Il Linux ELF HOWTO

Daniel Barlow <daniel.barlow@sjc.ox.ac.uk> v1.11, 13 September 1995

Traduzione a cura di Renato Favro, nos@maya.dei.unipd.it. Questo documento descrive il modo per modificare il vostro sistema linux per compilare e far girare i programmi che usano il formato binario ELF. Il documento è diviso in tre parti: (1) Che cosa è l'ELF, e perchè e se è il caso di implementarlo, (2) Come fare l'upgrade, e (3) che cosa si può fare dopo l'upgrade.

1 Cosa è l'ELF? Un introduzione

L'ELF (Executable and Linking Format) è un formato binario originariamente sviluppato da USL (UNIX System Laboratories) e correntemente usato nei sistemi operativi Solaris e System V Release 4. A causa della maggiore flssibilità rispetto al vecchio formato a.out, che linux attualmente usa, gli sviluppatori di librerie per GCC e per C, hanno deciso lo scorso anno di muoversi nella direzione di usare anche l'ELF come formato binario standard.

Questa 'migliore flessibilità' si manifesta essenzialmente in due benefici per il programmatore medio di applicazioni:

- E' molto più semplice creare librerie condivise con l'ELF. Usualmente, basta compilare tutti i file oggetto con `-fPIC`, e poi fare il link con un comando come

  ```
  gcc -shared -Wl,-soname,libfoo.so.y -o libfoo.so.y.x *.o
  ```

  Se questo sembra complicato, ovviamente è perchè non avete mai letto la procedura equivalente per le librerie condivise usate dal sistema a.out, che comporta compilare la libreria due volte, riservare lo spazio per tutti i dati che si pensa saranno richiesti in futuro, e registrare quello spazio degli indirizzi con una terza parte. (Il tutto è descritto in un documento lungo 20 pagine — vedere a

  `ftp://tsx-11.mit.edu/pub/linux/packages/GCC/src/tools-2.16.tar.gz`

  per i dettagli).

- Il caricamento dinamico (dynamic loading, cioè il meccanismo usato dai programmi che possono caricare vari moduli durante l'esecuzione) è molto più semplice. Questo è usato da Perl 5, Python, e il port di Java su Linux, fra le altre cose. Altri suggerimenti per il caricamento dinamico hanno incluso MUDs ultra veloci, dove il codice aggiuntivo può essere compilato separatamente e il link può essere fatto mentre l'eseguibile gira, senza dover far fermare e ripartire il programma.

Detto questo, bisogna tener presente che l'ELF può essere un pò più lento. Le voci che si raccolgono in giro dicono tra l'1% e il 5%, sebbene tutti i test attuali che sono stati condotti fino a questo momento, indicano che la differenza è sufficientemente piccola per passare inosservata nel 'rumore' di altri eventi che possono avvenire nello stesso tempo. Se si ha il TeX oppure una stampante o un convertitore Postscript, si può leggere `speed.comp-1.0.tar.gz`, che è disponibile da qualche parte in SunSite. (?)

Il rallentamento deriva dal fatto che il codice delle librerie ELF deve essere indipendente dalla posizione (questo è quello che `-fPIC` stà ad indicare) e così un registro deve essere sacrificato per contenere gli offset, il che implica un registro in meno per mantenere le variabili, e comunque i processori 80x86 hanno una carenza di registri general-purpose di per sè.

1.1 Che cosa ELF non è

Esiste un certo numero di fraintendimenti riguardo a cosa l'ELF farà per il proprio sistema:

Non è una maniera per far girare programmi per SVR4 o per Solaris

Sebbene l'ELF sia lo stesso tipo di 'contenitore binario' usato da SVR4, questo non significa che i programmi SVR4 diventino improvvisamente eseguibili su di un Linux. La cosa è analoga al formato di un disco — si puossono tenere i programmi per Linux su di un disco formattato in MSDOS o in MINIX, e viceversa, ma questo non significa che questi sistemi diventino capaci di far girare l'uno i programmi dell'altro.

Teoricamente è possibile eseguire applicazioni per altri unix per processori x86 sotto Linux, ma seguendo le istruzioni contenute in questo HOWTO *non* non si avrà questo effetto. Per questa esigenza, si provi a guardare il modulo per il kernel iBCS (da qualche parte su `tsx-11.mit.edu`) e si veda se questo coincide con le proprie esigenze.

Non è intrinsecamente più piccolo o più veloce

Si può anche finire per avere dei binari più piccoli tuttavia, poichè si possono creare più facilmente librerie condivise di codice comune tra varie applicazioni. In generale, se si usano le stesse opzioni di compilazione e il proprio codice binario risulta più piccolo rispetto a quanto avveniva con il sistema a.out, è più verosimile che sia un caso fortunato, oppure a causa di una versione del compilatore differente. Per quanto riguarda la velocità, ne sarei veramente sorpreso. Gli incrementi di velocità potrebbero verificarsi se il proprio codice binario risultasse più piccolo, a causa di un minor swapping, o di aree funzionali più grandi che riuscirebbero ad essere contenute nella cache.

Non richiede che vengano rimpiazzati tutti i codici binari sul proprio sistema

Alla fine della procedura qui descritta, si avrà un sistema capace di compilare ed eseguire programmi sia in ELF che in a.out. I nuovi programmi verranno compilati in ELF per default, sebbene questo possa essere cambiato con uno switch sulla riga di comando del compilatore. Esiste per la verità una penalizzazione che riguarda la memoria, per un sistema configurato per far girare un sistema misto ELF/a.out — se si hanno entrambe le famiglie di programmi che girano insieme, si hanno anche due copie della libreria C nel core, e così via. Dalle informazioni che ho avuto, la differenza di velocità non è percettibile nell'uso normale su di un sistema con 6Mb, (io certamente non ne ho notata in 8 Mb), così difficilmente è un problema seccante. Si perde molta più memoria ogni giorno, facendo girare programmi sovrabbondanti come Emacs, e programmi statici come Mosaic/Netscape :-)

Non ha nulla a che fare con Tolkien

O, almeno, non in questo contesto.

1.2 Perchè (non) ci si dovrebbe convertire a ELF

Ci sono essenzialmente due motivi per fare l'upgrade del proprio sistema per poter compilare ed eseguire il codice binario in ELF: il primo è la migliore flessibilità già spiegata, e il secondo è che, a causa del primo, tutti quanti lo faranno. Le release future della libreria C e GCC, saranno compilate unicamente per ELF, e ci si aspetta che altri sviluppatori si muoveranno verso l'ELF.

Piacevolmente per gli scopi di simmetria, ci sono anche due ragioni per non convertirsi per ora. La prima è che le cose stanno ancora cambiando, alcuni pacchetti (includendo la serie dei kernel 'stabili' 1.2) richiedono dei patch (delle modifiche) prima di poter essere compilati in ELF, e ci possono essere dei bugs residui; si potrebbe aspettare finchè Linus stesso si sarà covertito, per esempio.

La seconda è che sebbene la procedura di installazione descritta qui sia un piccolo lavoro di per se stesso, (può infatti essere completato in meno di un'ora, se non contiamo il tempo per tirare giu' il nuovo software), un errore in qualunque momento puo' lasciare il sistema in maniera che non sia più capace di eseguire il bootstrap. Se non ci si sente a proprio agio con l'eseguire l'upgrade delle librerie condivise, e i comandi `ldconfig` e `ldd` non significano nulla per chi legge, si può ottenere o aspettare di ottenere una nuova distribuzione di Linux in ELF, fare il backup, reinstallare e ripristinare il sistema, usando l'ELF. Poi (specialmente se il sistema non è usato per applicazioni critiche), si può voler continuare comunque, per imparare cose nuove.

Siete ancora con noi?

2 Installazione

2.1 Background

Lo scopo di questa conversione è quello di rimanere con un sistema che sia in grado di costruire e di far girare i programmi in ELF e col sistema a.out, con entrambe i tipi di programmi in grado di trovare le rispettive famiglie di librerie condivise. Questo ovviamente richiede un pò più di intelligenza nella ricerca delle librerie rispetto al semplice 'guarda in `/lib`, `/usr/lib` e in tutti i posti in cui il programma è stato detto di cercare'.

La bestiola responsabile di ricercare le librerie in linux è `/lib/ld.so`. Il compilatore ed il linker non inseriscono nel codice i percorsi assoluti delle librerie dentro i programmi; inseriscono invece il nome delle librerie, e il percorso assoluto di `ld.so`, e lasciano che sia `ld.so` ad associare il nome della libreria al percorso adeguato al runtime. Questo ha un effetto molto importante — significa che le librerie che un programma usa, possono essere mosse verso altre directory *senza ricompilare il programma* , sempreché a `ld.so` sia stato detto di cercare nelle nuove directory. Questa è una funzionalità essenziale per l'operazione di scambio delle directory che segue.

Ne consegue che, naturalmente, ogni tentativo di cancellare o spostare `ld.so` causerà il fatto che *ogni programma che sia linkato dinamicamente sul sistema linux, cesserà di funzionare*. Questa è generalmente ritenuta una cattiva cosa...

Per il codice binario in ELF, viene fornito un dynamic loader (caricatore dinamico del codice) alternativo. Si chiama `/lib/ld-linux.so.1`, e fa esattamente le stesse cose di `ld.so`, ma per i programmi in ELF. `ld-linux.so.1` usa gli stessi programmi e gli stessi file di supporto (`ldd`, `ldconfig`, e `/etc/ld.so.conf`, che vengono usati dal loader del formato a.out.

L'idea di base, poi, è che le cose che riguardano lo sviluppo in ELF (compilatori, include files e librerie) vadano in `/usr/{bin,lib,include}` dove attualmente si trovano quelle che riguardano l'a.out, e che queste ultime vadano spostate in `/usr/i486-linuxaout/{bin,lib,include}`. `/etc/ld.so.conf` farà una lista di tutti i posti dove ci si può aspettare di trovare una libreria, e `ldconfig` è abbastanza intelligente da distinguere tra le varianti di ELF e di a.out.

Ci sono un paio di eccezioni al piazzamento delle librerie, tuttavia.

- Alcuni vecchi programmi sono stati costruiti senza l'uso di `ld.so`. Questi cesseranno di funzionare se le loro librerie saranno spostate. In questo modo, `libc.so*` e `libm.so*` devono rimanere dove stanno in `/lib`, e le versioni in ELF avranno il loro major number aumentato, in maniera da non sovrascrivere le versioni per a.out. Le vecchie librerie per X (precedenti alla versione 6), è meglio lasciarle dove stanno, sebbene le più nuove (`libX*.so.6`) debbano essere spostate. Lo spostamento delle vecchie librerie causerà un apparente malfunzionamento dei programmi xview, mentre non spostare le nuove librerie, causerà la sovrascrittura quando si installeranno le librerie in ELF per X.

  Se si hanno programmi non-ld.so che richiedono altre librerie rispetto a quelle già citate, (se si sa quali programmi sono, si può far girare ldd su di loro per capire di quali librerie abbiano bisogno *prima* di rovinarli), allora si hanno essenzialmente due possibilità. La prima è che si possono estrarre i file tar in una directory temporanea, controllare se le proprie preziose librerie saranno sovrascritte, e, se così fosse, si sposterà la versione in ELF della libreria in, diciamo, `/usr/i486-linux/lib` invece che `/lib`. Sia sia sicuri che il proprio `ld.so.conf` abbia `/usr/i486-linux/lib`, e si faccia girare `ldconfig` e non ci si pensi più. La seconda cosa è che si può ricompilare o acquisire una copia più recente del programma che crea questa seccatura. Questa non sarebbe affatto una cattiva idea, se possibile.

- Se si ha `/usr` e `/` su differenti partizioni, sarà necessario muovere almeno alcune delle librerie in `/lib` da qualche parte sul root disk, non in `/usr`. Si può sia identificare i programmi necessari al system startup o quando si è in modo single user (singolo utente), e trovare le librerie che usano, oppure può dipendere dal vostro integratore del sistema o della distribuzione, che può aver già fatto questo per voi, e semplicemente spostare tutte (beh... alcune. Si veda quanto già detto per le eccezioni) le librerie da `/lib` a `/lib-aout`.

2.2 Prima di partire — Note e diffide

- E' necessario che sul proprio sistema giri un kernel posteriore alla versione 1.1.52 **con il supporto per il formato binario ELF**.
- E' molto raccomandabile preparare o procurarsi dei dischi di root/boot, ad esempio come quelli di Slackware. Probabilmente non se ne avrà bisogno, ma se succede di averne un paio e ci si ritrova sprovvisti, ci prenderemmo a calci da soli. Seguendo questa attitudine del tipo 'prevenire è meglio che curare', possono essere d'aiuto per ogni situazione scomoda in cui si può finire, delle copie linkate staticamente di `mv`, `ln`, e forse anche di altre utility per la manipolazione dei file (sebbene di fatto possa essere fatto tutto quello di cui si ha *bisogno* con le possibilità intrinseche della shell), per ogni situazione scomoda in cui si può finire.
- Se avete seguito lo sviluppo dell'ELF, potete avere le librerie ELF in `/lib/elf` (tipicamente `libc.so.4` e compagnia varia). Le applicazioni costruite in questa maniera devono essere ricostruite, e poi la directory rimossa. Non è necessario avere una directory `/lib/elf`!
- La maggior parte delle installazioni Linux di questi giorni, hanno aderito al 'FSSTND' standard file system, ma senza dubbio ci sono ancora dei sistemi che non seguono lo standard. Se si vedono referenze a `/sbin/`*qualcosa* e non si ha una directory `/sbin`, propabilmente si troveranno i programmi relativi in `/bin` o in `/etc/`.

2.3 Si avrà bisogno di...

I seguenti pacchetti sono disponibili a `ftp://tsx-11.mit.edu/pub/linux/packages/GCC/` e a `ftp://sunsite.unc.edu/pub/Linux/GCC/` . Entrambe i siti sono ampiamente distribuiti in giro per il mondo; per favore, prendetevi il tempo necessario per cercare il sito più vicino a voi. E' più veloce per voi e per chiunque altro.

Questi pacchetti (sia la versione mostrata qui che qualunque altra successiva) sono richiesti. Scaricate e leggete anche le varie release notes per ciascuno di essi: questi sono i file chiamati `release.`*packagename*. Questo specialmente se si scaricano versioni più recenti di quelle descritte qui, poichè la procedura può essere cambiata.

- `ld.so-1.7.3.tar.gz` — il nuovo linker dinamico
- `libc-5.0.9.bin.tar.gz` — le immagini ELF condivise per la libreria C e i suoi amici (`m` (maths), `termcap`, `gdbm`, e così via), più le librerie statiche corrispondenti e gli include files necessari per compilare i programmi con queste. libc 5.2.qualcosa potrebbe essere rilasciata durante la vita di questo HOWTO, ed è notevolmente differente dalla versione 5.0.9; se si intende installarla, si procede per conto proprio, ma io raccomando di installare prima la 5.0.9 e poi installare 'sopra' la versione successiva. Ci

sono molte parti della 5.0.9 che non sono incluse nella 5.2.x, per le quali i canali di distribuzione non sono ancora predisposti completamente.

- `gcc-2.7.0.bin.tar.gz` — il compilatore C in ELF. Include anche un compilatore C in a.out che capisce il nuovo ormato delle directory.
- `binutils-2.5.2l.17.bin.tar.gz` — le utility binarie della GNU per Linux. questi sono programmi come `gas`, `ld`, `strings` e così via, la maggior parte dei quali sono richiesti per far funzionare il compilatore C.

2.4 Ridisponendo il proprio filesystem

Bene.... Si noti che in tutto quel che segue, quando io dico 'si rimuova', naturalmente intendo 'si faccia un backup e poi si rimuova' :-). Inoltre, queste istruzioni si applicano direttamente soltanto alle persone che non hanno già smanettato con l'ELF — quelli che ci si aspetta abbiano già la capacità di adattare le cose in modo appropriato. Andiamo!

1. Creare le nuove directory verso cui si sposteranno le cose di a.out

```
mkdir -p /usr/i486-linuxaout/bin
mkdir -p /usr/i486-linuxaout/include
mkdir -p /usr/i486-linuxaout/lib
mkdir /lib-aout
```

2. Si scompatti il pacchetto del linker dinamico `ld.so-1.7.3` nella directory in cui di solito si mette il codice sorgente, poi si legga lo script `ld.so-1.7.3/instldso.sh` appena estratto. Se si ha un sistema realmente standard, è sufficiente farlo girare digitando `sh instldso.sh`, ma se si ha qualcosa di insolito allora si faccia l'installazione a mano. 'Qualcosa di insolito' comprende
   - usare zsh come shell (alcune versioni di zsh definiscono `$VERSION`, il che sembra confondere `instldso.sh`)
   - avere link simbolici da `/lib/elf` a `/lib` (il che non è ncessario, ma si possono aver avuto delle valide ragioni per farlo se si è seguito lo sviluppo dell'ELF).
3. Si editi `/etc/ld.so.conf` per aggiungere le nuove directory `/usr/i486-linuxaout/lib` (e `/lib-aout` se si pensa di averne bisogno). Poi si faccia girare `/sbin/ldconfig -v` per controllare che sono state caricate le nuove directory.
4. Si muovano tutte le librerie a.out da `/usr/*/lib` a `/usr/i486-linuxaout/lib`. Si noti che ho scritto 'librerie' non 'qualunque cosa'. Quindi si intende i file che rispondono al nome di `lib*.so*` , `lib*.sa*`, o `lib*.a`. Non si incominci a spostare `/usr/lib/gcc-lib` o niente di sciocco come questo in giro.
5. Adesso si guardi a `/lib`. Si lasci intatto `libc.so*`, `libm.so*`, e `libdl.so*`. Se si hano link simbolici a librerie X (`libX*.so.3*`) lasciate li anche questi — XView e qualche altro pacchetto potrebbe averne bisogno. Si lasci `ld.so*`, `ld-linux.so*` e qualunque altro file che comincia con `ld`. Per le librerie rimanenti, (se ne avete qualcuna): se si ha `/usr` sulla partizione root, si mettano in `/usr/i486-linuxaout/lib`. Se si ha `/usr` montata separatamente, si mettano in `/lib-aout`. Adesso si faccia girare `ldconfig -v`
6. Si rimuova la directory `/usr/lib/ldscripts` se è li, per la preparazione all'installazione delle utility binarie (che la ricreeranno).
7. Si rimuova qualsiasi copia di `ld` e di `as` (*eccetto* `ld86` e `as86`) che si può trovare in `/usr/bin`.
8. Alcune versioni del GNU tar sembrano avere problemi riguardo ai link simbolici nella directory di destinazione. Si hanno due possibilità a questo punto:

(a) (perferibilmente) Si usi `cpio` invece di `tar`, poichè non ha questo problema. `zcat /ovunque/sia/stato/messo/libc-5.0.9.tar.gz | cpio -iv` è l'incantesimo magico qui, che deve essere eseguito dalla directory root.

(b) (se non si ha installato cpio) Prima di installare le immagini della libc, si può andare in `/usr/include` e rimuovere alcune parti.

Questa è brutta. Molti pacchetti (come ncurses) sono installati in `/usr/include` dai manutentori della distribuzione e *non* sono fornite con la libreria C. Si faccia un backup dell'albero della directory `/usr/include`, si usi `tar tzf` per vedere cosa c'è nell'archivio prima di scompattarlo, poi cancellare le poche cose di `/usr/include` che sono al momento contenute. Poi si scompatti il file `libc-5.0.9.bin.tar.gz` dalla directory root.

9. installare il pacchettocon le utility binarie. `tar -xvzf binutils-2.5.2.l17.bin.tar.gz -C /` è una buona maniera per farlo.

10. A questo punto è stato installato tutto quello che serve per far girare gli eseguibili in ELF. Gli esperti medici raccomandano che i lavoratori VDU (VDU è un acronimo per Video Display Unit, quindi si indicano qui i lavoratori che usano qualunque dispositivo con un monitor N.d.T.) prendano delle pause regolari dallo schermo; questo sarebbe un momento opportuno. Non si dimentichi quello che si stava facendo tuttavia, a seconda della versione di gcc che si stava precedentemente usando, è possibile che non si sia in grado di compilare programmi in formato a.out, fino a che non si installa il nuovo gcc.

11. fare il backup e cancellare tutto in `/usr/lib/gcc-lib/{i486-linux, i486-linuxelf, i486-linuxaout}/` Se si usa un driver `gcc` non standard, (per esempio se si usa il GNU ADA), si copi anche quello in un posto sicuro. Poi si installi in pacchetto gcc, di nuovo scompattando il tutto dalla directory root.

12. Alcuni programmi (solitamente vari programmi per X) usano `/lib/cpp`, che generalmente sotto linux è un link a `/usr/lib/gcc-lib/i486-linux/`*version*`/cpp`. Siccome ilpasso precedente ha cancellato qualsiasi versione a cui `cpp` stava puntando, bisognerà ricreare il link:

```
$ cd /lib
$ ln -s /usr/lib/gcc-lib/i486-linux/2.7.0/cpp .
```

13. La gente di FSSTND hanno ancora una volta giustificato la loro esistenza muovendo i file `utmp` e `wtmp` da `/var/adm` a `/var/run` e a `/var/log` rispettivamente. Bisognerà aggiungere alcuni link che dipendono da dove attualmente questi risiedono, e può essere necessario creare le directory `/var/log` and `/var/adm`. Io ho riprodotto qui sotto l'output del comando `ls -l` delle parti appropriate del mio sistema:

```
$ ls -ld /var/adm /var/log /var/run /var/log/*tmp /var/run/*tmp
lrwxrwxrwx   1 root     root            3 May 24 05:53 /var/adm -> log/
drwxr-xr-x   9 root     root         1024 Aug 13 23:17 /var/log/
lrwxrwxrwx   1 root     root           11 Aug 13 23:17 /var/log/utmp -> ../run/utmp
-rw-r--r--   1 root     root       451472 Aug 13 23:00 /var/log/wtmp
drwxr-xr-x   2 root     root         1024 Aug 13 23:17 /var/run/
-rw-r--r--   1 root     root          448 Aug 13 23:00 /var/run/utmp
```

Si controlli il FSSTND (dagli archivi LDP come `ftp://sunsite.unc.edu/pub/Linux/docs/fsstnd/`) per la storia completa.

14. Questo passo è opzionale. Se si ha intenzione di continuare a compilare programmi in formato a.out, questo è il momento ideale per installare libc.so 4.7 x. Lo si scompatti dalla directory root, poichè a questo punto si è senz'altro in grado di farlo senza ulteriori spiegazioni.

Fatto! Semplici test che si possono provare sono:

```
$ gcc -v
Reading specs from /usr/lib/gcc-lib/i486-linux/2.7.0/specs
gcc version 2.7.0
$ gcc -v -b i486-linuxaout
Reading specs from /usr/lib/gcc-lib/i486-linuxaout/2.7.0/specs
gcc version 2.7.0
$ ld -V
ld version cygnus/linux-2.5.21.14 (with BFD cygnus/linux-2.5.21.11)
  Supported emulations:
   elf_i386
   i386linux
   i386coff
```

seguiti ovviamente dal tradizionale programma “Hello World”. Si provi con `gcc` e con `gcc -b i486-linuxaout` per verificare che sia il compilatore ELF, sia il compilatore a.out sono configurati correttamente.

2.5 Come dovrebbe sembrare (schema della struttura delle directory)

Questa è una guida deliberatamente vaga a che cosa sono i files che avete appena installato. Può essere utile per la soluzione di problemi, o per decidere cosa cancellare.

2.5.1 /lib

- Linker dinamici `ld.so` (a.out) e `ld-linux.so.1` (ELF). Entrambe possono essere link simbolici, ma siate sicuri che i file a cui puntano esistano.
- Librerie condivise di base `libc.so.4`, `libm.so.4` (a.out) Questi sono link simbolici, ma curate che anche questi puntino a file esistenti.
- Librerie condivise di base `libc.so.5`, `libm.so.5`, `libdl.so.1`, `libcurses.so.1`, `libtermcap.so.2`, (ELF). Anche questi sono link simbolici.
- *Un mucchio* di link simbolici. Per ciascuna libreria, dovrebbe esserci un file attuale (per esempio `libc.so.5.0.9`), un link simbolico che punta allo stesso con solo il major number della versione (`libc.so.5`) ed un link simbolico che punta a quest'ultimo senza numero di versione (`libc.so`). In questa maniera:

```
lrwxrwxrwx   1 root  root      9 May 24 05:52 libc.so -> libc.so.5
lrwxrwxrwx   1 root  root     13 Aug 25 12:48 libc.so.5 -> libc.so.5.0.9
-rwxr-xr-x   1 bin   bin  562683 May 19 04:47 libc.so.5.0.9
```

2.5.2 /usr/lib

- Tutti i file che non sono librerie e che esistevano già in precedenza.
- `libbfd.so*`,`libdb.so*`, `libgdbm.so*`, le librerie condivise di ELF. Tutte consistono di tre file come spiegato in precedenza nella sezione `/lib`.
- `libbsd.a`, `libgmon.a`, `libldso.a`, `libmcheck.a`, `libieee.a`, `libmcheck.a` ed un `lib*.a` file per ogni libreria condivisa in ELF. Quelli che duplicano le librerie condivise possono non essere estremamente utili per la maggioranza delle persone — quando si usa l'ELF si può usare lo switch `gcc -g` con le librerie condivise, così non ha più molto senso fare compilazioni statiche.

- `crt0.o`, `gcrt0.o`. a.out, che sono i file di 'inizio di programma';uno di questi è linkato come primo file in ogni programma di tipo a.out, a meno che non si prendano degli accorgimentio per evitarlo.
- `crt1.o`, `crtbegin.o`, `crtbeginS.o`, `crtend.o`, `crtendS.o`, `crti.o`, `crtn.o`, `gcrt1.o`. I file di startup dell'ELF. Questi fanno una cosa simile a quanto appena visto per i file `*crt0.o`

2.5.3 /usr/lib/ldscripts

- Questo è dove risiede lo script driver per `ld`, come suggerisce il nome stesso. Dovrebbe apparire come

```
$ ls /usr/lib/ldscripts/
elf_i386.x      elf_i386.xs      i386coff.xn      i386linux.xbn
elf_i386.xbn    elf_i386.xu      i386coff.xr      i386linux.xn
elf_i386.xn     i386coff.x       i386coff.xu      i386linux.xr
elf_i386.xr     i386coff.xbn     i386linux.x      i386linux.xu
```

2.5.4 /usr/i486-linux/bin

- `ar`, `as`, `gasp`, `ld`, `nm`, `ranlib`, `strip`. Questi attualmente sono tutti link simbolici alle utility binarie in `/usr/bin`

2.5.5 /usr/i486-linuxaout/bin

- `as` — L'assembler per l'a.out, e `gasp`, il suo preprocesore delle macro
- `ar`, `ld`, `nm`, `ranlib`, `strip` — link simbolici alle utility binarie in `/usr/bin`

2.5.6 /usr/i486-linux/lib

- `ldscripts` è un link simbolico a `/usr/lib/ldscripts`.

2.5.7 /usr/i486-linuxaout/lib

- `lib*.so*`. librerie condivise per a.out. Sono necesarie per far girare programmi in formato a.out.
- `lib*.sa`. mozziconi di librerie per il formato a.out. Necessarie per compilare programmi a.out. Se non si intende farlo, si possono tranquillamente rimuoverle.
- `lib*.a`. librerie statiche per il formato a.out. Necessarie per compilare programmi statici in a.out (ciaoè quando si compila con `-g`). Di nuovo, si possono rimuovere se non si ha questa intenzione.
- `ldscripts` è un link simbilico a `/usr/lib/ldscripts`

2.5.8 /usr/lib/gcc-lib/i486-linux/2.7.0

- Questa directory contiene una versione di gcc 2.7.0 predisposta per compilare programmi in ELF.

2.5.9 /usr/lib/gcc-lib/i486-linuxaout/2.7.0

- Questa directoryy contiene una versione del gcc 2.7.0, configurata per compilare programmi in a.out, che riconosce la nuova struttura delle directory. Se non si ha intenzione di compilare nulla in format a.out, si può cancellare questa directory. liberando in questo modo 4Mb.

2.6 Errori comuni (Niente Panico!)

... in grandi lettere amichevoli.

Avete spostato le cose sbagliate e non funziona nulla.

Avete ancora una shell in grado di girare, tuttavia, e con un pò di ingenuità potete fare un sacco di cose con le caratteristiche intrinseche della shell. Ricordate che `echo *` è un sostituto accettabile per `ls`, e che `echo >>filename` può essere usato per aggiungere linee ad un file. Inoltre non dimenticate che `ldconfig` è linkato staticamente. Se avete spostato, ad esempio, `libc.so.4` in `/lib-aout` erroneamente, potete fare `echo "lib-aout" >>/etc/ld.so.conf ; ldconfig -v` ed essere di nuovo a posto. se avete spostato `/lib/ld.so` potete essere capaci di fare `sln /stupido/posto/ld.so /lib/ld.so`, se avete una copia di ln linkata staticamente, e probabilmente vi ritroverete con tutto a posto.

`no such file or directory: /usr/bin/gcc`

... quando sapete che *esiste* quel file. Questo solitamente significa che il loader dinamico dell'ELF `/lib/ld-linux.so.1` non è installato, o non è leggibile per qualche motivo. Dovreste averlo installato al passo 2 in precedenza.

`not a ZMAGIC file, skipping`

dal comando `ldconfig`. Questo è perchè avete una vecchia versione del pacchetto ld.so, così dovete prenderne una nuova. Riguardate il passo 2dell'installazione.

`bad address`

tentando di far girare qualunque cosa in ELF. State usando il kernel 1.3.x, dove x<3. Fate un upgrade alla versione 1.3.3 oppure fate un downgrade a qualche versione 1.2.x

`_setutent: Can't open utmp file`

Questo messaggio è spesso visto in multipli di tre quando si fa partire un xterm. Andate avanti e leggete il papiro sul FSSTND verso la fine della procedura di installazione.

`gcc: installation problem, cannot exec` *`qualcosa`*`: No such file or directory`

Quando si tenta di fare una compilazione in a.out (*qualcosa* in genere sono `cpp` o `cc1`). Può essere sia una cosa corretta, oppure verosimilmente avete digitato

```
$ gcc -b -i486-linuxaout
```

quando avreste dovuto digitare

```
$ gcc -b i486-linuxaout
```

Notate che 'i486' *non* inizia con un meno.

3 Costruire programmi in ELF

3.1 Programmi ordinari

Per costruire programmi in ELF, usate `gcc` come sempre. Per costruire programmi in formato a.out usate la forma `gcc -b i486-linuxaout` .

```
$ cat >hello.c
main() { printf("hello, world\n"); }
^D
$ gcc -o hello hello.c
$ file hello
hello: ELF 32-bit LSB executable i386 (386 and up) Version 1
$ ./hello
hello, world
```

Questo è forse il momento appropriato per rispondere alla domanda "se il compilatore in a.out per default produce un programma chiamato `a.out`, che nome gli darà un compilatore ELF?". Sempre `a.out` è la risposta. Noia, noia, noia, ... `:-)`

3.2 Costruire le librerie

Per costruire libfoo.so come libreria condivisa, i passi di base da compiere assomigliano a questi:

```
$ gcc -fPIC -c *.c
$ gcc -shared -Wl,-soname,libfoo.so.1 -o libfoo.so.1.0 *.o
$ ln -s libfoo.so.1.0 libfoo.so.1
$ ln -s libfoo.so.1 libfoo.so
$ export LD_LIBRARY_PATH=`pwd`:$LD_LIBRARY_PATH
```

questo genererà una libreria condivisa chiamata `libfoo.so.1.0`, e il collegamento appropriato per ld (`libfoo.so`) e il linker dinamico (`libfoo.so.1`) per trovarla. Per provare, aggiungiamo la directory corrente a `LD_LIBRARY_PATH`.

Una volta contenti che la libreria funziona, bisogna spostarla, per esempio, in `/usr/local/lib`, e ricreare il link appropriato. Si noti che il link `libfoo.so` deve puntare a `libfoo.so.1`, così che non c'è bisogno di aggiornamenti ad ogni cambio della versione del minor number. il link da `libfoo.so.1` a `libfoo.so.1.0` è mantenuto a posto da `ldconfig`, che sulla maggior parte dei sistemi è fatto girare automaticamente al bootstrap.

```
$ su
# cp libfoo.so.1.0 /usr/local/lib
# /sbin/ldconfig
# ( cd /usr/local/lib ; ln -s libfoo.so.1 libfoo.so )
```

3.3 Programmi con il dynamic loading

Questo argomento è trattato estensivamente nel documento di H. J. Lu 'ELF programming' e nella pagina del manuale `dlopen(3)` che può essere trovata nel pacchetto ld.so. Qui c'è un piccolo esempio: fatene il link con `-ldl`

```
#include <dlfcn.h>
#include <stdio.h>

main()
{
  void *libc;
  void (*printf_call)();

  if(libc=dlopen("/lib/libc.so.5",RTLD_LAZY))
  {
```

```
    printf_call=dlsym(libc,"printf");
    (*printf_call)("hello, world\n");
  }

}
```

3.4 Debugging

La vostra copia esistente del `gdb` funzionerà prevalentemente senza cambiamenti, con i programmi in ELF. La nuova versione nella directory `GCC` su tsx-11 è ritenuta migliore per fare il debugging dei programmi che usano le librerie condivise e il dinamic loading, e anche per capire i core dump dei programmi in ELF.

Si noti che la serie di kernel 1.2 non possono generara i core dump dai programmi in ELF in nessun modo. La serie 1.3 invece può.

4 Patch e codice binario vario

A questo punto della procedura, è possibile fermarsi, se si vuole. Abbiamo installato tutto quello che è necessario per compilare e far girare i programmi in ELF.

Si può voler ricostruire alcuni programmi in ELF, sia per scopi di 'pulizia' sia per minimizzare l'uso della memoria. per la maggior parte delle applicazioni end-user (cioè per l'utente finale), questo è molto semplice; alcuni pacchetti tuttavia assumono troppe informazioni riguardo al sistema su cui girano, e possono fallire a causa di uno o più di questi motivi:

- Convenzioni differenti per quanto riguarda il carattere di sottolineatura in assembler: nel formato a.out, le labels esterne sono precedute da `_` , mentre in un eseguibile ELF no. Questo non fà differenza fino a che non si integra codice assembler scritto a mano: tutte le labels della forma `_foo`, devono essere tradotte in `foo` o (se si vuole fare codice portatile) in `EXTERNAL(foo)` dove `EXTERNAL` è una qualche macro che ritorna il suo argomento (se `__ELF` è definito) oppure `_` concatenato con il suo argomento se non è così.
- Differenze nella libc 5 dalla libc 4. L'interfaccia verso il supporto locale è cambiato, per dirne una.
- L'applicazione o il processo di costruzione del codice, che dipende dalla conoscenza del formato binario usato, — emacs per esempio copia l'immagine della sua memoria su disco in formato eseguibile, così ovviamente ha bisogno di sapere in che formato sono i vostri eseguibili.
- L'applicazione consiste di (oppure usa) le librerie condivise (X11 è un esempio ovvio). Queste ovviamente avranno bisogno di qualche cambiamento per accomodare il metodo differente di creazione delle librerie in ELF.

Adogni modo, ecco due liste: la prima è di programmi che hanno bisogno di cambiamenti per l'ELF, dove i cambiamenti sono già stati fatti (cioè avete bisogno di nuove versioni per compilarle come ELF), e la seconda è di programmi che ancora hanno bisogno di patch da terze parti.

4.1 Upgrade:

- **Dosemu**. Il modulo di tre dei quattro alberi del corrente albero dello sviluppo (non chiedere... meglio collegarsi direttamente alla linux-msdos mailing list) del dosemu funziona con l'ELF. Non c'è bisogno di diventare matti con il Makefile. Versioni correnti del dosemu sono disponibili a `ftp://tsx-11.mit.edu/pub/linux/ALPHA/dosemu/`

- **Emacs**. Emacs ha una procedura di costruzione che comprende il far girare una versione minima di se stesso, caricare tutto il codice utile come lisp, e poi scaricare la sua immagine in memoria di nuovo su disco come un file eseguibile. (FSF) Emacs 19.29 e XEmacs 19.12 (che in precedenza si chiamava Lucid Emacs) possono entrambe capire se si sta usando l'ELF e fare le cose giuste automaticamente.
- **MAKEDEV**. In alcune implementazioni, questa utility rimuove i device esistenti prima di ricrearli. Questa è una brutta notizia se succede di toccare `/dev/zero`, poichè questo device è necessario per le operazioni di tutti i programmi in ELF. Si controlli il pacchetto util-linux per una versione corretta.
- **perl 5.001**. Perl 5.00 più i patch "ufficialmente non ufficiali" a-e, si compila senza modifiche sui sistemi con l'ELF, comprendendo il dynamic loading. I patch sono disponibili da `ftp.metronet.com` oppure da `ftp.wpi.edu`
- Il programma `cal` in **util-linux 2.2** non funziona. Si faccia un upgrade alla

  versione 2.4 `ftp://tsx-11.mit.edu/pub/linux/packages/utils` o successiva.
- **XFree86**. XFree86 3.1.2 esiste sia in formato ELF che in formato a.out. Si faccia `ftp` a `ftp.xfree86.org`, e dopo aver letto il messaggio 'too many users' che è praticamente sicuro otterrete, e trovate il mirror più vicino a voi. Una volta ottenuto il contenuto delle directory `common` e `elf`, si deve modificare `/usr/X11R6/lib/X11/config/linux.cf` per cambiare le linee che dicono

  ```
  #define LinuxElfDefault         NO
  #define UseElfFormat            NO
  ```

  in modo da dire `YES` invece. In caso contrario una costruzione di xpm tenterà di fare le cose sbagliate con `jumpas` e le relative reliquie del passato
- **Mosaic**. Non ho certo il modo di ricostruirlo per conto mio, ma il codice binario del Mosaic 2.7b1 disponibile dalla NCSA, è in ELF. Tuttavia il link è stato eseguito con una versione errata di X, col risultato che su sistemi normali si lamenterà di non trovare `libXpm.so.4.5`. La correzione abbastanza semplice è di editarlo con cura con Emacs o con un altro editor che gestisce correttamente i file binari. Dopo aver trovato le occorrenze della stringa `libXpm.so.4.5^@` (dove `^@` è un carattere NUL — ASCII zero —), cancellare `.5` e aggiungere due caratteri dopo il NUL per evitare il cambiamento della lunghezza del file.

4.2 Patch

- **e2fsutils**. Le Utility per il Second Extended File System hanno bisogno di un patch da `ftp://ftp.ibp.fr/pub/linux/ELF/patches/e2fsprogs-0.5b.elf.diff.gz`

  per essere compilate correttamente come librerie condivise. Remy Card dice "Questo è il patch per l'ELF che probabilmente verrà incluso nella prossima versione di e2fsck e compagnia bella."
- **file**. Questo funziona in ogni caso, ma può essere migliorato: `http://sable.ox.ac.uk/~jo95004/patches/file.diff` aggiunge il supporto per identificare in file binari in ELF che hanno subito lo strip e che sono 'mixed-endian'.
- **Il Kernel**. Almeno dalla versione 1.3.8, lo sviluppo della serie di kernel 1.3 hanno una opzione nel `make config` per permettere di costruirlo usando i tool dell'ELF. Se si sta usando la serie 1.2, avete due possibilità:

  1. Si modifichi leggermente il Makefile per usare il compilatore a.out. Basta cambiare le definizioni `CC` e `LD` in modo che appaiano come

```
LD      =ld -m i386linux
CC      =gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

In alternativa,

2. Si applichino i patch di H J Lu che permettono di compilare il kernel in ELF (e inoltre aggiungono la possibilitè di fare i core dump in ELF).

Lasciatemi ripetere che ne l'una ne l'altra sono necessarie se si sta usando la serie 1.3 dei kernel.

- `ps` **(procps-0.97)** Il programma `psupdate` ha bisogno di un patch per lavorare se si è compilato il kernel in ELF. Questo è disponibile in

  `linux.nrao.edu:/pub/people/juphoff/procps` , sia come patch a vanilla 0.97 e sia come intero file di tar. Una nuova versione di procps dovrebbe essere distribuita presto con il patch già a posto, così se si potrà trovare procps 0.98 nel momento in cui leggerete questo, allora questo patch sarà probabilmente obsoleto.

5 Informazioni aggiuntive

- La mailing list `linux-gcc` è davvero il miglior posto per vedere cosa sta succedendo, di solito senza mai pubblicarci qualcosa. Ricordate, questo non è Usenet, perciò tenete per voi le vostre domande a meno che non stiate sviluppando. Per istruzioni per ottenere questa mailing list, si mandi un messaggio contenente la parola `help` a `majordomo@vger.rutgers.edu`. Gli archivi di questa mailing list sono a

  `http://homer.ncm.com/` .

- C'è un certo ammontare di informazioni riguardo quello che la lista linux-gcc sta facendo alla mia *pagina web sull'ELF* `http://sable.ox.ac.uk/~jo95004/elf.html` , quando mi ricordo di aggiornarla. Questa ha anche un link all'ultima versione di questo HOWTO, e i pacchetti a cui questo si riferisce. Per le persone negli Stati Uniti, e per gli altri con collegamenti scarsi ai siti accademici della Gran Bretagna (cioè praticamente chiunque fuori dall'ambiente accademico della Gran Bretagna), questa pagina e' completamente replicata a

  `http://www.blackdown.org/elf/elf.html`

- Si veda anche la *esperienza di upgrade ad ELF* `http://www.intac.com/~cully/elf.html`

  di Bobby Shmit

- La *GCC-FAQ* `ftp://sunsite.unc.edu/pub/Linux/docs/faqs/GCC-FAQ.html` contiene informazioni più generali sullo sviluppo e alcuni dettagli tecnici sull'ELF.

- Esiste anche la documentazione sul formato del file su *tsx-11* `ftp://tsx-11.mit.edu/pub/linux/packages/GCC/ELF.doc.tar.gz` . Questa pagina probabilmente sarà più utile alle persone che vogliono capire, fare il debug o riscrivere programmi che si cimentano direttamente con oggetti binari.

- Il documento di H J Lu *ELF: From The Programmer's Perspective* `ftp://tsx-11.mit.edu/pub/linux/packages/GCC/elf.latex.tar.gz`

  contiene informazioni molto utili e più dettagliate sul programmare con l'ELF. Se non si è in grado di gestire file in formato LaTeX, è anche disponibile in PostScript

- Esiste una pagina del manuale (man page N.d.T.) `dlopen(3)` fornita con il pacchetto `ld.so`.

6 Legalese

Tutti i trademark usati in questo documento sono riconosciuto come appartenenti ai rispettivi prorietari. *(Scovate l'ironia a questo punto...)*