ELKS: realizzazione personale

ELKS e i programmi di servizio del sistema sono distribuiti sia in forma sorgente, sia in dischetti già pronti per l'uso e l'installazione. Tuttavia, questi dischetti non funzionano bene con tutti i tipi elaboratore, per quanto questi possano essere vecchi, o anche antichi, pertanto conviene prepararsi alla realizzazione in proprio dei dischetti, in modo da poter raggirare eventuali ostacoli imprevisti.

In generale, per tutti i lavori necessari a preparare il proprio sistema ELKS, occorre utilizzare un elaboratore funzionante con un sistema GNU/Linux.

File system e dischetti

Salvo l'uso di estensioni particolari, ELKS è in grado di accedere soltanto a file system Minix con i nomi della lunghezza massima di 14 byte.

Quando si usa un sistema GNU/Linux per inizializzare i dischetti o i file-immagine dei dischetti da usare con ELKS, bisogna tenere presente che il programma '**`mkfs.minix`**' richiede l'uso di un'opzione appropriata, altrimenti genera un file system Minix incompatibile. Nell'esempio seguente si inizializza il dischetto corrispondente al file di dispositivo '`/dev/fd0`':

```
# mkfs.minix -n 14 /dev/fd0 [Invio]
```

File di dispositivo

«

I file di dispositivo usati da ELKS sono simili, ma non uguali a quelli di un sistema GNU/Linux comune. La situazione più importante da osservare riguarda i file di dispositivo per l'accesso ai dischetti e ai dischi fissi. Una problema abbastanza comune riguarda l'uso di '**`rdev`**' per modificare un kernel già compilato in modo che avvii un disco diverso da quello previsto in fase di compilazione: '**`rdev`**' può essere usato anche da un sistema GNU/Linux, ma i numeri primario e secondario dei file di dispositivo non corrispondono.

Tabella u178.1. File di dispositivo principali di un sistema ELKS.

Nome	Tipo	Numero primario	Numero secondario	Sigla in esadecimale
'`/dev/mem`'	caratteri	1	1	0101_{16}
'`/dev/kmem`'	caratteri	1	2	0102_{16}
'`/dev/null`'	caratteri	1	3	0103_{16}
'`/dev/zero`'	caratteri	1	5	0105_{16}
'`/dev/full`'	caratteri	1	7	0107_{16}

Nome	Tipo	Numero primario	Numero secondario	Sigla in esadecimale
‘/dev/bda’	blocchi	3	0	0300_{16}
‘/dev/bda1’	blocchi	3	1	0301_{16}
‘/dev/bda2’	blocchi	3	2	0302_{16}
‘/dev/bda3’	blocchi	3	3	0303_{16}
‘/dev/bda4’	blocchi	3	4	0304_{16}
‘/dev/bdb’	blocchi	3	64	0340_{16}
‘/dev/bdb1’	blocchi	3	65	0341_{16}
‘/dev/bdb2’	blocchi	3	66	0342_{16}
‘/dev/bdb3’	blocchi	3	67	0343_{16}
‘/dev/bdb4’	blocchi	3	68	0344_{16}
‘/dev/fd0’	blocchi	3	128	0380_{16}
‘/dev/fd1’	blocchi	3	192	$03C0_{16}$
‘/dev/tty1’	caratteri	4	0	0400_{16}
‘/dev/tty2’	caratteri	4	1	0401_{16}
‘/dev/tty3’	caratteri	4	2	0402_{16}
‘/dev/ttyS0’	caratteri	4	64	0440_{16}
‘/dev/ttyS1’	caratteri	4	65	0441_{16}

Nome	Tipo	Numero primario	Numero secondario	Sigla in esadecimale
‘/dev/ttyS2’	caratteri	4	66	0442_{16}
‘/dev/lp0’	caratteri	6	0	0600_{16}
‘/dev/lp1’	caratteri	6	1	0601_{16}
‘/dev/lp2’	caratteri	6	2	0602_{16}
‘/dev/tcpdev’	caratteri	8	0	0800_{16}

I file di dispositivo ‘`/dev/bd*`’ si riferiscono all’accesso al disco fisso, attraverso il BIOS. Eventualmente, sono previsti file di dispositivo per l’accesso diretto al disco fisso, con la denominazione consueta ‘`/dev/hd*`’, anche se inizialmente il kernel non è in grado di farlo, ma ugualmente il numero primario differisce da quello usato con i sistemi GNU/Linux.

Per la ricostruzione dei file di dispositivo è disponibile uno script ‘**`MAKEDEV`**’ già pronto, con descrizioni molto chiare, ma forse conviene riepilogare i comandi per i file di dispositivo elencati nella tabella:

```
#!/bin/sh

mknod mem       c 1 1
mknod kmem      c 1 2
mknod null      c 1 3

mknod zero      c 1 5
mknod full      c 1 7

mknod bda       b 3 0
mknod bda1      b 3 1
mknod bda2      b 3 2
mknod bda3      b 3 3
mknod bda4      b 3 4

mknod bdb       b 3 64
mknod bdb1      b 3 65
mknod bdb2      b 3 66
mknod bdb3      b 3 67
mknod bdb4      b 3 68

mknod fd0       b 3 128
mknod fd1       b 3 192

mknod tty1      c 4 0
mknod tty2      c 4 1
mknod tty3      c 4 2

mknod ttyS0     c 4 64
mknod ttyS1     c 4 65
mknod ttyS2     c 4 66

mknod lp0       c 6 0
mknod lp1       c 6 1
mknod lp2       c 6 2

mknod tcpdev    c 8 0
```

«

Un problema molto importante da risolvere quando si vuole mettere insieme un sistema operativo è il meccanismo di avvio. Il kernel ELKS, così come avviene nel caso di Linux, si avvia da solo, copiandolo in un dischetto come se fosse l'immagine dello stesso, offrendo così una base di partenza sicura.

Il fatto di avviare il kernel ELKS direttamente implica l'impossibilità di passargli dei parametri, pertanto bisogna utilizzare '**rdev**' o un programma simile per definire da quale file di dispositivo deve essere innestato il file system principale:

rdev ***file_kernel*** ***file_di_dispositivo*** [*Invio*]

Il programma '**rdev**' è disponibile sia in un sistema GNU/Linux comune, sia in un sistema ELKS; quello che cambia, purtroppo, sono i numeri primario e secondario dei file di dispositivo. Per esempio, se si sta operando attraverso un sistema GNU/Linux e si vuole impostare il file '`elks`', che si intende essere un kernel ELKS, occorre prima preparare il file di dispositivo appropriato, anche se questo lo si può fare in una directory temporanea:

mknod /tmp/fd0 b 3 128 [*Invio*]

rdev elks /tmp/fd0 [*Invio*]

rdev elks [*Invio*]

```
Root device 0x0380
```

Come si può intuire, il numero 0380_{16} rappresenta un file di dispositivo con numero primario pari a 03_{16}, ovvero 3_{10}, e numero secondario pari a 80_{16}, ovvero 128_{10}. Se invece si fa riferimento a un numero pri-

mario e secondario che in qualche modo sono previsti nel sistema, si può ottenere un'informazione un po' confusa:

```
# mknod /tmp/bda1 b 3 1 [Invio]

# rdev elks /tmp/bda1 [Invio]

# rdev elks [Invio]

  Root device /dev/hda1
```

Evidentemente occorre fare attenzione per non confondersi.

Bootblocks

Il pacchetto Bootblocks [1] consente di avviare un sistema ELKS contenuto in un dischetto o in una partizione del disco fisso, con il kernel inserito nello stesso file system. Il pacchetto viene distribuito assieme agli strumenti di sviluppo Dev86, ma non viene compilato automaticamente assieme a quelli. Si trova precisamente nella sottodirectory '`bootblocks/`' dei sorgenti di Dev86. Si compila in modo molto semplice con il comando '**`make`**':

```
# cd sorgenti_dev86/bootblocks [Invio]

# make [Invio]
```

Dalla compilazione si ottengono diversi file; per quanto riguarda il problema dell'avvio di ELKS sono utili:

File	Descrizione
'**`makeboot`**'	programma per l'installazione del settore di avvio, da usare attraverso un sistema GNU/Linux comune;
'**`makeboot.com`**'	programma analogo a '**`makeboot`**', da usare con un sistema Dos;

File	Descrizione
‘**minix_elks.bin**’	programma di appoggio per l’avvio di un kernel ELKS.

Per fare in modo che un dischetto, con file system Minix, contenente un sistema ELKS, completo di kernel e di programmi di servizio, occorre predisporre alcune cose come se le aspetta il sistema di avvio. Per la precisione si deve preparare la directory ‘`/boot/`’ contenente il kernel e il file ‘`minix_elks.bin`’, ma entrambi devono avere un nome appropriato; pertanto, il kernel deve essere ‘`/boot/linux`’ e il programma di appoggio (in origine ‘`minix_elks.bin`’) deve chiamarsi ‘`/boot/boot`’.

Supponendo di utilizzare un sistema GNU/Linux, supponendo di avere preparato il dischetto Minix (con i nomi al massimo di 14 byte) contenente tutto quello che serve, soprattutto con la directory ‘`/boot/`’ come spiegato, se questo dischetto risulta inserito nell’unità corrispondente al file di dispositivo ‘`/dev/fd0`’, **senza essere stato innestato**, si può eseguire il comando seguente, tenendo conto che il programma ‘**makeboot**’ si presume collocato in una directory prevista tra i vari percorsi della variabile di ambiente ‘**PATH**’:

```
# makeboot minix /dev/fd0 [Invio]

  Wrote sector 0
  Wrote sector 1
```

Se il programma si accorge che il settore di avvio del dischetto contiene già qualcosa, si rifiuta di procedere, a meno di usare l’opzione ‘**-f**’:

```
# makeboot -f minix /dev/fd0 [Invio]
```

```
Boot block isn't empty, zap it first
Wrote sector 0
Wrote sector 1
```

Riquadro u178.8. Come cambiare il nome del file del kernel.

Il sistema di avvio che si ottiene con l'uso di '**makeboot**' e del file '`minix_elks.bin`' prevede che il kernel sia precisamente il file '`/boot/linux`', mentre forse sarebbe più appropriato '`/boot/elks`'. Per ovviare a questa piccola incoerenza, basta intervenire nel sorgente, precisamente nel file '***sorgenti_dev86***`/bootblocks/minix_elks.c`'. A un certo punto, questo file contiene le righe seguenti:

```
elks_name:
  .asciz        "linux"
  .byte         0,0,0,0,0,0,0,0,0
```

Per fare in modo di avviare il kernel contenuto nel file '`/boot/elks`', si deve modificare e ricompilare:

```
elks_name:
  .asciz        "elks"
  .byte         0,0,0,0,0,0,0,0,0
```

Per installare ELKS in una partizione del disco fisso, le cose si complicano, perché manca la possibilità di usare il programma '**makeboot**' con il sistema ELKS stesso e purtroppo, il programma '`makeboot.com`', funzionante con un sistema Dos, si rifiuta di intervenire in partizioni che non siano Dos. Pertanto, occorre preparare una porzione di codice da incollare poi con il programma '**dd**' o simile:

```
# touch avvio[Invio]

# makeboot minix avvio[Invio]
```

```
Cannot read sector 0, clearing
Cannot read sector 1, clearing
Wrote sector 0
Wrote sector 1
```

In pratica, si crea un file vuoto, in questo caso il file ‘`avvio`’, quindi gli si inserisce il codice necessario, da incollare successivamente all’inizio della partizione da avviare.

Supponendo di avere inserito questo file in un dischetto contenente un sistema ELKS funzionante, con il quale si è riusciti a predisporre una partizione del disco fisso allo scopo di alloggiare il sistema stesso, basta copiare il file in questo modo:

```
# dd if=avvio of=/dev/bda1 [Invio]
```

Naturalmente, in questo caso si sta facendo riferimento alla prima partizione.

Bisogna ricordare che il file del kernel deve essere modificato con ‘**`rdev`**’, in modo da utilizzare il file system contenuto nella partizione stessa.

Perché il codice iniziale della partizione venga messo in funzione, è necessario che il primo settore del disco fisso (MBR) indichi la partizione stessa come avviabile.

Nonostante tutto, può succedere che il codice inserito all'inizio della partizione venga rifiutato dal BIOS, per mancanza della firma $55AA_{16}$. In presenza di difficoltà di questo tipo, rimane la possibilità di avviare a partire da un dischetto con il kernel ritoccato attraverso '**rdev**' in modo da utilizzare la partizione corretta.

Bootkit

Bootkit [2] è un sistema di avvio abbastanza buono, che in particolare può passare dei parametri di avvio al kernel ELKS.

Il problema di Bootkit è che non si trovano i sorgenti.

Si può trovare Bootkit incorporato nella distribuzione EDE (*ELKS distribution edition*) e si utilizza all'interno del sistema ELKS stesso.

```
bootkit opzioni file
```

Per ottenere il risultato, occorre predisporre la directory '`/boot/`' nel file system Minix che contiene il sistema ELKS da avviare. All'interno di questa directory si colloca il file '`boot.conf`', con una serie di direttive e di solito il file '`boot.txt`', il cui contenuto deve essere mostrato in fase di avvio. A titolo di esempio, il file '`boot.conf`' potrebbe contenere le righe seguenti:

```
message=boot.txt
prompt
timeout 100
image=elks
    label=fd0
    root=0x380
image=elks
    label=fd1
    root=0x381
image=elks
    label=bda1
    root=0x301
image=elks
    label=bda2
    root=0x302
image=elks
    label=bda3
    root=0x303
image=elks
    label=bda4
    root=0x304
#other=/dev/bda1
#    label=dos
```

Inizialmente, la direttiva '**`message=boot.txt`**' dichiara di usare il file '`boot.txt`' per mostrare un messaggio all'avvio, quindi, la direttiva '**`prompt`**' fa sì che venga mostrato un invito, con il quale poter scegliere tra le diverse possibilità di avvio. La direttiva '**`timeout 100`**' fa sì che la prima delle varie modalità di avvio sia scelta in mancanza di una risposta all'avvio, entro 10 s.

Le varie direttive '**`image=elks`**' delimitano l'inizio di una sezione, distinguibile in base al valore assegnato alla direttiva '**`label`**', che dichiara l'uso del kernel contenuto nel file '`elks`', che si deve trovare nella stessa directory ('`/boot/`'). Come si vede, all'interno di queste sezioni, la direttiva '**`root`**' consente di specificare il file system che il kernel deve innestare, anche se ciò deve essere fatto indicando i numeri primario e secondario del file di dispositivo, in forma

esadecimale.

Il file ‘`boot.txt`’ potrebbe contenere il messaggio seguente:

```
Please select one of the following root file systems:

fd0     (default)
fd1
bda1
bda2
bda3
bda4
```

Si completa il lavoro mettendo nella directory ‘`/boot/`’ il file del kernel, che in base agli esempi deve chiamarsi ‘`elks`’, quindi occorre creare il file ‘`boot`’, attraverso ‘`bootkit`’:

```
# bootkit -i boot /boot/boot [Invio]
```

Si osservi che questo comando va dato all’interno di un sistema ELKS già funzionante, ma una volta creato, il file può essere copiato così in altri dischetti.

Rimane il problema della creazione di un settore di avvio, che poi cerca il programma ‘`/boot/boot`’, il quale poi legge la configurazione del file ‘`/boot/boot.conf`’:

```
# bootkit -i minix -b copia_settore /dev/fd0 [Invio]
```

In questo modo si installano 1024 byte di codice all’inizio del dischetto (corrispondente al file di dispositivo ‘`/dev/fd0`’) e si salva una copia di quanto era presente prima nel file indicato come argomento dell’opzione ‘**`-b`**’. L’opzione ‘**`-b`**’ è obbligatoria in questo caso, quindi, se non si vuole salvare ciò che viene sovrascritto, si può usare ‘`/dev/null`’:

```
# bootkit -i minix -b /dev/null /dev/fd0 [Invio]
```

Teoricamente tutto questo potrebbe funzionare anche con una partizione di un disco fisso, per esempio la prima secondo il comando seguente:

```
# bootkit -i minix -b /dev/null /dev/bda1 [Invio]
```

Per funzionare, è comunque necessario che il settore iniziale del disco fisso (MBR) indichi la partizione come avviabile.

Anche Bootkit potrebbe fallire nel compito di avviare il sistema dal disco fisso, ma rimane il fatto che da dischetto è un meccanismo ottimo, che non richiede l'uso di '**rdev**' per dire al kernel quale disco o partizione utilizzare.

Installazione manuale nel disco fisso

«

L'installazione di un sistema ELKS nel disco fisso richiede di poter disporre di un dischetto funzionante e relativamente completo, possibilmente con una shell efficiente; inoltre, prima di iniziare occorre avere un quadro abbastanza chiaro di come questo dischetto è organizzato, cosa che si può fare utilizzando un sistema GNU/Linux, attraverso tutti gli strumenti a cui si è abituati.

Quando è tutto pronto, con dischetto di ELKS, o anche con un altro sistema operativo, se ciò è possibile e preferibile, occorre intervenire nella suddivisione delle partizioni del disco fisso. Considerato che l'accesso al disco avviene attraverso il BIOS, è bene che la partizione sia piccola e si trovi all'inizio o vicino all'inizio del disco stesso.

Teoricamente, dato il fatto che l'accesso avviene tramite le funzioni del BIOS, la partizione può arrivare a un massimo di 32 Mibyte, ma

in pratica, può darsi che si debba ridurre ancora di più. Comunque, la partizione può anche essere relativamente grande, ma poi, quando la si va a inizializzare, bisogna creare un file system piccolo.

La partizione deve essere di tipo *old minix*, corrispondente al codice 80_{16}.

Il sistema ELKS dovrebbe disporre del programma '**fdisk**' per modificare l'organizzazione delle partizioni:

```
# fdisk /dev/bda [Invio]
```

Il suo utilizzo è simile a quello dello stesso programma usato nei sistemi GNU/Linux (in particolare il comando [*?*] richiama la guida degli altri comandi disponibili), ma conviene armarsi di calcolatrice per fare i conti di quanti cilindri servono per la partizioni che si vogliono creare.

Ammesso che il settore di avvio del disco fisso contenga del codice corretto, è bene ricordare di rendere avviabile la partizione che si va a creare per ospitare ELKS.

Una volta sistemate le partizioni e salvate con il comando [*w*], si può abbandonare il programma '**fdisk**' ([*q*]) per passare a '**mkfs**' con il quale creare il file system Minix:

```
# mkfs /dev/bda1 5000 [Invio]
```

In questo caso si suppone di dover inizializzare la prima partizione del primo disco fisso, con un file system di 5000 Kibyte.

Si osservi che la dimensione in settori fornita da '**fdisk**' rappresenta una quantità espressa in unità da 512 byte, mentre il valore che si fornisce al programma '**mkfs**' esprime una quantità in unità da 1024 byte. In pratica, il file system va creato con un valore che non può eccedere la metà di quanto riporta '**fdisk**'.

È ormai chiaro che la partizione può essere più grande del file system Minix che si va a creare. Una volta creato, è bene provare a innestarlo, per esempio con il comando seguente ammettendo che sia disponibile la directory '`/mnt/`':

```
# mount /dev/bda1 /mnt [Invio]
```

Se si ottiene una segnalazione di errore nella quale viene affermato che il file system non è valido (non è Minix), bisogna provare a inizializzarlo con una dimensione minore.

Quando si arriva al punto di essere stati capaci di innestare il file system creato nella partizione del disco fisso, si può procedere con la copia del contenuto del dischetto, ma con prudenza. Infatti, il comando '**cp**' potrebbe essere molto poco amichevole, inoltre, fornendogli troppi argomenti (quando si usano dei modelli la shell li espande in elenchi che possono essere anche abbastanza numerosi) può bloccarsi, assieme a tutto il sistema.

Comunque, con un po' di prudenza, si possono ricreare le directory e al loro interno vi si possono copiare i file che si trovano nel dischetto (il programma '**cp**' potrebbe essere capace di copiare soltanto file normali); quindi occorre riprodurre una directory '`/dev/`' con i file di dispositivo necessari e infine si può cercare di risolvere il problema dell'avvio.

Come si può comprendere molto presto, quando ci si cimenta in un lavoro di questo tipo, occorre una strategia, che può consistere nella preparazione preventiva di qualche script che faccia buona parte di questo lavoro in modo automatico.

In queste spiegazioni ci sono molte piccole cose che sono omesse, soprattutto perché le situazioni che si presentano cambiano facilmente con una grande quantità di sfumature. È evidente che si tratta di un lavoro che può affrontare solo chi ha già una buona padronanza di un sistema GNU/Linux e non si lascia scoraggiare dai piccoli fallimenti a cui si va incontro sicuramente.

Adattamento della mappa della tastiera

Un problema che può presentarsi è quello di adattare la mappa della tastiera, cosa che richiede la ricompilazione del kernel selezionando il tipo corretto per ciò che si deve usare.

Se la mappa che serve non c'è, oppure se non funziona come ci si aspetterebbe, occorre predisporre in proprio un file con un nome che corrisponda al modello '`keys-`***xy***`.h`', dove ***xy*** sono due lettere che identificano la nazionalità. Questo file, insieme agli altri delle altre mappe, va collocato nella directory '***sorgenti_elks***`/arch/i86/drivers/char/KeyMaps/`'. Naturalmente, il file deve avere una certa forma, che si può intuire osservando quelli già esistenti.

Bisogna tenere in considerazione il fatto che il sistema è predisposto per un elaboratore con tastiera «XT», pertanto non può esistere il tasto [*AltGr*] e nemmeno si possono attuare tutte quelle combinazioni che invece sono disponibili con un sistema GNU/Linux comune.

Probabilmente si può usare con successo solo la parte alfanumerica, i tasti freccia, [*Esc*] e probabilmente i tasti funzionali. Quando si inserisce il [*Fissamaiuscole*] non è detto che la spia corrispondente si accenda; il tasto [*BlocNum*] non funziona e probabilmente i numeri sulla tastiera numerica si ottengono inserendo il [*Fissamaiuscole*].

Dovendo predisporre o modificare una mappa che comprende anche lettere accentate e altri simboli speciali che sono al di fuori del codice ASCII tradizionale, occorre considerare che la codifica usata è quella originale degli elaboratori 8086, ovvero quella che era nota come CP 437 (si veda la sezione 47.7.7 per trovare una copia completa della codifica CP 437).

Viene proposto un esempio di mappa per la tastiera italiana, che deve corrispondere al file '***sorgenti_elks***`/arch/i86/drivers/char/KeyMaps/keys-it.h`'. Si osservi in particolare l'intestazione che serve a uno script per fare in modo che il file venga preso in considerazione durante la configurazione del kernel.

Listato u178.14. Esempio di un file per la configurazione della tastiera secondo la disposizione italiana.

```
/* Keymap:IT:Italiano:Italy     */

#ifndef __KEYMAP_IT__
#define __KEYMAP_IT__

#if defined(CONFIG_KEYMAP_IT)

/*
    \ 1 2 3 4 5 6 7 8 9 0 ' ì
       q w e r t y u i o p è +
        a s d f g h j k l ò à ù
         z x c v b n m , . -
*/

static unsigned char xtkb_scan[] = {
```

```
  0,
  033,
  '1',  '2',  '3',  '4',  '5',  '6',  '7',  '8',  '9',  '0', '\'', 0215, '\b',
 '\t',  'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  'o',  'p', 0212,  '+',
  015,
 0202,  'a',  's',  'd',  'f',  'g',  'h',  'j',  'k',  'l', 0225, 0205,
 '\\',
 0200,
 0227,
  'z',  'x',  'c',  'v',  'b',  'n',  'm',  ',',  '.',  '-',
 0201,
  '*', 0203,  ' ', 0204,
 0241, 0242, 0243, 0244, 0245, 0246, 0247, 0250, 0251, 0252,
 0205,  '?',
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

/*
    | ! " £ $ % & / ( ) = ? ^
       Q W E R T Y U I O P é *
        A S D F G H J K L ç ° §
         Z X C V B N M ; : _
*/

static unsigned char xtkb_scan_shifted[] = {
    0,
  033,
  '!',  '"', 0234,  '$',  '%',  '&',  '/',  '(',  ')',  '=',  '?',  '^', '\b',
 '\t',  'Q',  'W',  'E',  'R',  'T',  'Y',  'U',  'I',  'O',  'P', 0202,  '*',
 '\r',
 0202,  'A',  'S',  'D',  'F',  'G',  'H',  'J',  'K',  'L', 0207, 0370,
  '|',
 0200,
  025,
  'Z',  'X',  'C',  'V',  'B',  'N',  'M',  ';',  ':',  '_',
 0201,
  '*', 0203,  ' ', 0204,
 0221, 0222, 0223, 0224, 0225, 0226, 0227, 0230, 0231, 0232,
 0204, 0213,
```

```
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

/*
    \ < > 3 4 5 6 { [ ] } ` ~
       q w e r t y u i o p [ ]
        a s d f g h j k l @ # ù
         « » c v b n m , . -
*/

static unsigned char xtkb_scan_ctrl_alt[] = {
    0,
  033,
  '<',  '>',  '3',  '4',  '5',  '6',  '{',  '[',  ']',  '}',  '`',  '~', '\b',
 '\t',  'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  'o',  'p',  '[',  ']',
 '\r',
 0202,  'a',  's',  'd',  'f',  'g',  'h',  'j',  'k',  'l',  '@',  '#',
 '\\',
 0200,
 0227,
 0256, 0257,  'c',  'v',  'b',  'n',  'm',  ',',  '.',  '-',
 0201,
  '*', 0203,  ' ', 0204,
 0241, 0242, 0243, 0244, 0245, 0246, 0247, 0250, 0251, 0252,
 0205,  '?',
  '7',  '8', '9',
  '-',
  '4',  '5', '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

/*
    \ 1 2 3 4 5 6 7 8 9 0 ' ì
       Q W E R T Y U I O P è +
        A S D F G H J K L ò à ù
         Z X C V B N M , . -
*/
```

```
static unsigned char xtkb_scan_caps[] = {
    0,
  033,
  '1',  '2',  '3',  '4',  '5',  '6',  '7',  '8',  '9',  '0', '\'', 0215, '\b',
 '\t',  'Q',  'W',  'E',  'R',  'T',  'Y',  'U',  'I',  'O',  'P', 0212,  '+',
 '\r',
 0202,  'A',  'S',  'D',  'F',  'G',  'H',  'J',  'K',  'L', 0225, 0205,
 '\\',
 0200,
 0227,
  'Z',  'X',  'C',  'V',  'B',  'N',  'M',  ',',  '.',  '-',
 0201,
  '*', 0203,  ' ', 0204,
 0221, 0222, 0223, 0224, 0225, 0226, 0227, 0230, 0231, 0232,
 0204, 0213,
  '7',  '8',  '9',
  '-',
  '4',  '5',  '6',
  '+',
  '1',  '2',  '3',
  '0',  '.'
};

#endif

#endif
```

Come si può intuire dai nomi degli array, si distingue tra quattro situazioni: la disposizione normale, la disposizione che entra in gioco quando si preme il tasto delle maiuscole, quando si inserisce il [*Fissamaiuscole*], infine quando si preme la combinazione [*Ctrl Alt* ***x***]. In pratica, per ottenere simboli come la chiocciola e il cancelletto, occorre usare la combinazione [*Ctrl Alt ò*] e [*Ctrl Alt à*]. Per qualche motivo, non è stato possibile collocare correttamente i simboli [<] e [>], che si ottengono invece con [*Ctrl Alt 1*] e [*Ctrl Alt 2*] rispettivamente. Inoltre, sono disponibili anche le parentesi graffe, l'accento rovesciato, la tilde e le virgolette

basse uncinate, come si fa di solito in un sistema GNU/Linux, ma sempre usando una combinazione del tipo [*Ctrl Alt* ***x***].

Si osservi che quanto mostrato vale come esempio e in caso di dubbio conviene verificare il funzionamento del kernel con la mappa americana standard.

Tabella u178.15. Estratto dalla codifica CP 437, per la definizione della mappa della tastiera.

Ottale	Decimale	Esadecimale	Codice corrispondente nell'insieme di caratteri universale	Aspetto
202_8	130_{10}	82_{16}	U+00E9	é
205_8	133_{10}	85_{16}	U+00E3	ã
207_8	135_{10}	87_{16}	U+00E7	ç
212_8	138_{10}	$8A_{16}$	U+00E8	è
215_8	141_{10}	$8D_{16}$	U+00EC	ì
225_8	149_{10}	95_{16}	U+00F2	ò
227_8	151_{10}	97_{16}	U+00F9	ù
234_8	156_{10}	$9C_{16}$	U+00A3	£
256_8	174_{10}	AE_{16}	U+00AB	«
257_8	175_{10}	AF_{16}	U+00BB	»

[1] **Bootblocks** GNU GPL

[2] **Bootkit** GNU GPL