quindi è necessario che permetta di massimizzare la riusabilità del codice tra le diverse piattaforme hardware;
- Questo richiede un'organizzazione del codice pulita, dove i **device drivers** sono separati dai **controller drivers**, il codice specifico per l'architettura separato dai driver, e così via;
- In Linux viene quindi implementato uno specifico “device model” che deve essere rispettato quando si interviene per aggiungere o modificare codice esistente per adattalo alla propria architettura;

Device driver

- Nel kernel un driver è sempre interfacciato con un framework e un bus di infrastruttura;
- Il framework permette di esporre le funzioni dell'hardware in modo generico;
- Il bus di infrastruttura permette di comunicare con l'hardware;

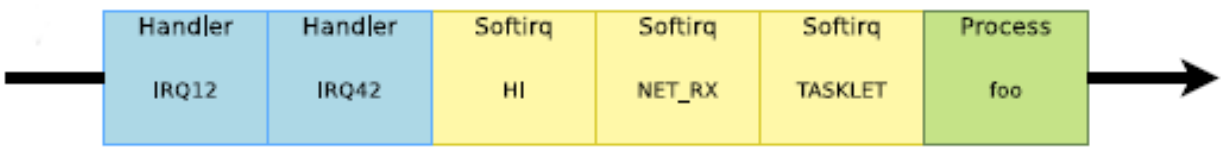


Interrupt : punti importanti

- Non possono trasferire direttamente dati da/per lo user space
- L'esecuzione dell'interrupt handler è gestita dalla CPU e non dallo scheduler
- Gli interrupt handler vengono eseguiti a interrupt disabilitati, devono quindi completare il loro lavoro ne più breve tempo possibile

Threaded Interrupt

- Dalla versione 2.6.30 è stato aggiunto il supporto ai threaded interrupt (TI)
- I TI Permettono di inserire una priorità nell'esecuzione dell'ISR, molto utile nei sistemi embedded con necessità di real time
- Usando i TI gli ISR vengono divisi in due parti:
  - Top half : il vero interrupt handler che deve terminare il prima possibile a interrupt disabilitati. Se possibile prende i dati dal device e attiva la parte Bottom half per processarli
  - Bottom half : la parte rimanente del lavoro dell'ISR che può essere svolta a interrupt abilitati



Il debug

Del kernel

Degli applicativi

- Strumenti differenti : ICD x Kernel, gdb x Applicativi
- Un breakpoint in debug del kernel „blocca“ tutti gli ISR ed i processi/thread
- Per il debug del kernel viene richiesta una capacità di trace e di decodifica degli indirizzi di memoria virtuali per poter seguire il codice separandolo per contesti di esecuzione
- gdb è „invasivo“ perché richiede il codice compilato in debug e falsa le condizioni di esecuzione

Il debug con ICD e gli applicativi

- I debug del kernel con ICD è caldamente consigliato e non dà problemi perchè il kernel viene caricato in memoria e la memoria è fissa;
- Nel debug degli applicativi con ICD è necessario disattivare il Load on demand applicando una patch manuale al kernel, altrimenti non sarà possibile aggiungere dei BP al codice se questo non è ancora stato caricato in memoria;

Analisi statica dei sorgenti

Controllo del formalismo
e
documentazione API

Controllo allocazioni
di memoria

Controllo della qualità del sw (indicatori)

Analisi statica dei sorgenti

Doxygen	cppcheck	cccc
Documentazione API	Controllo allocazioni della memoria	Controllo indici di qualità sw

Perchè analizzare i sorgenti ?

- Per generare la documentazione delle API con un minimo costo (Doxygen)
- Per verificare la corretta allocazione e rilascio della memoria (cppcheck)
- Per controllare il grado di complessità del codice (cccc) ed eventualmente intervenire per un refactory

Sul mercato si trovano anche ottimi strumenti commerciali !!

L'uso degli script

Molti comandi della shell sono script

A volte con uno script si risparmiano moltissime righe di codice C

Permettono la modifica ed il test senza compilazione

Disponibili in molte varianti

(es. C like)

Utilizzabili anche come web CGI su httpd busybox

Disponibilità di vari linguaggi interpretati

(Python, lua, tcl,..)

L'uso degli script : Shell

ESEMPIO DI SCRIPT PER L'IMPOSTAZIONE DEI PARAMETRI IP

```
#!/bin/sh

if [ $1 -eq 1 ] ; then
    udhcpc &
else
    ifconfig eth0 $2 netmask $3 up
    route add default gw $4 eth0
    echo -e nameserver $5 \\nnameserver $6 > /var/run/resolv.conf
fi
```

L'uso degli script :

ESEMPIO DI SCRIPT PYTON PER „Hello World“ su LCD Daysy24
In Python

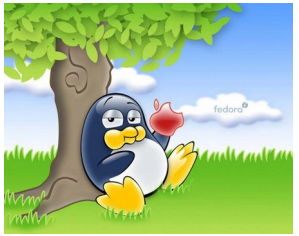
```
import ablib
import time

lcd = ablib.Daisy24()
lcd.backlighton()
lcd.putstring("Hello World !")
```

Esempi e tutorial con gli script :

http://www.acmesystems.it/tutorials
http://www.acmesystems.it/playground
https://github.com/tanzilli/playground

E questo è tutto …. per ora !!



Strumenti per avviare un progetto

Seminari ATMEL/EBV/Acmesystems

NerInformatica

Relatore : Luciano Neri

Libero professionista
Ordine degli Ingegneri di Vicenza

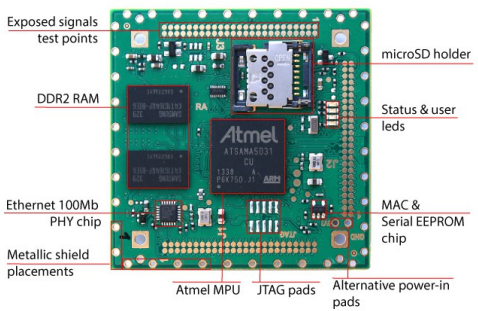
Agenda (30')

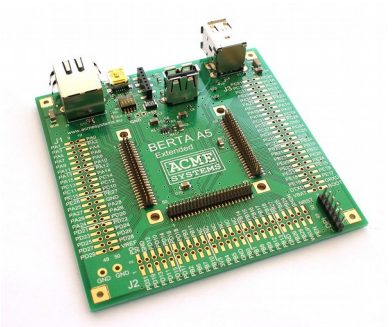
- Cenno agli strumenti necessari per avviare un progetto
- Componenti software di base: bootloader, kernel e rootfs; l’ecosistema GNU/Linux, toolchain, librerie;
- distribuzioni binarie vs compilazione dei sorgenti
- gli ambienti di sviluppo (IDE)
- l’adattamento del kernel su schede differenti usando il device tree
- il portale at91.com
- il controllo sui componenti del sistema embedded

Prima di tutto serve un hardware

- ATMEL XPLD

- Acmesystems ACQUA-A5





Riprendiamo i moduli sw base

… nelle giuste proporzioni ...

Bootloader

Kernel

200MHz

16MB Flash

32MB RAM

Applicativi e

Librererie

(il rootfs)

1,2GHz

16GB eMMC

2GB RAM

Cos'è l'ecosistema Linux ??

- In Linux è possibile trovare diverse librerie che eseguono sostanzialmente la stessa funzionalità di base ma con obiettivi diversi. es. libc, glibc, uClibc, eglibc;
- Questa flessibilità aumenta in modo esponenziale le possibilità di permutazione tra le varie liberie, generando un insieme elevatissimo di combinazioni;
- L'ecosistema Linux quindi è l'insieme di tutte le risorse software disponibili su internet;

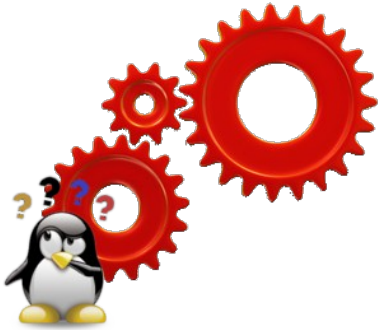
E quindi come posso scegliere ?

- Come abbiamo visto prima le combinazioni ottenibili combinando le librerie dell'ecosistema sono moltissime, **MA NON TUTTE FUNZIONANO !!!**
- Anche in Linux, i pacchetti software hanno dipendenze con librerie dinamiche specifiche;
- Per poter selezionare in modo semplificato delle combinazioni funzionanti di librerie che generino un sistema Linux funzionante, ci si affida alle distribuzioni binarie o a tools che permettono di generare l'intero sistema Linux partendo dai sorgenti;

Le librerie di base di Linux

- libc GNU C library
- libm C Math library
- libgcc_s gcc shared library
- libdl dynamic linking library
- librt realtime extension libray (per compatibilità POSIX)
- libcrypt encryption/decryption library
- libpam pluggable authentication module library
- libpthread POSIX thread library

E cosa compila tutto questo sw ?



- Per compilare tutti i componenti del sistema linux serve un [cross]compilatore che sia in grado di compilare sia bootload/kernel (firmware), che applicativi e librerie Linux;
- Stiamo parlando del gcc (GNU C Compiler) che a dispetto del nome non compila solo C e non solo per Linux;
- Il gcc, assieme ad altri tool software di supporto alla compilazione, prende il nome di toolchain;

Quindi :

bootloader + kernel + librerie
+ rootfs + toolchain

=

Linux (o GNU/Linux)

.. un aiutino ??

Le DISTRIBUZIONI Linux aiutano a gestire l'intero sistema

Binarie o da compilare?

DISTRIBUZIONI BINARIE:

- Sono generalmente disponibili su internet per il download in formato binario del rootfs;
- All'utente non resta che associare un kernel e un boootloader, a volte anche questo viene già fornito assieme all'immagine della memoria FLASH (es. SD);
- Permettono di scaricare pacchetti per nuove funzionalità o aggiornamenti del sistema con semplici comandi (es: apt-get install <pacchetto>);
- Permettono di utilizzare lo stesso sistema Linux disponibile su PC;
- Di solito sono vincolate ad una versione specifica di gcc

Binarie o da compilare ?

DISTRIBUZIONI DA COMPILARE:

- Sono generalmente disponibili su internet per il download come pacchetto da decomprimere sul proprio PC;
- Richiedono un PC Linux configurato con i tool di compilazione e con sufficienti risorse (HDD, uC, RAM);
- Molte permettono la configurazione del kernel e del rootfs in modo „sartoriale“ sia per l'occupazione di FLASH (8-512MB) che di RAM (32MB-2GB);
- Si può scegliere di compilare il toolchain della versione voluta

Alcune distribuzioni

BINARIE:

- Debian
- UBUNTU
- Yocto
- …

DA COMPILARE:

- Busybox
- Buildroot
- OpenWRT
- ...
- Yocto
- Ångström
- Openembedded
- ...

La distribuzione minimale

BUSYBOX:

BusyBox: The Swiss Army Knife of Embedded Linux

BusyBox combines tiny versions of many common UNIX utilities into a single small executable. It provides replacements for most of the utilities you usually find in GNU fileutils, shellutils, etc. The utilities in BusyBox generally have fewer options than their full-featured GNU cousins; however, the options that are included provide the expected functionality and behave very much like their GNU counterparts. BusyBox provides a fairly complete environment for any small or embedded system.

BusyBox has been written with size-optimization and limited resources in mind. It is also extremely modular so you can easily include or exclude commands (or features) at compile time. This makes it easy to customize your embedded systems. To create a working system, just add some device nodes in /dev, a few configuration files in /etc, and a Linux kernel.

BusyBox is maintained by Denys Vlasenko, and licensed under the GNU GENERAL PUBLIC LICENSE version 2.

Distribuzioni: Pro/Contro (IMHO)

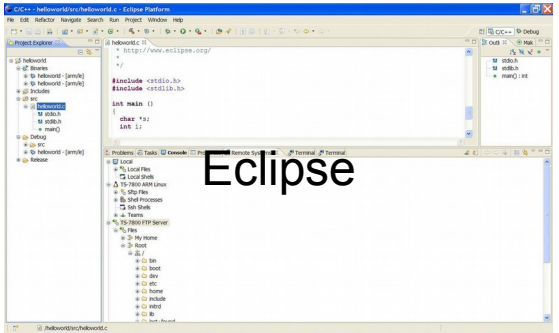
BINARIE:

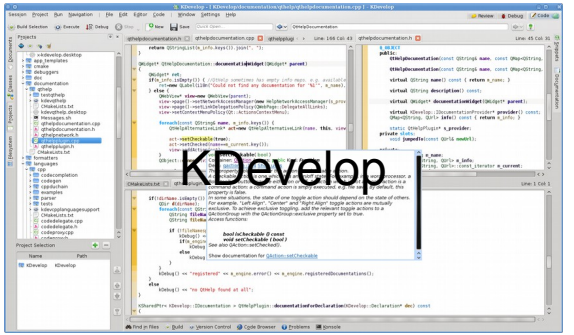
- Minimo tempo per la creazione di un rootfs
- Stabilità garantita del supporto alla distribuzione
- Aggiornamenti automatici
- Toolchain standard
- Installazione nuovi pacchetti sw e dipendenze
- Comportamento come PC
- Difficile controllo delle licenze
- Esecuzione di molti servizi a volte superflui
- Versioni delle librerie non facilmente modificabili
- Debug del sistema
- Dimensione del rootfs

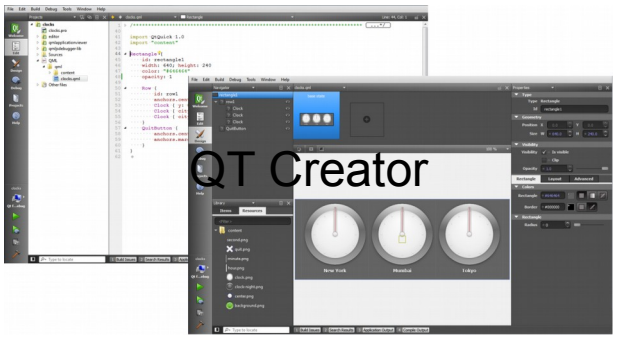
DA COMPILARE:

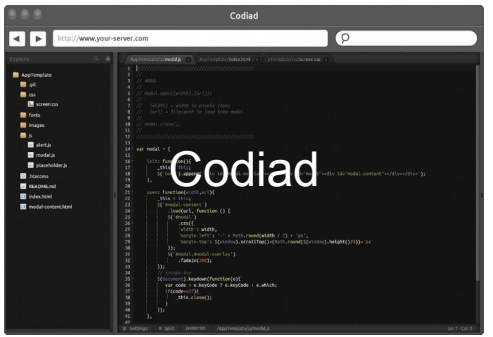
- Personalizzazione
- Intero sistema sotto controllo dello sviluppatore
- Possibilità di creare una toolchain personalizzata
- Debug del sistema
- Comportamento da „Sistema Embedded“
- Verifica delle dipendenze non sempre automatico
- Tempo di compilazione
- Aggiormamento del sistema non sempre semplice
- Integrazione di nuovi pacchtti in sorgente non sempre semplice e veloce

Alcuni ambienti di sviluppo (IDE):









Eclipse

- Nato per lo sviluppo di applicativi JAVA
- Estensione delle funzionalità mediante plugin
- In seguito sono state creati plugin per la programmazione in altri linguaggi (es. CTD per C/C++)
- Utilizzato come base per moltissimi IDE commerciali
- Portabile (Windows, MacOsX, Linux, …)
- Facilmente integrabile con vari toolchain

QT Creator

- Nato per lo sviluppo di applicativi QT
- Integra l'ambiente di progettazione della GUI
- Utilizza qmake
- Si può utilizzare per sviluppare codice C/C++ senza framework QT
- Integrabile con tools per il controllo delle versioni (cvs, svn, ...)
- Portabile (Windows, MacOsX, Linux, …)
- Facilmente integrabile con vari toolchain

KDevelop

- Nato per lo sviluppo di applicativi C/C++ in Linux
- Compilato nativamente per Linux
- Molto veloce
- Integrabile con tools per il controllo versione (cvs, svn, ..)
- Facilmente integrabile con vari toolchain

Codiad

- Nato per lo sviluppo di applicativi via web
- Estensione delle funzionalità mediante plugin
- Supporta molti linguaggi
- Integrabile con tools per il controllo versione (cvs, svn, ..)
- Può essere utilizzato come IDE per la personalizzazione di pagine web su un sistema embedded

http://www.acmesystems.it/codiad

Adattare il kernel: Device Tree

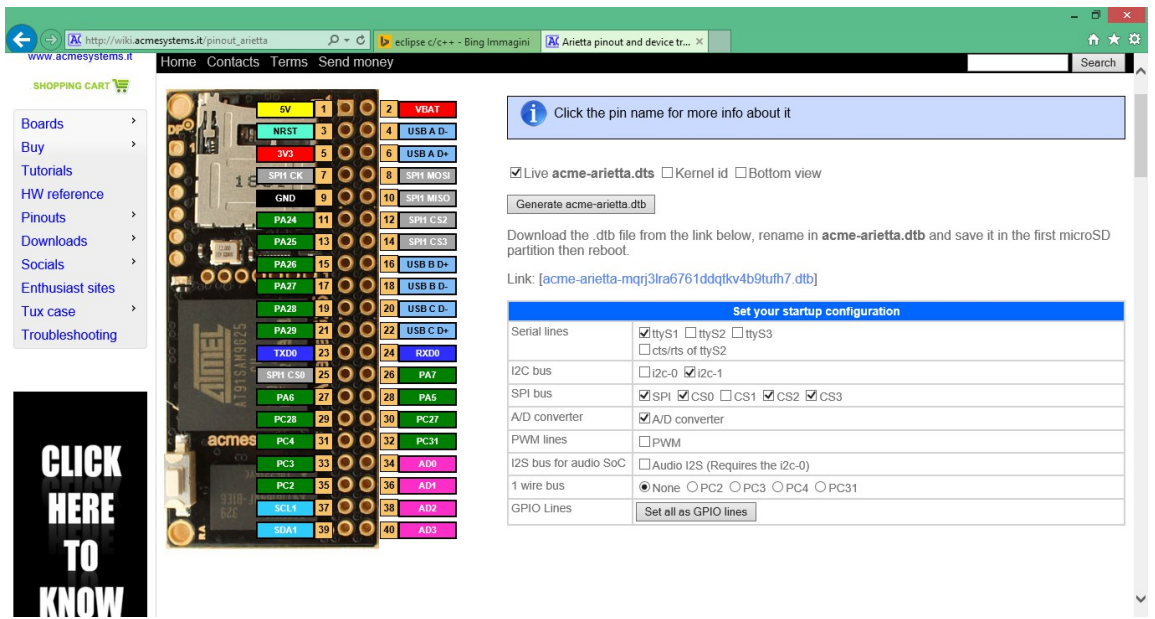
Il Device tree è una struttura dati che descrive l'hardware, il multiplexing dei pin, i parametri e altri dati che vengono passati al kernel allo startup;

Permette quindi di modificare i collegamenti del uC senza dover ricompilare il kernel;

Non risolve tutti i problemi … ma semplifica i piccoli adattamenti

Fare degli esperimenti sul DT

Andare sul sito : http://wiki.acmesystems.it/pinout_arietta



Il portale www.at91.com

- Punto di riferimento per il download dei sorgenti di Linux, Android e Windows CE (bootstrap, bootloade, kernel, rootfs)
- Forum e discussioni tecniche
- Wiki e howto pratici e ben documentati
- FAQ aggiornate
- Immagini binarie demo per prove veloci delle schede di valutazione

Il controllo sui componenti

- Progettare e sviluppare un prodotto basato su Linux Embedded richiede il completo controllo di tutti i componenti hardware e software,
- in ogni modulo infatti può verificarsi un problema mai riscontrato in altri sistemi simili,
- deve essere sempre possibile identificarlo e correggerlo ..
- ... in tempi certi !!!

Il controllo sui componenti SW

- Le distribuzioni da compilare (es. Buildroot) permettono di generare l'intero sistema compilandolo dai sorgenti,
- questo permette di poter fare debug dell'intero sistema sia con gdb che con un ICD,
- permette inoltre di applicare le patch sul pacchetto o di aggiornare il modulo software con una versione più recente che non presenta il problema

Il controllo sui componenti SW

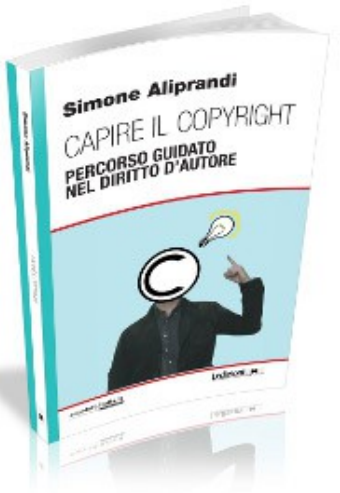
- Inoltre, l'utilizzo delle distribuzioni da compilare permette di avere una lista ridotta ed essenziale delle librerie utilizzate dal sistema e dal programma,
- questo permette di non violare la licenza GPL con il proprio programma proprietario del quale non si vuole rilasciare i sorgenti,
- Mantenendo un ridotto numero di componenti software si riducono anche le possibilità di effetti collaterali e „bachi“

Il controllo sui componenti HW

- Una scelta attenta dei componenti di contorno del microcontrollore e dei bus permette di ridurre le problematiche software e di rendere più efficiente il sistema Linux;
- Il tempo di sviluppo di un nuovo driver può essere ridotto, ma il tempo di test necessario a garantirne la stabilità è comunque sempre elevato;
- E' necessario ridurre al minimo le problematiche legate a frequenze di BUS elevate / PCB / temperatura / EMI perché questi problemi non sono facilmente diagnosticabili da software

(ma di solito questi problemi vengono inizialmente imputati al sw)

Per approfondire il tema „Licenze“

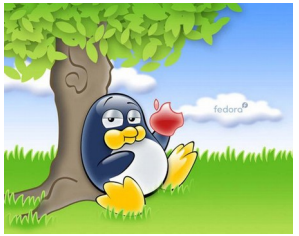


Consiglio gli ottimi libri di Simone Aliprandi :

„Capire il Copyright – Ed. 2012“

„The International Free and Open Source Software Law Book“

E questo è tutto …. per ora !!



QT per i sistemi embedded

Seminari ATMEL/EBV/Acmesystems

NerInformatica

Relatore : Luciano Neri

Libero professionista
Ordine degli Ingegneri di Vicenza

Agenda (30')

- Perchè usare QT
- QT5 vs QT4
- QT come framework di astrazione del sistema operativo
  - panoramica sulle API di gestione dei thread
  - file e classi di utilità
- l’integrazione di QT nei sistemi embedded
- Lo sviluppo multi-piattaforma
- QT Creator: l’IDE per i progetti QT
- La gestione della grafica usando i widget e QT Quick 2
- QT5 e le gestures con e senza EGL

Una domanda : perchè usare QT ?

Nella mia applicazione embedded non uso la grafica, quindi QT non mi serve !!

Inoltre, perchè dovrei complicarmi la vita usando una libreria enorme, in C++ per fare quello che posso fare con codice in POSIX/C ?

... Ok, possiamo chiudere qui la presentazione su QT … o no ?

Cosa offre il framework QT

QT è un framework multi piattaforma per lo sviluppo di applicazioni e rende disponibili molti moduli sw :

- QtCore
- QtNetwork
- QtXml
- QtXmlPatterns
- QtScript
- QtSql
- QtGui
- QtMultimedia
- QtOpenGL
- … altro ancora

Quali vantaggi mi offre

- Usare il C++ come linguaggio compilato;
- L'approccio OOP omogeneo dell'intero framework QT;
- La gestione degli eventi con signal e slot
- La documentazione del framework;
- La facilità di trasporto dell'applicazione su altre piattaforme o sistemi operativi;
- Il facile riutilizzo del codice sviluppato;
- La licenza LGPL che permette l'uso con sw proprietario;

E i contro ??

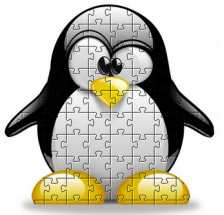
- Non lo possiamo usare dove abbiamo vincoli stringenti sulle prestazioni, perchè non è semplice risalire alla complessità del codice eseguito dal framework;

- Non lo possiamo usare su sistemi Linux Embedded con scarse risorse :
  - es. ARM9/200MHz, 16MB NOR FLASH, 32MB RAM

Le librerie di QT possono occupare da qualche MB a qualche decina di MB

NOTA: QT è un'insieme di librerie che occupano spazio in FLASH nel filesystem, ma che vengono caricate in memoria solo nelle parti che servono effettivamente (Load on Demand)

QT Multi OS



Sistemi operativi supportati, nella versione enterprise

- Windows
- Linux
- Mac OS X
- Embedded Linux
- Embedded Android
- Windows Embedded
- VxWorks
- QNX
- INTEGRITY
- Android
- iOS
- WinRT

Signal/Slot: eventi tra oggetti

Signal/Slot è un meccanismo base di tutto il framework QT

- Gli oggetti possono dichiarare degli Slot, funzioni chiamate in risposta ai „Signal“ emessi da altre classi;
- Gli oggetti possono emettere dei „Signal“, eventi che possono avere anche dati associati;
- Un „Signal“ di un oggetto può essere connesso a uno o piu “Slots“ compatibili a livello di definizione

Questo meccanismo permette la gestione semplice e veloce degli eventi tra gli oggetti dell'applicativo.

Molte classi QT utilizzano Signal e Slot per gestire gli eventi

Signal / Slot : un esempio

```
class Consumer : public QObject {
Q_OBJECT
...
public slots:
void ConsumeData(int identifier);
...
};
void Consumer::ConsumeData(int identifier){
    /* consumo il dato ricevuto */
}
```

```
class Producer : public QObject {
Q_OBJECT
...
signals:
void newIdFromSerial(int identifier);
...
};
void Producer::readAndDecodeCharFromSerial(void){
int id;
...
emit newIdFromSerial(id);
...
}
```

```
Consumer cons;  Producer prod;
QObject::connect(&cons, SIGNAL(newIdFromSerial(int)), &prod,SLOT(ConsumeData(int)));
```

QT in azione

VEDIAMO ALCUNI ESEMPI

Gestione dei Thread in QT

```
/* DEFINISCO UNA CLASSE PER IL THREAD */



class MyThread : public Qthread
{
    Q_OBJECT

 protected:
    void run();
 };

 void MyThread::run()
 {
    While(1){
        blink_led();
        Sleep(1);
     }
 }
```

```
/* IL MIO MAIN */

#include <QThread>
#include “MyThread.h”

Int main() {
..
..
    MyThread mt;
    mt.start();
..
..

}
```

Gestione dei Thread in Linux

```
#include <pthread.h>
Int variable_altro_thread;

void * thread1()
{
        while(1){
                blink_led(); sleep(1);
        }
}

int main()
{
        int status;
        pthread_t tid1;
        pthread_create(&tid1,NULL,thread1,NULL);
        pthread_join(tid1,NULL);
        return 0;
}
```

La differenza fondamentale è nella visibiltà delle variabili dal thread:

- In QT è limitata dalla classe
- In POSIX/C è il campo di visibilità della funzione

Gestione dei processi

```
/* creo il processo */
p = new Qprocess();

/* collego un evento all'exit del processo */
connect(p, SIGNAL(finished(int, QProcess::ExitStatus)),
this, SLOT(myDone(int, Qprocess::ExitStatus)));

/* eseguo il processo */
p->start("connect_modem.sh");
..
..
/* devo terminare il processo ?? */
p->terminate();
```

E' estremamente semplice creare processi in QT

L'esecuzione di processi e script è fondamentale per poter interagire con il sistema

es. ifup wan

Connessione HTTP

```
/* creo l'oggetto */
network = new QNetworkAccessManager();

/* connessione all'HTTP server */
QNetworkRequest httpReq(QUrl("http://mioserver.it"));
reply = network->post(httpReq, dati.toAscii());
connect(reply, SIGNAL(finished(void)), this, SLOT(terminato(void)));
```

Server TCP

```
/* qui attendo la connessione */


MyTcpServer::MyTcpServer(void)
{
srv = new QTcpServer(this);
srv->listen(QHostAddress::any, 40444);
connect(srv, SIGNAL(newConnection()), this,
SLOT(acceptConnection()));
}
```

```
/* qui accetto la connessione e attendo i dati */


void MyTcpServer::acceptConnection(void)
{
QTcpSocket *sk = srv->nextPendingConnection();
connect(sk, SIGNAL(readyRead()), this,
SLOT(readClient()));
}
```

Timers

```
class MyClass : public QObject
{
    Q_OBJECT
    private slots:
        void periodicTimer(void);
    private:
        QTimer timer;
}
```

```
MyClass::MyClass()
{
    connect(&timer, SIGNAL(timeout()),
            this, SLOT(periodicTimer()));
    timer.start(1000);
}
void MyClass::periodicTimer(void)
{
    /* chiamata ogni secondo */
    blink_led();
}
```

Decodifica XML

```
QFile *f = new Qfile(“mydata.xml”);
f->open(QIODevice::ReadOnly | QIODevice::Text);
dom = new QDomDocument();
dom->setContent(f);

/* poi si decodifica con le classi:
   QDomNodeList, QDomNode e QDomElement    */
```

QT nei sistemi embedded

- QT 5.x può essere integrato in sistemi Linux su uC ARM9 e CORTEX-A7, anche con QT Quick v1.x
- La cross-compilazione di QT 5 su un sistema embedded è un'operazione di “sartoria” perché deve essere adattata alle funzioni necessarie all'applicazione

QT e lo sviluppo multi-piattaforma

QT utilizza un proprio sistema di compilazione chiamato qmake

- Molto semplice se confrontato con i makefile
- Un file .pro descrive i sorgenti del progetto e le librerie; il makefile viene generato automaticamente da qmake in base a queste informazioni
- Per la cross-compilazione è necessario avere un file qmake legato alla versione compilata per il target

QT Creator: l'IDE per i progetti QT

- IMHO, uno dei migliori IDE disponibili su Linux
- Le ultime versioni permettono un'altissima flessibilità di configurazione, che facilita le operazioni di cross-compilazione e debug sulla scheda target
- Si integra nativamente con i sistemi di controllo delle versioni
- La versione base è gratuita ma sono disponibili versioni con maggiori funzionalità a pagamento
- L'ambiente standard permette comunque un buon grado di efficienza nella gestione dei progetti software

La grafica con i widget e QT Quick

- Provare a sviluppare un'interfaccia HMI moderna con le API e i widget C++ di QT è un'attività estremamente ardua
- L'uso del linguaggio QML (Quick v1.x) permette di ottenere risultati simili alle moderne UI presenti nella maggior parte dei dispositivi di consumo
- L'utilizzo della nuova struttura Quick v2 permette il rendering di scene quasi realistiche con animazioni e fluidità eccezionali (se supportate da EGLS)

QT5 e le gestures

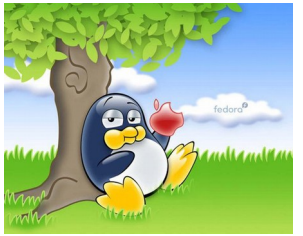
Sfatiamo un mito : sia le QT 4.8.x che le 5.x supportano il multi-touch (con QT 4.8.x è un po' più complicato)

E' da verificare se il kernel, il driver del touch screen e la libreria user space (es. tslib) supportano gli eventi multi touch

L'uso dei touch screen capacivi aumenta la user experience sopratutto per l'uso del multi touch

Le QT 5, utilizzando il nuovo QT Quick assieme alla gestione degli eventi, può dare ottimi risultati rispetto alla versione 4.8.x

E questo è tutto …. davvero !!



.. ma restiamo a disposizione per le demo e gli approfondimenti