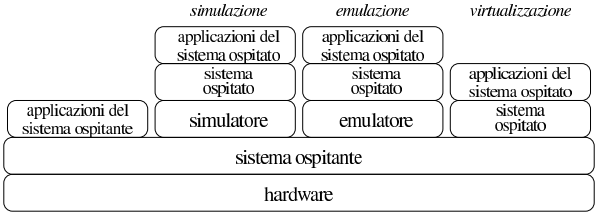
`kvm` 1393 `qemu` 1389

Quando si vuole eseguire un sistema operativo o comunque del software, in un contesto ambientale diverso da quello per cui è realizzato, si usano normalmente delle tecniche per riprodurre tale contesto. Queste tecniche prendono nomi differenti in base alla metodologia di adattamento predisposta, i cui confini non sono però ben delimitati. In linea di massima si possono distinguere tre filoni fondamentali: ***simulazione***, quando l'ambiente necessario al funzionamento del sistema ospitato non è abbinato alla realtà e rappresenta una pura finzione; ***emulazione***, quando il sistema ospitante riproduce l'ambiente necessario al sistema ospitato e l'attività svolta in tale ambiente produce effettivamente i risultati attesi; ***virtualizzazione***, quando il sistema ospitante è realizzato specificatamente per mettere a disposizione diversi ambienti, ad altrettanti sistemi ospitati, con un livello minimo di intermediazione tra tali ambienti e l'hardware sottostante.

31.1 Ospitalità

Quando ci si riferisce ai concetti di simulazione, emulazione e virtualizzazione, si distinguono due parti, rappresentate dal sistema ospitante (*host*) e dal sistema ospitato (*guest*). Purtroppo, in italiano il termine «ospite» si presta a rappresentare indifferentemente il ruolo di colui che ospita, o di colui che viene ospitato; pertanto si rende sempre necessario chiarire la cosa.

Figura 31.1. Sistema ospitante e sistema ospitato.



Tornando ai termini introdotti all'inizio del capitolo, un'ospitalità simulata è tale per cui il sistema ospitato non possa raggiungere in alcun modo la realtà dell'hardware; mentre un'ospitalità emulata consente di raggiungere l'hardware, ma attraverso la mediazione dell'emulatore e del sistema ospitante; infine, un'ospitalità virtualizzata consente al sistema ospitato di raggiungere l'hardware direttamente, nell'ambito dei confini stabiliti dal sistema ospitante. Come si vede dallo schema, tra il sistema ospitante e il sistema ospitato può interporsi un programma che ricostruisce l'ambiente necessario al funzionamento del sistema ospitato, ma quando si arriva alla virtualizzazione, di norma è il sistema ospitante che rende possibile tale astrazione e potrebbe non esserci alcuna intermediazione ulteriore.

31.2 Tipologia del contesto riprodotto

Il contesto di funzionamento riprodotto per l'utilizzo di un sistema ospitato, può essere fondamentalmente di due tipi: hardware o software. In altri termini, si può riprodurre il comportamento di un certo tipo di hardware, oppure di un sistema operativo completo. Pertanto, un contesto operativo che riproduca l'hardware x86 può servire per installarvi sopra un altro sistema operativo, mentre un contesto che riproduca il funzionamento di un certo sistema operativo, potrebbe servire per eseguire direttamente applicazioni di quel tale sistema. Per esempio, Bochs (*http://bochs.sourceforge.net/*) è un emulatore che riproduce un certo tipo di hardware e si utilizza per eseguire altri sistemi operativi, mentre WINE (*http://www.winehq.org/*) è un emulatore di Windows e consente di eseguire direttamente le applicazioni di tale sistema operativo nell'ambito di un sistema ospitante GNU/Linux.

Intuitivamente si comprende che, quando un certo contesto operativo riproduce direttamente il comportamento di un certo sistema operativo, può trattarsi solo di una simulazione o di un'emulazione, perché la virtualizzazione riguarda la riproduzione del comportamento dell'hardware reale sottostante.

31.3 File-immagine e partizioni

Lavorando con un sistema simulato o emulato, può essere necessario predisporre dei file-immagine che riproducono il contenuto completo di un disco, spesso suddiviso in partizioni. Il file-immagine più comune è quello «grezzo», o *raw*, che rappresenta una copia pura e semplice dei settori di un disco, dal primo all'ultimo, senza altre aggiunte e accorgimenti. Per creare un file del genere, in modo veloce, ci si può avvalere anche di '**dd**', come nell'esempio seguente:

```
# dd if=/dev/zero of=hd.img bs=1 count=0 seek=10G [Invio]
```

In questo modo, si crea il file '`hd.img`', il cui contenuto è nullo e viene allocato solo quando i settori vengono scritti effettivamente. Questo file può essere partizionato con gli strumenti consueti, confermando eventualmente il fatto che si intende operare su un file e non su un dispositivo vero e proprio:

```
# fdisk hd.img [Invio]
```

Al termine delle operazioni di partizionamento, il programma '**fdisk**' potrebbe mostrare la situazione seguente:

```
Disk hd.img: 1073 MB, 1073741824 bytes
255 heads, 63 sectors/track, 130 cylinders, total 2097152 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x51da06e3

 Device Boot   Start      End  Blocks  Id  System
hd.img1   *     2048  1845247  921600  83  Linux
hd.img4      1845248  2097151  125952  82  Linux swap / Solaris
```

D'altro canto, con l'aiuto del programma '**file**', si possono ottenere ugualmente queste informazioni:

```
# file hd.img [Invio]

hd.img: x86 boot sector; partition 1: ID=0x83, active, ↩
↪starthead 32, startsector 2048, ↩
↪1843200 sectors; partition 4: ID=0x82, ↩
↪starthead 219, startsector 1845248, ↩
↪251904 sectors, code offset 0x0
```

In un modo o nell'altro, si viene comunque a sapere qual è il primo settore della partizione che può essere interessante raggiungere. In un sistema GNU/Linux è facile innestare un file-immagine come se fosse un dispositivo di memorizzazione normale, ma non è altrettanto facile inizializzare e innestare una partizione, ovvero un file system che cominci a partire da un certo settore. Per prima cosa è necessario che il kernel sia predisposto per questo, attraverso l'abilitazione della voce *Device mapper support*:

- *Device drivers*
  - *Multi-device support (RAID and LVM)*
    - *Multiple devices driver support (RAID and LVM)*
      - *Device mapper support*

Se la funzione in questione è disponibile, ma solo come modulo, allora va caricato il modulo '**dm_mod**':

```
# modprobe dm_mod [Invio]
```

A questo punto, serve il programma Kpartx,[1] con il quale si generano dei file di dispositivo utili per accedere le partizioni esistenti:

```
# kpartx -v -a hd.img [Invio]

add map loop4p1 (253:3): 0 1843200 linear /dev/loop4 2048
add map loop4p4 (253:4): 0 251904 linear /dev/loop4 1845248
```

A questo punto, è possibile inizializzare le partizioni, perché si può fare riferimento a file di dispositivo:

```
# mkfs.ext3 /dev/mapper/loop4p1 [Invio]

# mkswap /dev/mapper/loop4p4 [Invio]
```

Il resto viene da sé, come in questo esempio dove si innesta la prima partizione in sola lettura:

```
# mount -o ro /dev/mapper/loop4p1 /mnt/test [Invio]
```

31.4 QEMU

QEMU[2] è un pacchetto di emulatori hardware di vari tipi di piattaforme, con la particolarità, nel caso l'hardware reale e quello emulato siano entrambi di tipo x86, di poterlo utilizzare direttamente, se il sistema operativo lo consente. In altri termini QEMU è un emulatore che in certi casi può sconfinare verso la virtualizzazione.

Il pacchetto che costituisce QEMU, una volta compilato per il sistema ospitante in cui deve essere utilizzato, dispone normalmente di file eseguibili differenti, a seconda della piattaforma che si va a emulare. Ma questi eseguibili sono divisibili in due gruppi: quelli che emulano l'hardware e si prestano per l'esecuzione di un sistema ospitato e quelli che invece si limitano a eseguire un programma, realizzato per lo stesso sistema operativo ospitante, ma per un'altra piattaforma hardware. Pertanto, i file eseguibili il cui nome corrisponde al modello seguente, sono adatti a emulare l'hardware, indipendentemente dal sistema operativo:

```
qemu-system-hardware
```

Al contrario, i nomi corrispondenti al modello seguente consentirebbero di utilizzare programmi per lo stesso sistema operativo, ma per un'altra piattaforma hardware:

```
qemu-hardware
```

In generale, il programma con il nome '**qemu**' dovrebbe corrispondere a quello che emula la stessa piattaforma hardware reale, a favore di un sistema ospitato completo.

31.4.1 Emulazione per gli applicativi

Il livello più semplice di emulazione offerto da QEMU riguarda la possibilità di eseguire programmi compilati per altre piattaforme, purché per lo stesso sistema operativo nel quale si sta agendo. In tal caso, oltre al programma è necessario disporre delle librerie dinamiche, di cui questo si avvale.

Per poter verificare questa cosa occorre procurarsi un programma molto semplice, il quale si avvalga esclusivamente delle librerie C standard. Per esempio si potrebbe recuperare una versione della shell Dash che utilizza solo la libreria '`libc.so.6`', mettendo il file eseguibile '`dash`' nella directory '`/tmp/x/bin/`' e i file del pacchetto che contiene la libreria '`libc.so.6`' nella directory '`/tmp/x/lib/`'. Supponendo che il file eseguibile e i file della libreria siano compilati per la piattaforma ARM, si potrebbe usare il comando seguente per avviare Dash:

```
$ qemu-arm -L /tmp/x/ /tmp/x/bin/dash [Invio]
```

L'opzione '**-L**' sta per *library* e indica la posizione di partenza per la ricerca delle librerie richieste dal programma. Così facendo, quando il programma cerca il file '`/lib/libc.so.6`', l'emulatore gli offre invece il file '`/tmp/x/lib/libc.so.6`'.

Se l'emulazione funziona, la shell avviata in questo modo si comporta come se fosse stata compilata per la piattaforma reale del proprio elaboratore.

31.4.2 Emulazione per un sistema completo

Per l'emulazione di un sistema completo, di norma è sufficiente disporre di un file-immagine che riproduca un disco avviabile: può essere un dischetto, un disco suddiviso in partizioni o un CD/DVD-ROM. Per esempio, per avviare un sistema contenuto nel file '`sistema.iso9660`', si può usare QEMU nel modo seguente:

```
$ qemu -cdrom sistema.iso9660 -boot d [Invio]
```

Il programma eseguibile '**qemu**' corrisponde a quello che emula la stessa architettura hardware di quella reale, altrimenti si possono usare i programmi '**qemu-system-*hardware***', se ciò che serve è qualcosa di diverso.

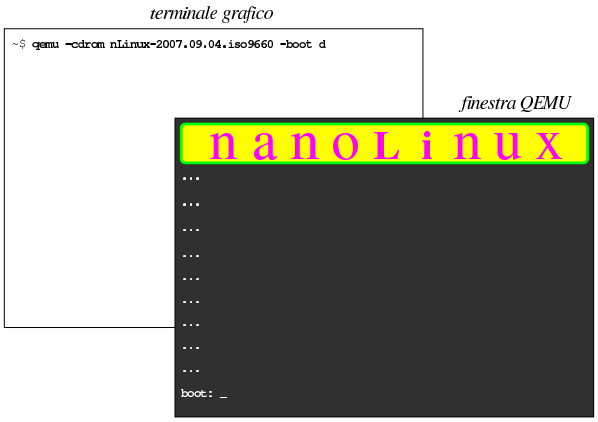
Per comprendere il senso dell'esempio mostrato e per poter fare altri tipi di esperimenti, occorre conoscere almeno un gruppo essenziale di opzioni, come si vede nella tabella successiva.

Tabella 31.5. Alcune opzioni per l'uso di QEMU, allo scopo di emulare una piattaforma hardware completa.

`-fda` ***file*** `-fdb` ***file***	Indica di usare un certo file da considerare come la prima o la seconda unità a dischetti. Il file può essere un file-immagine, oppure un file di dispositivo.
`-hda` ***file*** `-hdb` ***file*** `-hdc` ***file*** `-hdd` ***file***	Indica di usare un certo file da considerare come primo, secondo, terzo o quarto disco PATA. Il file può essere un file-immagine, oppure un file di dispositivo.
`-cdrom` ***file***	Indica di usare un certo file da considerare come unità ATAPI (CD-ROM o DVD-ROM).
`-boot a`	Richiede di avviare il sistema attraverso un dischetto (come stabilito con l'opzione '**-fda**').
`-boot c`	Richiede di avviare il sistema attraverso un disco PATA (come stabilito con l'opzione '**-hda**').
`-boot d`	Richiede di avviare il sistema attraverso un disco ATAPI (come stabilito con l'opzione '**-cdrom**').
`-m` ***n***	Richiede di offrire al sistema emulato ***n*** Mibyte. Se non si usa questa opzione, il valore predefinito è di 128 Mibyte.

Va osservato che per l'uso «normale» di QEMU, è necessario agire attraverso la grafica di X, in modo da consentire a QEMU anche l'emulazione della grafica per il sistema ospitato. Altrimenti, per un uso differente, è necessaria un po' di esperienza con QEMU, ma qui non viene mostrato alcun esempio diverso dai casi più semplici da realizzare. La figura successiva mostra l'avvio di QEMU attraverso un comando impartito con un terminale grafico, mentre il risultato visibile dell'emulazione appare in una finestra apposita.

Figura 31.6. Avvio di un sistema NLNX da un file-immagine che rappresenta un DVD-ROM.



31.4.3 Console di QEMU

L'incarnazione grafica di QEMU che esegue un sistema operativo, viene considerata dalla documentazione originale come la «console» di QEMU. A questo proposito si parla anche di «console virtuali», ma queste non vanno confuse con quelle del sistema GNU/Linux che probabilmente ha il ruolo di sistema ospitante. Quando la finestra della console di QEMU è attiva, valgono le combinazioni di tasti descritti nella tabella successiva.

Tabella 31.7. Combinazioni di tasti per il controllo della console di QEMU, funzionante in modalità grafica.

Combinazione di tasti	Descrizione
[*Ctrl Alt*]	Aggancia o sgancia la tastiera e il mouse dal sistema eseguito da QEMU.
[*Ctrl Alt f*]	Espande la finestra a occupare lo schermo intero, oppure riporta il funzionamento all'interno di una finestra.
[*Ctrl Alt 1*]	Seleziona la prima console virtuale di QEMU, corrispondente al sistema ospitato in esecuzione.
[*Ctrl Alt 2*]	Seleziona la seconda console virtuale di QEMU che dovrebbe corrispondere al «monitor», ovvero un terminale di controllo del funzionamento di QEMU.
[*Ctrl Alt 3*]	Seleziona la terza console virtuale di QEMU che dovrebbe corrispondere a un terminale associato alla porta seriale, ammesso che ciò sia stato previsto.
[*Ctrl Alt 4*]	Seleziona la quarta console virtuale di QEMU che dovrebbe corrispondere a un terminale associato alla porta parallela, ammesso che ciò sia stato previsto.

Nella console di QEMU, la cosa più importante è quello che viene chiamato «monitor», ovvero una console virtuale con una shell che consente di impartire dei comandi a QEMU, durante il suo funzionamento. La tabella successiva descrive alcuni di questi comandi.

Tabella 31.8. Alcuni comandi del monitor di QEMU.

Comando	Descrizione
`help` [***comando***] `?` [***comando***]	Mostra l'elenco dei comandi disponibili o una guida sintetica di quello indicato come argomento.
`quit` `q`	Conclude il funzionamento di QEMU.
`system_reset`	Riavvia il sistema ospitato controllato da QEMU.

Comando	Descrizione
`screendump` ***file***	Salva l'immagine della prima console virtuale di QEMU in un file in formato PPM. Il file indicato si intende collocato nel file system del sistema ospitante.
`stop` `resume`	Ferma o fa riprendere l'emulazione.
`sendkey` ***tasti***	Invia al sistema ospitato controllato dall'emulatore la combinazione di tasti indicata; per esempio, '**`sendkey ctrl-alt-f1`**' invia la combinazione di tasti [*Ctrl Alt F1*], la quale, se fosse premuta realmente, verrebbe intercettata dal sistema grafico, passando così alla prima console virtuale del sistema GNU/Linux ospitante.

31.4.4 Unità di memorizzazione e file-immagine

I file-immagine usati per rappresentare le unità di memorizzazione necessarie all'emulazione, possono avere formati differenti. Il formato più «semplice» è quello grezzo (viene individuato dalla parola chiave '**`raw`**') e può essere realizzato con gli strumenti comuni del sistema operativo ospitante. Altri formati possono essere utili per ridurre la dimensione occupata effettivamente nel file system ospitate, oppure per poter accedere a file-immagine di altri ambienti.

Per generare i formati particolari di file-immagine si utilizza il programma '**`qemu-img`**', ma qui non ne viene descritta la sintassi. A ogni modo non c'è nulla di complicato e si può consultare la pagina di manuale *qemu-img(1)*.

Oltre ai file-immagine, QEMU può accedere direttamente ai file di dispositivo, ammesso che sia avviato con i privilegi necessari. A questo proposito va osservato che con le opzioni '**`-hda`**', '**`-hdb`**',... '**`-hdd`**' si può fare riferimento a un file di dispositivo che rappresenta un disco intero, come '`/dev/hda`', '`/dev/hdb`',... mentre non è possibile indicare soltanto una partizione.

Per quanto riguarda l'accesso diretto ai file di dispositivo delle unità di memorizzazione, occorre considerare che questo non può avvenire in modo concorrente con il sistema operativo ospitante, perché si otterrebbe certamente un file system incoerente. In altri termini, nell'ambito dell'emulazione, se si accede a unità di memorizzazione a cui accede anche il sistema ospitante, è necessario limitarsi alla sola lettura.

È possibile offrire al sistema ospitato (sotto il controllato dell'emulatore) l'accesso a una directory del sistema operativo ospitante, mostrando questa directory come un'unità di memorizzazione avente un file system Dos-FAT. L'esempio seguente fa sì che il sistema ospitato veda la directory '`/tmp/mia/`' del sistema ospitante come se fosse la prima partizione del secondo disco PATA:

```
$ qemu -cdrom sistema.iso9660 -boot c -hdb fat:/tmp/mia [Invio]
```

È bene chiarire che l'opzione è '**`-hdb`**', ma in questo caso, si considera come se fosse la prima partizione.

Se si vuole fare la stessa cosa, ma mostrando che si tratta di un dischetto, occorre modificare leggermente il comando:

```
$ qemu -cdrom sistema.iso9660 -boot c ↩
↪      -fda fat:floppy:/tmp/mia [Invio]
```

31.5 L'acceleratore KQEMU

Quando il sistema ospitante è GNU/Linux, è possibile compilare un modulo che dovrebbe consentire di ottenere qualche miglioramento nella velocità di funzionamento del sistema ospitato tramite QEMU. Questo modulo è noto con il nome KQEMU, ma non fa parte dei sorgenti standard del kernel Linux.

Per procedere alla compilazione ci possono essere varie modalità, ma se si parte dai sorgenti originali del kernel Linux, occorre compilare un proprio kernel, installarlo assieme ai moduli relativi, quindi, senza cancellare i vari file-oggetto, si procede alla compilazione del modulo KQEMU. In questa sezione viene mostrato un esempio a tale proposito, ma va tenuto in considerazione che ci può essere un metodo più semplice se si segue l'organizzazione della propria distribuzione GNU/Linux

Si suppone di avere installato i sorgenti del kernel Linux nella directory '`/usr/src/linux-2.6.22.6/`' e di avere installato quelli del modulo KQEMU nella directory '`/usr/src/kqemu-1.3.0/`'. Si procede con la configurazione del kernel Linux, sapendo che non richiede accorgimenti particolari per quanto riguarda QEMU:

```
# cd /usr/src/linux-2.6.22.6 [Invio]

# make menuconfig [Invio]
```

La compilazione e l'installazione non viene descritta, ma si tratta solo di seguire il procedimento che si usa normalmente, con l'accortezza di non eliminare i file-oggetto generati dalla compilazione. Al termine si passa a configurare e compilare il modulo KQEMU:

```
# cd /usr/src/kqemu-1.3.0 [Invio]

# ./configure --kernel-path=/usr/src/linux-2.6.22.6 [Invio]

# make [Invio]
```

Se la compilazione si conclude con successo, si trova il file '`kqemu.ko`' nella directory '`/usr/src/kqemu-1.3.0/`'; questo file va copiato nella directory '`/lib/modules/2.6.22.6/misc/`':

```
# mkdir /lib/modules/2.6.22.6/misc [Invio]

# cp kqemu.ko /lib/modules/2.6.22.6/misc [Invio]
```

Una volta riavviato il sistema GNU/Linux ospitante, per far sì che il kernel in funzione sia quello appena compilato, si può caricare manualmente il modulo '`kqemu.ko`':

```
# insmod /lib/modules/2.6.22.6/misc/kqemu.ko [Invio]
```

Ma logicamente conviene predisporre uno script e organizzare la procedura di avvio in modo che il modulo venga caricato automaticamente a ogni riavvio.

31.6 KVM

KVM è una funzionalità del kernel Linux per i microprocessori x86 con le estensioni necessarie alla virtualizzazione. Il programma usato per sfruttare la virtualizzazione attraverso tali funzionalità del kernel Linux, si usa sostanzialmente nello stesso modo di QEMU e, in mancanza delle estensioni per la virtualizzazione, si avvale di QEMU stesso. Il codice di KVM fa parte dei sorgenti del kernel standard.

- *Device Drivers*
  - *Virtualization*
    - <M> *Kernel-based Virtual Machine*
      - <M> *KVM for Intel processor support*
      - <M> *KVM for AMD processor support*

KVM è utile quando il microprocessore dispone di estensioni adatte alla virtualizzazione. Per verificare se queste sono disponibili basta controllare nel file '`/proc/cpuinfo`', dove deve apparire la sigla '**`vmx`**' oppure '**`svm`**', tra gli indicatori (*flag*).

A fianco dei moduli che vengono prodotti dalla compilazione del kernel Linux si usa un programma per avviare l'esecuzione del sistema ospitato. Questo programma dovrebbe essere '**`kvm`**':

```
kvm opzioni
```

La sintassi per l'uso del programma '**`kvm`**' è sostanzialmente la stessa di QEMU, quando riguarda l'emulazione per ospitare un sistema completo. A questo proposito va osservato che, in mancanza

delle estensioni hardware per la virtualizzazione, ‘**kvm**’ si avvale di ‘**qemu**’, anche se in tal caso l’emulazione è molto più lenta rispetto a quanto si otterrebbe con la virtualizzazione assistita dalle estensioni hardware.

31.7 Installazione di una distribuzione Debian ARM con QEMU

Tra tutte le piattaforme esistenti, dopo l’architettura x86 che domina il mercato, è interessante ARM (*Acorn RISC machines*, ovvero *Advanced RISC machines*). Come suggerisce il nome, si tratta di un’architettura RISC (*Reduced instruction set computer*) usata prevalentemente su componenti di piccole dimensioni, inclusi i telefoni e i sistemi a tavoletta.

Se si vuole prendere confidenza con l’architettura ARM può essere utile l’emulatore QEMU, attraverso il quale, per esempio, è possibile installare una distribuzione Debian in un file-immagine. L’installazione che si propone qui comporta l’emulazione di un dispositivo «ARM Versatile Platform Baseboard» (Versatile/PB) con CPU ARM926EJ-S.

Per cominciare si deve predisporre una directory e un file-immagine, per esempio di 10 Gibyte. Il file-immagine potrebbe essere creato utilizzando un formato speciale di QEMU, ma supponendo di operare in un sistema GNU/Linux è forse meglio limitarsi al formato grezzo, il quale potrebbe così essere innestato successivamente:

```
# mkdir /opt/qemu/arm [Invio]

# cd /opt/qemu/arm [Invio]

# dd if=/dev/zero of=hd0.img bs=1 count=0 skip=10G [Invio]
```

Il file viene creato immediatamente, perché in pratica, nel modo mostrato, non viene allocato ancora nulla di concreto. A questo punto è necessario disporre del kernel e del disco-RAM iniziale per la procedura di installazione Debian. Supponendo di fare riferimento alla versione «squeeze», questi file si trovano presso *http://ftp.de.debian.org/debian/dists/squeeze/main/installer-armel/current/images/versatile/netboot/vmlinuz-2.6.32-5-versatile* e *http://ftp.de.debian.org/debian/dists/squeeze/main/installer-armel/current/images/versatile/netboot/initrd.gz* . Per non fare confusione, questi possono essere collocati opportunamente in una sottodirectory adatta:

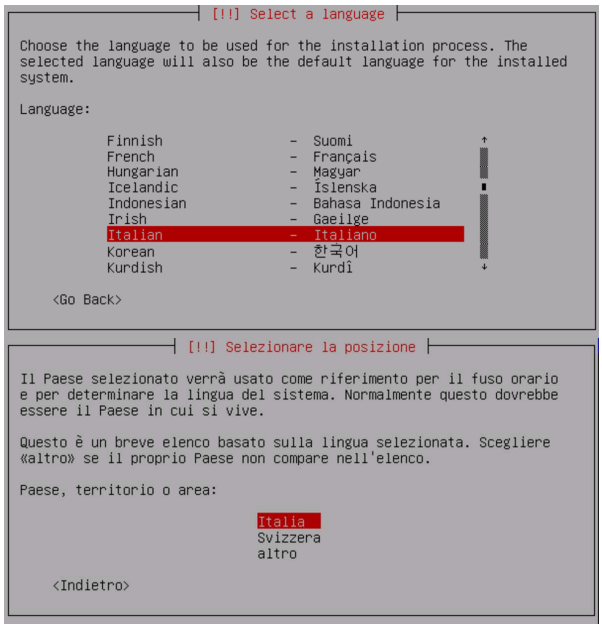
```
# mkdir inst [Invio]

# mv vmlinuz* initrd* inst [Invio]
```

Quindi conviene preparare un piccolo script per avviare QEMU, con il kernel e il disco-RAM adatto all’installazione della distribuzione Debian:

```
#!/bin/sh
qemu-system-arm -M versatilepb \
                -kernel inst/vmlinuz-2.6.32-5-versatile \
                -initrd inst/initrd.gz \
                -hda hd0.img \
                -append "root=/dev/ram"
```

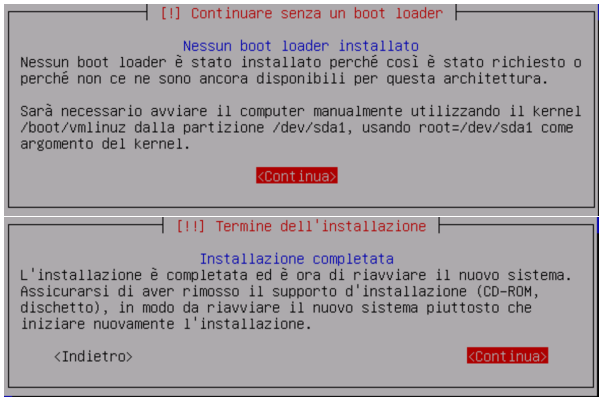
Con questo script si avvia l’emulatore, partendo con l’installazione della distribuzione.

Figura 31.10. Prima fase dell’installazione della distribuzione Debian per architettura ARM.



La procedura di installazione procede, anche con la definizione delle partizioni e la loro inizializzazione (si tratta del file-immagine preparato in precedenza), quindi inizia l’installazione (partendo dal presupposto che il sistema ospitante sia effettivamente connesso alla rete esterna). Alla fine della procedura di installazione, viene fatto osservare che, per l’architettura ARM, non è disponibile un sistema di avvio, per cui occorre provvedere per conto proprio ad avviare il sistema.

Figura 31.11. Fasi finali dell’installazione.



Al termine, il sistema si arresta (e QEMU si ferma). Ma per riavviare il sistema ARM occorre utilizzare il kernel e il disco-RAM adatto. Questi file si troverebbero nel file system contenuto nel file-immagine, precisamente nella directory ‘`/boot/`’. Per estrarre questi file ci si può aiutare con Kpartx, come mostrato all’inizio del capitolo. Una volta ottenuti questi file, vanno collocati opportunamente e va realizzato un altro script per l’avvio del sistema installato:

```
# mkdir boot [Invio]

# mv vmlinuz* initrd* boot [Invio]
```

```
#!/bin/sh
qemu-system-arm -M versatilepb \
                -kernel boot/vmlinuz-2.6.32-5-versatile \
                -initrd boot/initrd.img-2.6.32-5-versatile \
                -hda hd0.img \
                -append "root=/dev/sda1"
```

Va osservato che anche l’opzione ‘**root=**’ è diversa.

31.8 Riferimenti

«

- Fabrice Bellard, *QEMU* *http://wiki.qemu.org/Main_Page*
- *Kernel Based Virtual Machine* *http://www.linux-kvm.org/page/Main_Page*
- *RealView Platform Baseboard for ARM926EJ-S, HBI-0117, User Guide* *http://infocenter.arm.com/help/topic/com.arm.doc.dui0224i/DUI0224I_realview_platform_baseboard_for_arm926ej26_s_ug.pdf*
- *ARM926EJ-S Development Chip, Reference Manual* *http://infocenter.arm.com/help/topic/com.arm.doc.ddi0287b/DDI0287B_arm926ej26s_dev_chip_technical_reference_manual.pdf*

[1] **Kpartx** GNU GPL

[2] **QEMU** GNU GPL