The Embedded GNU/Linux Planet

Francesco P. Lovergine
<frankie@lugbari.org>

LD2005











...Telecamere digitali
Smartphone
Palmari
Tablet PC
Router
Network appliances
Telefoni e centralini VoIP
Centraline avanzate
Sistemi a GPS...

















Il mondo Embedded

Un sistema Embedded

Letteralmente sistema IMMERSO, in generale indica un sistema molto lontano da un computer desktop o laptop che sia, e che si distingue per alcune caratteristiche più o meno costanti:

- Mono applicativi e comunque NON general purpose
- Con risorse (memoria e storage) estremamente limitate
- Con dispositivi di I/O mediamente esotici o custom
- Con limitate capacità di espansione
- Con architetture specifiche (es. PC104)
- Con processori specifici a basso consumo e performance (es. ARM)
- Usati per controllare dispositivi speciali
- Che richiedono lo sviluppo di driver e moduli software di basso livello ad hoc (spesso a kernel level)

Perché un sistema operativo?

- Perché è più semplice
- Perché il tempo di sviluppo è minore
- Perché molto lavoro è già fatto a livello di kernel
- Perché se non si deve lavorare in pochi Kbyte non vale la pena usare sistemi custom e riscoprire l'acqua calda
- Perché l'ambiente di sviluppo è già familiare
- Perché certe cose funzionano meglio se c'è un sistema operativo sotto che se ne occupa
- Per lo stesso motivo per cui nessuno lavora più in assembly se non ci è assolutamente costretto

Perché GNU/Linux?

- Perché no?
- Perché è più leggero
- Perché è estremamente customizzabile
- Perché è portato su 11 architetture di processore diverso e qualcosa vorra dire...
- Perché ci sono i sorgenti di tutto e soprattutto anche del kernel
- Perché è indipendente dai vendor
- Perché le varie generazioni di kernel sono estremamente stabili
- Perché se sta in giro da 14 anni qualcosa vorrà dire...
- Perché Windows CE... beh lasciamo perdere...
- Perché il kernel è pensato per interfacciarsi con dispositivi esotici
- Perché esistono anche versioni real-time all'occorrenza
- Perché i costi di sviluppo sono minori nella maggior parte dei casi

Perché **non** GNU/Linux

- Perché incompatibili con la licenza GPL/LGPL
- Perché ci sono prodotti che rispondono meglio alle esigenze applicative
- Perché ~2MB di RAM sono troppi
- Per motivi di marketing o decisioni di management

La dotazione fondamentale

- Una development host nella architettura più convenieniente
- Una target host prototipale dotato delle interfacce più comode per lo sviluppo (es. seriale)
- Molta pazienza !
- La documentazione essenziale per il controllo delle schede ed interfacce

In alcuni casi ci si dovrà avvalere di una piattaforma di sviluppo prefabbricata (es. Montavista) e/o parzialmente proprietaria.

I mattoni software fondamentali

- Un bootloader per il target host
- Un kernel con la configurazione adeguata per il target host
- Una (cross) libreria C (C++) per il due host
- Un (cross) compilatore C (C++) per il dev. host
- Un target root fs con tutti i programmi e librerie fondamentali per il target host (sistema base)
- Un intero set delle (cross) librerie adeguate alla/e applicazione/i per i due host
- Un debugger (gdb) “remotizzabile” per i due host

Il BIOS del target

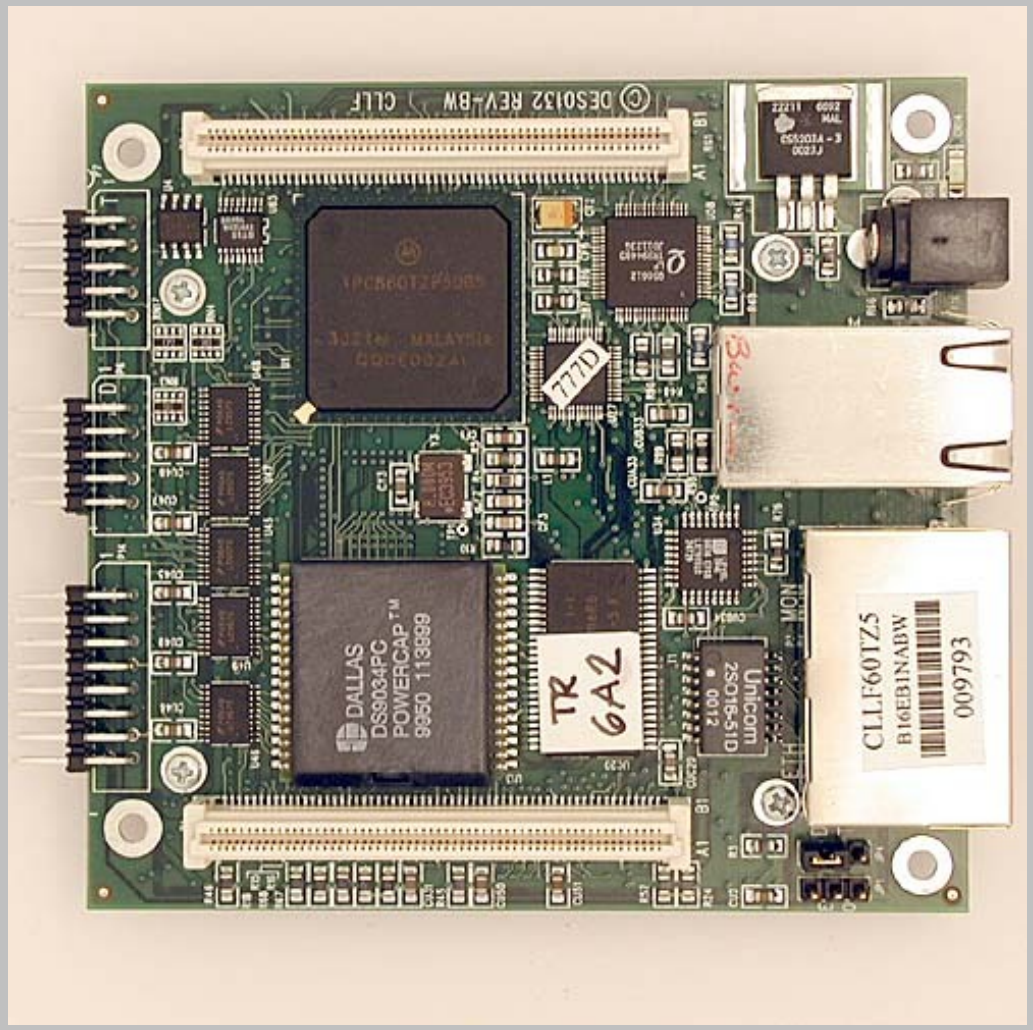
I firmware di controllo del target (pre boot-loader) cambiano a seconda della architettura della macchina.

In ambito i386 e compatibili i BIOS convenzionali sono
MOLTO LENTI (20 sec per il boot)

Esiste un progetto LINUX BIOS che sostituisce il firmware per specifiche schede. Nella maggioranza dei casi
il vendor dell'hardware ha sviluppato una soluzione ad hoc per il caricamento del boot loader,
anche da dispositivi “esotici” per esempio ethernet, usb o seriale, oltre che dai dispositivi storage (memorie flash in genere) propriamente detti e bisogna adattarsi a questo.

Nella maggior parte dei casi, è necessario comunque un boot a doppio passo, caricando il kernel da un apposito storage esterno.

Un esempio EP RPX-CLLF



... il boot

```
PlanetCore Boot Loader v1.02
Copyright 2000 Embedded Planet. All rights reserved.
DRAM available size = 16 MB
wvCV
DRAM OK
Autoboot in 2 seconds.
ESC to abort, SPACE or ENTER to go.
PlanetCore Diagnostics and Utilities - Release 2.01
pcd#q
>tftp
Load using tftp via Ethernet
Enter server IP address <192.168.1.11> : 192.168.1.11
Enter server filename <powerpc-zImage> : powerpc-zImage
Enter (B)inary or (S)record input mode <B> : B
Enter address offset : <FFF80000 hex> : 400000
.1.2.4.8.16.32.64.128.256.512.1024.2048.4096
Total bytes = 2313209 in 4136171 uSecs
Loaded address 00400000 through 00634BF8.
Start Address = 00400000
>
>go
```

Il booting

In molti casi il ROM monitor o firmware della scheda sarà preconfigurato per effettuare il loading automatico del codice a partire da un indirizzo prefissato da una memoria flash, opportunamente programmata per contenere le due componenti essenziali del sistema:

Il kernel
Il filesystem di root

Alcuni sistemi hanno una interfaccia IDE per CF e quindi vedono una unità flash come normale disco IDE. In tale caso il sistema funzionerà in tutto e per tutto come un normale sistema non-embedded e si dovrà usare un boot loader convenzionale, per esempio

LILO
Grub
U-Boot (PPC, MIPS, ARM)
RedBoot (x86, SH, PPC, v850, ARM, MIPS)
Blob, Angel, Aleph, Smartfirmware, ...



RAM FS

Nella maggior parte dei casi, l'immagine del root fs sarà semplicemente caricata in RAM come initrd (2.4+) o initramfs (2.6+) e utilizzata in questo modo. L'area storage flash potrà essere usata per mantenere informazioni persistenti:

- aggiornamenti del kernel
- aggiornamenti del rootfs
- informazioni di configurazione
- dati di lavoro

```
-rw-r--r-- 1 root    root    616K 2005-02-13 11:24 kernel
-rw-r--r-- 1 root    root    4,3M 2005-02-13 11:24 rootfs
```

I passi fondamentali: toolchain

Nella ipotesi di avere la massima libertà nella scelta delle componenti di creazione dell'ambiente per il target, i passi fondamentali da seguire saranno:

Nel caso di architetture development e target differenti: creazione di un toolchain di crosscompilazione, comprendente kernel compilato, cross-compilatore e cross-libreria C

Nel caso di architetture identiche: si potrà utilizzare il toolchain di compilazione ordinario della distribuzione di development SE adeguata. In generale potrebbe essere necessario comunque creare un toolchain di compilazione per il target, per esempio usando una libreria diversa dalla GNU libc come libreria base, oppure utilizzando una specifica selezione di versioni per gcc e libc.

Imprescindibile la creazione di una configurazione adeguata per il kernel

I passi fondamentali: base

In generale nella costruzione del rootfs si dovranno compilare from scratch i pacchetti di base.
Può essere conveniente utilizzare la base di una distribuzione convenzionale come partenza, opportunamente strippata di quanto non necessario.

Nel contesto Debian sono estremamente utili:

Alien
Debootstrap (fino a sarge)
Cdebootstrap (da etch)

La fase di “strippaggio” manuale è però imprescindibile, una volta collezionati i package necessari:

Eliminazione manuali
Eliminazione cache
Eliminazione sistema di packaging
Eliminazione librerie superflue
Eliminazione package superflui
Eliminazione configurazioni / init scripts superflui

I passi fondamentali: base (2)

Diversi programmi base possono essere sostituiti dalla compilazione di

Busybox (da 150KB a 600KB)

che crea un eseguibile monolitico per fornire una serie di programmi di base in modo molto conveniente/efficiente/compresso.

addgroup, adduser, adjtimex, ar, arping, ash, awk, basename, bunzip2,busybox, bzcat, cal, cat, chgrp, chmod, chown, chroot, chvt, clear, cmp,cp, cpio, crond, crontab, cut, date, dc, dd, deallocvt, delgroup, deluser,devfsd, df, dirname, dmesg, dos2unix, dpkg, dpkg-deb, du, dumpkmap,dumpleases, echo, egrep, env, expr, false, fbset, fdflush, fdformat, fdisk,fgrep, find, fold, free, freeramdisk, fsck.minix, ftpget, ftpput, getopt,getty, grep, gunzip, gzip, halt, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifup, inetd, init, insmod,install, ip, ipaddr, ipcalc, iplink, iproute, iptunnel, kill, killall,klogd, lash, last, length, linuxrc, ln, loadfont, loadkmap, logger, login,logname, logread, losetup, ls, lsmod, makedevs, md5sum, mesg, mkdir,mkfifo, mkfs.minix, mknod, mkswap, mktemp, modprobe, more, mount, msh, mt,mv, nameif, nc, netstat, nslookup, od, openvt, passwd, patch, pidof, ping,ping6, pipe_progress, pivot_root, poweroff, printf, ps, pwd, rdate,readlink, realpath, reboot, renice, reset, rm, rmdir, rmmod, route, rpm,rpm2cpio, run-parts, rx, sed, seq, setkeycodes, sha1sum, sleep, sort,start-stop-daemon, strings, stty, su, sulogin, swapoff, swapon, sync,sysctl, syslogd, tail, tar, tee, telnet, telnetd, test, tftp, time, top,touch, tr, traceroute, true, tty, udhcpc, udhcpd, umount, uname,uncompress, uniq, unix2dos, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, watch, watchdog, wc, wget, which, who, whoami, xargs,yes, zcat

I passi fondamentali: base (3)

La scelta del tipo di filesystem per il rootfs è legata all'impiego o meno dello stesso in ramfs o meno. Nel primo caso potrebbe essere adeguato un filesystem a sola lettura come il cramfs.
Nel secondo caso è imprescindibile l'impiego di un filesystem journaled, come ext3, reiserfs o quanto altro.

Si consideri però che i fs journaled possono essere particolarmente onerosi (doppio logging) per sistemi di memoria flash. Nel caso di tecnologia MTD/MMC, è in generale indicato e più efficiente il jffs2 di RedHat o il ffs di LinuxWorks.

La scelta del fs può essere importante/obbligata anche in base al bootloader che si intende o si deve usare.

GNU libc o NO?

In caso di requirement stringenti di memoria si può sostituire la libc che in media occupa 1.5MB di RAM con una libreria C più light e abbastanza completa (1/3 delle dimensioni o meno):

ucLibC
dietLibC

Il toolchain dovrà essere compilato opportunamente per utilizzare una di tali librerie. Potrebbero però sorgere problemi di compatibilità con il resto del toolchain o con le applicazioni...

Si noti che per la compilazione del kernel non è necessario utilizzare lo stesso compilatore usato per il resto del sistema. Il kernel richiede un compilatore 2.95, anche se compila *in genere* bene anche con una versione 3.x

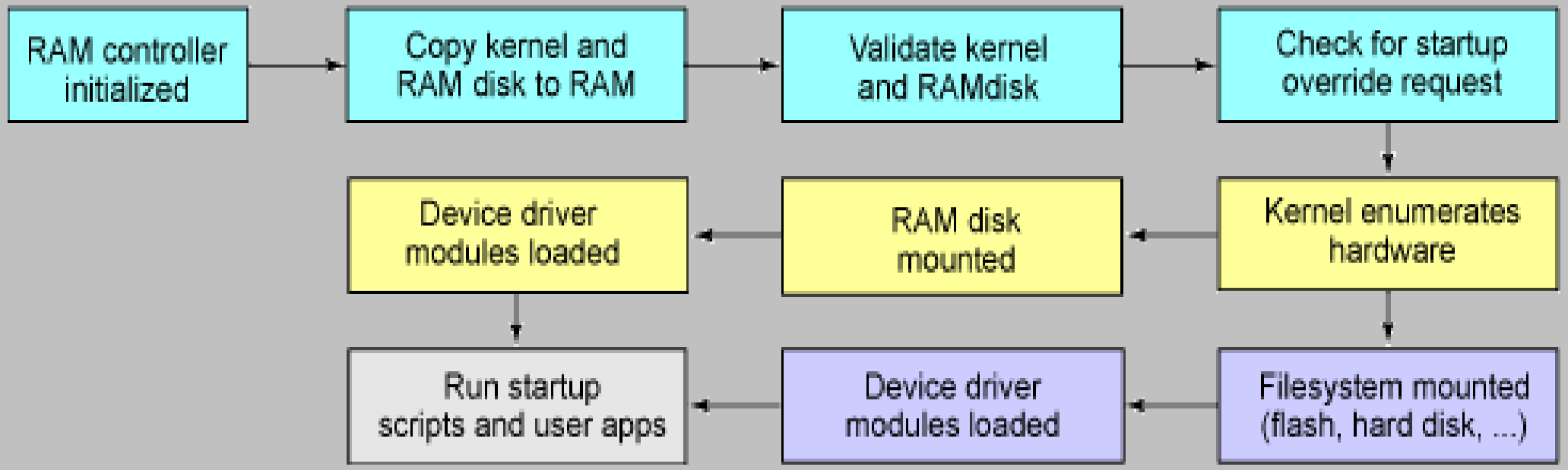
“Per via di levare” o “per via di aggiungere” ?

Tre approcci alla creazione del sistema base (in ordine di complessità):

- Installare tutti i package base di una distribuzione e poi eliminare il superfluo (non applicabile in caso di cross-compiling).
- Installare un pool di package di una distribuzione (by alien) e creare le necessarie configurazioni e directory/file di run
- Definire un pool di software da installare, compilare e installare tutto from scratch, generando finanche i necessari file di device. Si pone il problema di combinare versioni compatibili fra loro e compatibili con l'architettura.

In tutti i casi è opportuno realizzare propri script di build !

Il boot (semplificato)



Se non diversamente configurato, il initrd veine decompresso e caricato in RAM, il kernel lo usa come rootfs ed esegue il programma/script linuxrc. Quest'ultimo tipicamente esegue un pivot_root e passa il controllo a init da dove il run del sistema vero e proprio tipicamente prosegue. Init legge il file /etc/inittab.

La maggior parte dei sistemi embedded prosegue semplicemente in RAM il proprio lavoro

Le applicazioni

Nella maggior parte dei casi le applicazioni da aggiungere al sistema base sono strettamente dipendenti dal tipo di sistema che si sta mettendo in piedi.
Esistono naturalmente (a parte busybox) alcuni package da considerare per ambienit “tiny” che si possono prendere in considerazione, sia free, sia proprietari:



- Boa
- thttpd
- tiny-perl
- Qtopia
- Qtembedded
- Vari flavor di scripting light
- Tiny X
- Environment grafici non X
- ...

Ognuno può avere limitazioni importanti e comunque bisogna sempre tenere conto delle specifiche problematiche dipendenti dalla assenza di VM, di RAM in quantità e spesso anche di storage.

Demo & Question section



Copyright (c) 2005 Francesco P. Lovergine

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at *http://www.gnu.org/licenses/fdl.txt*