Building and Installing Software Packages for Linux

Mendel Cooper mailto:thegrendel@theriver.com — *http://personal.riverusers.com/˜thegrendel/* v1.91, 27 luglio 1999

- Compilazione ed installazione di pacchetti software per Linux - Questa è un'ampia guida per compilare ed installare "generiche" distribuzioni di software UNIX sotto Linux. Vengono inoltre discussi i formati binari preimpacchettati "rpm" e "deb". Traduzione di *Fabrizio Stefani* , 23 settembre 1999.

Indice

1 Introduzione

Parecchi pacchetti software per i vari dialetti di UNIX e Linux sono dati come archivi compressi di file sorgenti. Lo stesso pacchetto può essere compilato per girare su differenti macchine fissate, e ciò risparmia l'autore del software dal dover produrre versioni multiple. Una singola versione di un pacchetto software può così finire col girare, in varie incarnazioni, su una macchina Intel, un DEC Alpha, una workstation RISC, o anche un mainframe. Sfortunatamente, questo scarica la responsabilità della effettiva compilazione ed installazione del software sull'utente finale, l'«amministratore di sistema» de facto, il tizio seduto alla tastiera – voi. Fatevi coraggio, comunque, il processo non è poi così terrificante o misterioso come sembra, come dimostrerà questa guida.

2 Spacchettare i file

Avete scaricato o vi siete procurati in altro modo un pacchetto software. Molto probabilmente è archiviato (in formato *tar*) e compresso (in formato *gzip*), e quindi il nome del file terminerà con `.tar.gz` o `.tgz` (N.d.T: Gli archivi tar compressi, in inglese, vengono colloquialmente detti tarball, d'ora in poi ci riferiremo ad essi come pacchetti tar). Innanzi tutto copiatelo in una directory di lavoro. Poi decomprimetelo (con *gunzip*) e spacchettatelo (con *tar*). Il comando appropriato per farlo è **tar xzvf *nomefile***, dove *nomefile* è il nome del file, ovviamente. Il processo di dearchiviazione generalmente installerà i file appropriati nelle sottodirectory che avrà creato. Notate che se il nome del pacchetto ha suffisso *.Z*, la procedura su esposta sarà ancora buona, sebbene funzioni anche eseguire **uncompress**, seguito da **tar xvf**. Potete vedere un'anteprima di tale processo con **tar tzvf nomefile**, che elenca i file contenuti nell'archivio senza in effetti estrarli.

Il suddetto metodo per spacchettare i pacchetti tar è equivalente ad uno o l'altro dei seguenti:

- gzip -cd nomefile | tar xvf -
- gunzip -c nomefile | tar xvf -

(Il '-' forza il comando *tar* a prendere il suo input dallo `stdin`.)

I file sorgenti nel nuovo formato *bzip2* (`.bz2`) possono essere estratti con un **bzip2 -cd nomefile** | **tar xvf -**, o, più semplicemente, con un **tar xyvf nomefile**, sempre che `tar` sia stato opportunamente corretto con l'apposita patch (riferirsi al *Bzip2.HOWTO* 16 ((tradotto)) per i dettagli). La distribuzione Linux di Debian usa una diversa patch per `tar`, scritta da Hiroshi Takekawa, che, in quella particolare versione di `tar`, usa le opzioni *-I, –bzip2, –bunzip2*.

[Grazie tante a R. Brock Lynn e Fabrizio Stefani per le correzioni e gli aggiornamenti sull'informazione sopra citata]

A volte i file archiviati devono essere estratti, usando tar, ed installati dalla home directory dell'utente, o magari in una cert'altra directory, tipo `/`, `/usr/src`, o `/opt`, come specificato nelle informazioni di configurazione del pacchetto. Qualora si riceva un messaggio di errore tentando l'estrazione dall'archivio, questa potrebbe esserne la ragione. Leggete i file di documentazione del pacchetto, specialmente i file `README` e/o `Install`, se presenti, ed editate i file di configurazione e/o i `Makefile` come necessario, consistentemente con le istruzioni di installazione. Osservate che di solito **non** si dovrebbe modificare il file `Imake`, poiché ciò potrebbe avere conseguenze impreviste. La maggior parte dei pacchetti permettono di automatizzare questo processo eseguendo **make install** per mettere i binari nelle appropriate aree di sistema.

- Potreste incontrare file `shar`, o *archivi shell*, specialmente sui newsgroup riguardanti il codice sorgente in Internet. Questi vengono ancora usati perché sono in formato testo, e ciò permette ai moderatori dei newsgroup di consultarli e respingere quelli inadatti. Essi possono essere spacchettati col comando **unshar nomefile.shar**. Altrimenti la procedura per trattarli è la stessa dei pacchetti tar.

- Alcuni archivi sorgente sono stati manipolati usando utilità di compressione non standard DOS, Mac o anche Amiga, tipo *zip*, *arc*, *lha*, *arj*, *zoo*, *rar*,e *shk*. Fortunatamente, *Sunsite* `http://metalab.unc.edu` e altri posti hanno delle utilità di decompressione per Linux che possono trattare molti o tutti tali formati.

Occasionalmente, potrebbe essere necessario aggiungere delle correzioni per dei bug o aggiornare i file sorgenti estratti da un archivio usando un file `patch` o `diff` che elenca le modifiche. I file di documentazione e/o `README` vi informeranno se ciò è necessario. La normale sintassi per invocare la potente utilità di *patch* di Larry Wall è **patch < patchfile**.

Ora potete procedere alla fase di compilazione del processo.

3 Usare make

Il `Makefile` è la chiave del processo di compilazione. Nella sua forma più semplice, un Makefile è uno script per la compilazione, o building, dei binari, le parti eseguibili di un pacchetto. Il Makefile può anche fornire un mezzo per aggiornare un pacchetto software senza dover ricompilare ogni singolo file sorgente in esso, ma questa è un'altra storia (o un altro articolo).

Ad un certo punto, il Makefile lancia `cc` o `gcc`. Questo in realtà è un preprocessore, un compilatore C (o C++), ed un linker, invocati in quell'ordine. Questo processo converte il sorgente nei binari, i veri eseguibili.

L'invocazione di *make* di solito richiede solo di battere **make**. Ciò, generalmente, serve a compilare tutti i file eseguibili necessari per il pacchetto in questione. Tuttavia, make può svolgere anche altri compiti, come installare i file nelle loro directory appropriate (**make install**) e rimuovere i file oggetto stantii (**make clean**). L'esecuzione di **make -n** permette di vedere un'anteprima del processo di compilazione, poiché stampa tutti i comandi che sarebbero attivati da un make, ma senza in effetti eseguirli.

Solo i software più semplici usano un Makefile generico. Le installazioni più complesse richiedono un Makefile su misura secondo la posizione delle librerie, dei file include e delle risorse sulla vostra specifica macchina. Questo, in particolare, è il caso che si ha quando durante il processo di compilazione servono le librerie `X11` per l'installazione. *Imake* e *xmkmf* adempiono questo compito.

Un `Imakefile` è, per citare la pagina di manuale, un prototipo di Makefile. L'utilità imake costruisce un Makefile adatto al vostro sistema dall'Imakefile. Nella maggior parte dei casi, tuttavia, preferirete eseguire **xmkmf**, uno script shell che invoca imake, un suo front end. Controllate il file README o INSTALL incluso nell'archivio per istruzioni specifiche (se, dopo aver estratto dall'archivio i file sorgenti, è presente un file `Imake` nella directory base, è un chiaro segno che dovrebbe essere eseguito **xmkmf**). Leggere le pagine di manuale di `Imake` e `xmkmf` per una più dettagliata analisi della procedura.

È bene essere consapevoli che `xmkmf` e `make` potrebbero dover essere invocati come root, specialmente nel fare un **make install** per spostare i binari sulle directory `/usr/bin` o `/usr/local/bin`. Usare make come utente normale, senza privilegi di root, porterà probabilmente a dei messaggi d'errore tipo *write access denied* (accesso in scrittura negato), perché manca il permesso per la scrittura nelle directory di sistema. Controllate anche che i binari creati abbiano i giusti permessi di esecuzione per voi ed ogni altro utente appropriato.

Quando viene invocato, **xmkmf** usa il file `Imake` per costruire il Makefile appropriato per il vostro sistema. Sarete soliti invocare **xmkmf** con l'argomento **-a**, per effettuare automaticamente *make Makefiles*, *make includes* e *make depend*. Ciò imposta le variabili e definisce le posizioni delle librerie per il compilatore ed il linker. A volte, non ci sarà il file `Imake`, ci sarà invece uno script `INSTALL` o `configure`, che dovrebbe essere invocato come **./configure** per assicurarsi che venga chiamato il giusto script `configure`. Nella maggior parte dei casi, il file `README` incluso con la distribuzione spiegherà la procedura di installazione.

È di solito una buona idea guardare dentro il `Makefile` che `xmkmf`, o uno degli script di installazione, costruiscono. Il Makefile sarà di solito adatto per il vostro sistema, ma occasionalmente potrebbe essere necessario ritoccarlo o correggere manualmente degli errori.

Per installare i binari appena compilati nelle appropriate directory di sistema di solito basta eseguire, come root, **make install**. Solitamente, le directory per i binari del sistema sulle moderne distribuzioni Linux sono `/usr/bin`, `/usr/X11R6/bin`, e `/usr/local/bin`. La directory da preferire per i nuovi pacchetti è `/usr/local/bin`, poiché in tal modo si terranno separati i binari che non fanno parte della installazione Linux originale.

I pacchetti originariamente mirati per versioni commerciali di UNIX potrebbero provare ad installarsi in `/opt` o in un'altra directory sconosciuta. Ciò, naturalmente, causerà un errore di installazione se la directory in questione non esiste. Il modo più semplice per risolvere questo problema è quello di creare, come root, una directory `/opt`, lasciare che il pacchetto vi si installi, e poi aggiungere tale directory alla variabile d'ambiente `PATH`. Oppure, potete creare dei link simbolici alla directory `/usr/local/bin`.

La procedura di installazione generale sarà quindi:

- Leggere il file `README` ed altri file di documentazione appropriati.
- Eseguire **xmkmf -a**, o lo script `INSTALL` o `configure`.
- Controllare il `Makefile`.
- Se necessario, eseguire **make clean**, **make Makefiles**, **make includes**, e **make depend**.
- Eseguire **make**.
- Controllare i permessi.
- Se necessario, eseguire **make install**.

Note:

- Normalmente non si compilano i pacchetti come root. L'effettuare un **su** a root è necessario solo per installare i binari compilati nelle directory di sistema.
- Una volta che avete preso confidenza con *make* e col suo uso, potreste volere che vengano aggiunte nel `Makefile` standard incluso nel (o creato col) pacchetto che state installando delle opzioni di ottimizzazione aggiuntive. Alcune di tali opzioni, le più usate, sono *-O2*, *-fomit-frame-pointer*, *-funroll-loops* e *-mpentium* (se usate un processore Pentium). Siate cauti e fatevi guidare dal buon senso quando modificate un `Makefile`!
- Dopo che *make* ha creato i binari, potreste voler usare **strip** su essi. Il comando **strip** elimina dai file binari le informazioni per il debugging simbolico e ne riduce la dimensione, spesso drasticamente. Ovviamente ciò disabiliterà il debugging.

- Il *Pack Distribution Project* `http://sunsite.auc.dk/pack/` utilizza un diverso approccio per la creazione di archivi di pacchetti software, basato su un insieme di strumenti scritti in Python per la gestione di link simbolici a file installati in *directory di raccolta* separate. Questi archivi sono degli ordinari *pacchetti tar*, ma si installano nelle directory `/coll` e `/pack`. Potreste dover scaricare il *Pack-Collection* dal suddetto sito qualora vi capiti di imbattervi in una di tali distribuzioni.

4 Binari preimpacchettati

4.1 Cosa c'è che non va negli rpm?

La compilazione e l'installazione manuale dei pacchetti dal sorgente è un compito apparentemente così spaventoso per alcuni utenti Linux che essi hanno abbracciato i popolari formati di pacchetti *rpm* e *deb*, o il più recente Stampede *slp*. Sebbene possa essere vero che l'installazione di un *rpm* di solito procede tanto facilmente e tanto velocemente quanto l'installazione del software di un certo altro noto sistema operativo, è il caso di spendere qualche parola riguardo gli svantaggi della installazione-fai-da-te dei binari preimpacchettati.

Primo, sappiate che i pacchetti software vengono di solito rilasciati inizialmente come pacchetti tar, e i binari preimpacchettati li seguono giorni, settimane, persino mesi dopo. Un pacchetto *rpm* corrente è tipicamente almeno un paio di versioni minori indietro rispetto all'ultimo pacchetto tar. Quindi, se desiderate stare al passo con tutto il software dell'ultima generazione, potreste non voler aspettare che appaia un *rpm* o un *deb*. Alcuni pacchetti meno popolari potrebbero non essere mai convertiti in *rpm*.

Secondo, il pacchetto tar potrebbe facilmente essere più completo, avere più opzioni, e prestarsi meglio ad una personalizzazione ed una messa a punto. La versione binaria rpm potrebbe non avere alcune delle funzionalità della versione completa. Gli *rpm* sorgenti contengono il codice sorgente completo e sono equivalenti ai corrispondenti pacchetti tar, e allo stesso modo necessitano di essere compilati ed installati usando l'opzione **rpm –recompile nomepacchetto.rpm** oppure **rpm –rebuild nomepacchetto.rpm**.

Terzo, alcuni binari preimpacchettati non si installano bene, e anche se si installano, potrebbero piantarsi e fare un core dump. Essi potrebbero dipendere da versioni di libreria diverse da quelle presenti nel vostro sistema, o potrebbero essere stati preparati impropriamente o essere semplicemente difettosi. Ad ogni modo, quando installate un *rpm* o un *deb* necessariamente fate affidamento sulla competenza delle persone che hanno preparato quel pacchetto.

Infine, aiuta avere il codice sorgente in mano, per poter effettuare delle riparazioni ed imparare da esso. È molto più conveniente avere il sorgente nell'archivio da cui si stanno compilando i binari, piuttosto che in un differente pacchetto *rpm*.

L'installazione di un pacchetto *rpm* non è necessariamente una bazzecola. Se c'è un conflitto di dipendenza, l'installazione dell'*rpm* fallirà. L'*rpm* potrebbe richiedere una versione delle librerie diversa da quelle presenti sul vostro sistema, l'installazione potrebbe non funzionare, anche se create dei link simbolici alle librerie mancanti da quelle a posto. Malgrado la loro convenienza, le installazioni degli *rpm* spesso falliscono per le stesse ragioni per cui lo fanno quelle dei pacchetti tar.

Dovete installare gli *rpm* e i *deb* come root, per avere i necessari permessi di scrittura, e ciò apre un buco di sicurezza potenzialmente serio, poiché potreste inavvertitamente massacrare i binari di sistema e le librerie, o anche installare un *cavallo di Troia* che potrebbe liberare il caos sul vostro sistema. È quindi importante ottenere pacchetti *rpm* e *deb* da una fonte fidata. In ogni caso, dovreste eseguire una 'verifica della firma' (rispetto ad un codice di controllo MD5) sul pacchetto, **rpm –checksig nomepacchetto.rpm**, prima di installarlo. Allo stesso modo è fortemente raccomandata l'esecuzione di **rpm -K –nopgp nomepacchetto.rpm**. I comandi corrispondenti per i pacchetti *deb* sono **dpkg -I | –info nomepacchetto.deb** e **dpkg -e | –control nomepacchetto.deb**.

- `rpm -checksig gnucash-1.1.23-4.i386.rpm`

```
gnucash-1.1.23-4.i386.rpm:  size md5 OK
```

- `rpm -K -nopgp gnucash-1.1.23-4.i386.rpm`

```
gnucash-1.1.23-4.i386.rpm:  size md5 OK
```

Per i tipi veramente paranoici (e in questo caso ci sarebbe molto da dire a proposito di paranoia), ci sono le utilità *unrpm* e *rpmunpack* disponibili presso la *directory utils/package di Sunsite* per estrarre e controllare i singoli componenti dei pacchetti.

Klee Diene `mailto:klee@debian.org` ha scritto il pacchetto sperimentale *dpkgcert*, per la verifica dell'integrità dei file *.deb* installati, usando i codici di controllo MD5. È disponibile nell' *archivio ftp Debian* `ftp://ftp.debian.org/pub/debian/project/experimental` . L'attuale nome / versione è *dpkgcert_ 0.2-4.1_ all.deb*. Il sito *Jim Pick Software* `http://dpkgcert.jimpick.com` mantiene un server database sperimentale per fornire certificati *dpkgcert* per i pacchetti di una tipica installazione Debian.

Nella loro forma più semplice, i comandi **rpm -i nomepacchetto.rpm** e **dpkg –install nomepacchetto.deb** automaticamente aprono il pacchetto ed installano il software. Siate cauti, comunque, poiché usare tali comandi ciecamente può essere pericoloso per la salute del vostro sistema!

Notate che gli avvertimenti suddetti si applicano anche, sebbene in minor misura, all'utilità di installazione *pkgtool* della Slackware. Tutto il software di installazione automatico richiede cautela.

I programmi *martian* `http://www.people.cornell.edu/pages/rc42/program/martian.html` e *alien* `http://kitenet.net/programs/alien/` permettono la conversione tra i formati dei pacchetti *rpm*, *deb*, Stampede *slp* e *tar.gz*. Ciò rende questi pacchetti accessibili a tutte le distribuzioni Linux.

Leggere attentamente le pagine di manuale dei comandi *rpm* e *dpkg*, e fare riferimento all' *RPM HOWTO* , alla

Quick Guide to Red Hat's Package Manager `http://www.tfug.org/helpdesk/linux/rpm.html` del TFUG, e a *The Debian Package Management Tools* `http://www.debian.org/doc/FAQ/debian-faq-7.html` per informazioni più dettagliate.

4.2 Problemi con gli rpm: un esempio

Jan Hubicka `mailto:hubicka@paru.cas.cz` ha scritto un bellissimo pacchetto per i frattali, chiamato *xaos*. Sulla sua

home page `http://www.paru.cas.cz/~hubicka/XaoS` sono disponibili entrambi i pacchetti `.tar.gz` e `rpm`. In nome della comodità proviamo la versione rpm, piuttosto che il pacchetto tar.

Sfortunatamente, l'rpm di *xaos* non si installa. Due diverse versioni rpm fanno i capricci.

rpm -i –test XaoS-3.0-1.i386.rpm

```
error: failed dependencies:
        libslang.so.0 is needed by XaoS-3.0-1
        libpng.so.0 is needed by XaoS-3.0-1
        libaa.so.1 is needed by XaoS-3.0-1
```

rpm -i –test xaos-3.0-8.i386.rpm

```
error: failed dependencies:
        libaa.so.1 is needed by xaos-3.0-8
```

La cosa strana è che `libslang.so.0`, `libpng.so.0`, e `libaa.so.1` sono tutte presenti nella directory `/usr/lib` del sistema usato. Gli rpm di *xaos* devono essere stati compilati con delle versioni leggermente diverse di quelle librerie, anche se i numeri di versione sono identici.

Come test, proviamo ad installare `xaos-3.0-8.i386.rpm` con l'opzione *–nodeps* per forzarne l'installazione. Provando ad eseguire *xaos* si pianta.

```
xaos: error in loading shared libraries: xaos: undefined symbol: __fabsl
```

(errore nel caricamento delle librerie condivise, il simbolo __fabsl non è definito)

Cerchiamo testardamente di andare in fondo alla cosa. Lanciando *ldd* sul binario di *xaos*, per trovare da quali librerie dipende, vediamo che le librerie necessarie ci sono tutte. Lanciando *nm* sulla libreria `/usr/lib/libaa.so.1`, per vedere i suoi riferimenti simbolici, ci accorgiamo che *__fabsl* manca davvero. Naturalmente il riferimento che manca *potrebbe* non essere presente in una qualsiasi delle altre librerie... Non c'è niente da fare, salvo rimpiazzare una o più librerie.

Basta! Scarichiamo il pacchetto tar, `XaoS-3.0.tar.gz`, disponibile sul *sito ftp* `ftp://ftp.ta.jcu.cz/pub/linux/hubicka/XaoS/3.0` o reperibile dalla home page. Proviamo a compilarlo. L'esecuzione di **./configure**, **make** e infine (come root) **make install** procede senza intoppi.

Questo è solo uno fra i tanti esempi di binari preimpacchettati che portano più problemi che vantaggi.

5 Problemi riguardo termcap e terminfo

Secondo la pagina di manuale, *terminfo è un database che descrive i terminali, usato da programmi orientati-allo-schermo...*. Esso definisce un generico insieme di sequenze di controllo (codici di escape) usati per mostrare il testo sui terminali, e rende possibile il supporto per differenti terminali hardware senza la necessità di driver speciali. Le librerie *terminfo* si trovano in `/usr/share/terminfo`, sulle moderne distribuzioni di Linux.

Il database *terminfo* ha ampiamente sostituito il più vecchio *termcap* ed il completamente obsoleto *termlib*. Ciò normalmente non ha nessuna attinenza con l'installazione dei programmi, eccetto quando si ha a che fare con un pacchetto che richiede *termcap*.

La maggior parte delle distribuzioni Linux ora usano *terminfo*, ma ancora conservano le librerie *termcap* per compatibilità con le applicazioni legacy (vedere `/etc/termcap`). A volte c'è uno speciale pacchetto di compatibilità che è necessario aver installato per facilitare l'uso dei binari linkati con termcap. Raramente, potrebbe essere necessario togliere il commento da una dichiarazione *#define termcap* in un file sorgente. Controllate i file di documentazione appropriati nella vostra distribuzione per informazioni a tal riguardo.

6 Compatibilità all'indietro con i binari a.out

In rarissimi casi, è necessario usare binari a.out, o perché il codice sorgente non è disponibile o perché, per una qualche ragione, non è possibile compilare nuovi binari ELF dal sorgente.

Come succede, le installazioni ELF hanno quasi sempre un completo insieme di librerie a.out nella directory `/usr/i486-linuxaout/lib`. Lo schema di numerazione delle librerie a.out differisce da quello delle ELF, evitando intelligentemente conflitti che potrebbero creare confusione. I binari a.out dovrebbero perciò essere in grado di trovare le giuste librerie in fase di esecuzione, ma ciò potrebbe non accadere sempre.

Notate che il kernel necessita di avere il supporto per a.out, o direttamente o come modulo caricabile. Potrebbe essere necessario ricompilare il kernel per abilitare ciò. Inoltre, alcune distribuzioni di Linux richiedono l'installazione di uno speciale pacchetto di compatibilità, come `xcompat` di Debian, per eseguire applicazioni X a.out.

6.1 Un esempio

Jerry Smith ha scritto il comodissimo programma *xrolodex* alcuni anni fa. Esso usa le librerie Motif, ma fortunatamente è disponibile come binario linkato staticamente in formato a.out. Sfortunatamente, il sorgente necessita di numerosi aggiustamenti per essere ricompilato usando le librerie *lesstif*. Ancor più sfortunatamente, il binario a.out su di un sistema ELF va in bomba con il seguente messaggio d'errore.

```
xrolodex: can't load library '//lib/libX11.so.3'
No such library
```

(Traducendo: non è possibile caricare la libreria //lib/libX11.so.3; non c'è nessuna libreria con quel nome)

Si dà il caso che ci sia una tale libreria, in `/usr/i486-linuxaout/lib`, ma xrolodex è incapace di trovarla in fase di esecuzione. La soluzione semplice è di fornire un link simbolico nella directory `/lib`:

```
ln -s /usr/i486-linuxaout/lib/X11.so.3.1.0 libX11.so.3
```

Ne viene fuori che è necessario fornire link simili per le librerie libXt.so.3 e libc.so.4. Ciò deve essere fatto come root, naturalmente. Notate che dovrete essere assolutamente certi di non sovrascrivere o provocare conflitti di versione con librerie preesistenti. Fortunatamente, le nuove librerie ELF hanno numeri di versione più alti delle più vecchie a.out, per prevenire ed impedire proprio tali problemi.

Dopo aver creato i tre link, *xrolodex* funziona bene.

Il pacchetto *xrolodex* era originariamente pubblicato su *Spectro* `http://www.spectro.com/` , ma sembra che sia sparito da lì. Attualmente può essere scaricato da *Sunsite* `http://metalab.unc.edu/pub/Linux/apps/reminder/xrolodex.tar.z` come file sorgente [512k] in formato *tar.Z*.

7 Risoluzione dei problemi

Se *xmkmf* e/o *make* hanno funzionato senza problemi, potete passare alla 8 (prossima sezione). Tuttavia, nella vita reale, poche cose vanno bene al primo tentativo. È in questi casi che la vostra intraprendenza viene messa alla prova.

7.1 Errori in fase di link

- Supponiamo che *make* fallisca con un: `Link error: -lX11: No such file or directory` (Nessun file o directory con quel nome), anche dopo che xmkmf è stato invocato. Ciò potrebbe significare che il file *Imake* non è stato preparato correttamente. Controllate che nella prima parte del *Makefile* ci siano delle righe tipo:

  ```
  LIB=          -L/usr/X11/lib
  INCLUDE=      -I/usr/X11/include/X11
  LIBS=         -lX11 -lc -lm
  ```

  Le opzioni `-L` e `-I` dicono al compilatore e al linker dove cercare i file *library* e *include*, rispettivamente. In questo esempio, le *librerie di X11* dovrebbero essere nella directory `/usr/X11/lib`, e i *file include di X11* dovrebbero essere nella directory `/usr/X11/include/X11`. Se sulla vostra macchina non è così, apportate i cambiamenti necessari al *Makefile* e riprovate il *make*.

- Riferimenti non definiti alle funzioni della libreria matematica, come il seguente:

  ```
  /tmp/cca011551.o(.text+0x11): undefined reference to 'cos'
  ```

  La soluzione è di linkargli esplicitamente la `libreria matematica`, aggiungendo un **-lm** al flag *LIB* o *LIBS* nel `Makefile` (vedere esempio precedente).

- Ancora un'altra cosa da provare se *xmkmf* fallisce è lo script seguente:

  ```
  make -DUseInstalled -I/usr/X386/lib/X11/config
  ```

  Che è una specie di *xmkmf* ridotto all'osso.

- In rarissimi casi, l'esecuzione di *ldconfig* come *root* potrebbe essere la soluzione:

  # **ldconfig** aggiorna i link simbolici alla libreria condivisa. *Questo potrebbe non essere necessario.*

- Alcuni `Makefile` usano degli alias non riconosciuti per le librerie presenti nel vostro sistema. Per esempio, il binario potrebbe richiedere `libX11.so.6`, ma in `/usr/X11R6/lib` non c'è nessun file o link con quel nome. Però, c'è un `libX11.so.6.1`. La soluzione è di fare un **ln -s /usr/X11R6/lib/libX11.so.6.1 /usr/X11R6/lib/libX11.so.6**, come root. Ciò potrebbe dover essere seguito da un **ldconfig**.

- A volte il sorgente necessita delle vecchie librerie nella versione X11R5 per essere compilato. Se avete le librerie R5 in /usr/X11R6/lib (avete avuto la possibilità di installarle durante la prima installazione di Linux), allora dovete solo assicurarvi di avere i link di cui il software ha bisogno per la compilazione. Le `librerie R5` sono chiamate `libX11.so.3.1.0`, `libXaw.so.3.1.0`, e `libXt.so.3.1.0`. Di solito vi servono dei link, come *libX11.so.3 -> libX11.so.3.1.0*. Forse il software avrà bisogno anche di un link del tipo *libX11.so -> libX11.so.3.1.0*. Naturalmente, per creare un link mancante, usate il comando **ln -s libX11.so.3.1.0 libX11.so**, *come root*.

- Alcuni pacchetti esigeranno l'installazione di versioni aggiornate di una o più librerie. Per esempio, le versioni 4.x della suite *StarOffice* della StarDivision GmbH erano famose per richiedere `libc` in versione 5.4.4 o successiva. Anche il più recente *StarOffice* 5.0 non girerà nemmeno dopo l'installazione con le nuove librerie `glibc 2.1`. Fortunatamente, il più nuovo *StarOffice* 5.1 risolve tali problemi. Se avete una versione di *StarOffice* più vecchia, potreste dover copiare, da root, una o più librerie nelle directory appropriate, rimuovere le vecchie librerie, poi ripristinare i link simbolici (controllate l'ultima versione dello `StarOffice miniHOWTO` 16 ((tradotto)) per maggiori informazioni su questo argomento).

  Attenzione: Usate molta cautela nel fare ciò, poiché potreste rendere non funzionante il vostro sistema se combinate dei pasticci.

  Potete trovare le librerie più aggiornate presso *Sunsite* .

7.2 Altri problemi

- Uno script *Perl* o shell installato vi dà un `No such file or directory` come messaggio d'errore. In questo caso, controllate i permessi del file per assicurarvi che il file sia eseguibile e controllate l'intestazione del file per accertarvi che la shell o il programma invocato dallo script sia nel posto specificato. Per esempio, lo script potrebbe iniziare con:

  ```
  #!/usr/local/bin/perl
  ```

Se infatti *Perl* è installato nella vostra directory `/usr/bin` invece che nella `/usr/local/bin`, allora lo script non funzionerà. Ci sono due modi per correggere questo problema. L'intestazione del file script può essere cambiata in `#!/usr/bin/perl`, o si può aggiungere un link simbolico alla giusta directory, **ln -s /usr/bin/perl /usr/local/bin/perl**.

- Alcuni programmi X11 richiedono le librerie Motif per la compilazione. Le distribuzioni Linux standard non hanno le librerie Motif installate, e al momento Motif costa 100-200$ extra (sebbene in parecchi casi funzioni anche la versione freeware *Lesstif* `http://www.lesstif.org/`). Se vi serve Motif per la compilazione di un certo pacchetto, ma vi mancano le librerie Motif, può essere possibile ottenere dei *binari linkati staticamente*. Il linkaggio statico incorpora le routine di libreria nei binari stessi. Ciò si traduce in file binari più grandi, ma il codice girerà sui sistemi in cui mancano le librerie.

Quando un pacchetto richiede, per la compilazione, delle librerie non presenti sul vostro sistema, ciò provocherà errori in fase di link (errori tipo `undefined reference` - riferimento non definito). Le librerie potrebbero essere del tipo costoso (proprietà di qualcuno) o difficili da trovare per qualche altra ragione. In tal caso, ottenere un binario *linkato staticamente* dall'autore del pacchetto, o da un gruppo utenti Linux, può essere il modo più facile per effettuare delle riparazioni.

- Eseguendo uno script *configure* è stato creato uno strano Makefile, uno che sembra non avere nulla a che fare col pacchetto che state tentando di compilare. Ciò significa che è stato eseguito il *configure* sbagliato, uno trovato da qualche altra parte nel path. Lanciate sempre *configure* come **./configure** per evitare questo problema.

- La maggior parte delle distribuzioni sono passate alle librerie `libc 6 / glibc 2` dalla più vecchia `libc 5`. I binari precompilati che funzionavano con la vecchia libreria potrebbero andare in bomba se avete aggiornato la libreria. La soluzione è o di ricompilare le applicazioni dal sorgente o di ottenere dei nuovi binari precompilati. Se state aggiornando il vostro sistema a `libc 6` e riscontrate dei problemi, fate riferimento al *Glibc 2 HOWTO* 16 ((tradotto)) di Eric Green.

Notate che ci sono delle piccole incompatibilità fra le versioni minori di `glibc`, così un binario compilato con `glibc 2.1` potrebbe non funzionare con `glibc 2.0` e vice versa.

- A volte è necessario togliere l'opzione *-ansi* dai flag di compilazione nel `Makefile`. Ciò abilita le caratteristiche supplementari di gcc, quelle non-ANSI in particolare, e permette la compilazione di pacchetti che richiedono tali estensioni. (Grazie a Sebastien Blondeel per questa indicazione).

- Alcuni programmi esigono di essere *setuid root*, per poter essere eseguiti con *privilegi di root*. Il comando per effettuare ciò è **chmod u+s nomefile**, *come root* (osservate che il programma deve già essere di proprietà di root). Questo ha l'effetto di impostare il bit *setuid* nei permessi del file. Questo problema viene fuori quando il programma accede all'hardware di sistema, come un modem o un lettore CD ROM, o quando le librerie SVGA vengono chiamate dal modo console, come in un particolare noto pacchetto di emulazione. Se un programma funziona quando eseguito da root, ma dà messaggi di errore tipo *access denied* (accesso negato) ad un utente normale, sospettate che la causa sia questa.

  Avvertimento: Un programma con *setuid* impostato come root può porre un rischio di sicurezza per il sistema. Il programma gira con privilegi di root ed ha così il potenziale di causare danni significativi. Accertatevi di sapere cosa fa il programma, guardando il sorgente se possibile, prima di impostare il bit *setuid*.

7.3 Ritocchi e messa a punto

Potreste voler esaminare il `Makefile` per accertarvi che vengano usate le migliori opzioni di compilazione possibili per il vostro sistema. Per esempio, impostando il flag *-O2* si sceglie il più alto livello di ottimizzazione ed il flag *-fomit-frame-pointer* provoca la generazione di un binario più piccolo (sebbene il debugging sarà così disabilitato). **Però non giocherellate con tali opzioni, a meno che non sappiate cosa state facendo, e comunque non prima di aver ottenuto un binario funzionante.**

7.4 Dove trovare maggiore aiuto

Nella mia esperienza, forse il 25% delle applicazioni supera la fase di compilazione così com'è, senza problemi. Un altro 50%, o giù di lì, può essere persuaso a farlo con uno sforzo variabile da lieve ad erculeo. Questo significa che ancora un numero significativo di pacchetti non ce la faranno, non importa cosa si faccia. In tal caso, i binari Intel `ELF` e/o `a.out` di questi potrebbero essere trovati presso *Sunsite* o presso *TSX-11 archive* . *Red Hat* `http://redhat.com` e

Debian `http://www.debian.org` hanno vasti archivi di binari preimpacchettati della maggior parte dei più popolari software per Linux. Forse l'autore del software può fornire i binari compilati per il vostro particolare tipo di macchina.

`Notate che se ottenete i binari precompilati, dovrete controllarne la compatibilità con il vostro sistema:`

- `I binari devono girare sul vostro hardware (i.e., Intel x86).`
- `I binari devono essere compatibili con il vostro kernel (i.e., a.out o ELF).`
- `Le vostre librerie devono essere aggiornate.`
- `Il vostro sistema deve avere le appropriate utilità di installazione (rpm o deb).`

Se tutto il resto non funziona, potete trovare aiuto nei newsgroup appropriati, come *comp.os.linux.x* o *comp.os.linux.development* .

Se non funziona proprio niente, almeno avrete fatto del vostro meglio, ed avrete imparato molto.

8 Conclusioni

Leggete la documentazione del pacchetto software per stabilire se certe variabili d'ambiente hanno bisogno di impostazioni (in `.bashrc` o `.cshrc`) e se i file `.Xdefaults` e `.Xresources` hanno bisogno di essere personalizzati.

Potrebbe esserci un file di default per l'applicazione, di solito chiamato `Xprog.ad` nella distribuzione Xprog originale. Se è così, editate il file Xprog.ad per adattarlo alla vostra macchina, poi cambiategli nome (**mv**) in Xprog ed installatelo nella directory `/usr/lib/X11/app-defaults` *come root*. Un insuccesso nel fare ciò potrebbe far sì che il software si comporti stranamente o addirittura si rifiuti di girare.

La maggior parte dei pacchetti software comprendono una o più pagine di manuale preformattate. *Come root*, copiate il file Xprog.man nella directory `/usr/man`, `/usr/local/man`, o `/usr/X11R6/man` appropriata (`man1` - `man9`), e cambiategli nome come del caso. Per esempio, se Xprog.man finisce in /usr/man/man4, dovrebbe essere rinominato Xprog.4 (mv Xprog.man Xprog.4). Per convenzione, i comandi utente vanno in `man1`, i giochi in `man6`, ed i pacchetti di amministrazione in `man8` (vedere la *documentazione di man* per maggiori dettagli). Naturalmente, se vi va, potete discostarvi dalla convenzione, sul vostro sistema.

Alcuni, pochi, pacchetti non installeranno i binari nelle appropriate directory di sistema, cioè essi non hanno l'opzione *install* nel `Makefile`. In tal caso, potete installare manualmente i binari copiandoli nell'appropriata directory di sistema, `/usr/bin`, `/usr/local/bin` o `/usr/X11R6/bin`, *come root*, naturalmente. Notate che `/usr/local/bin` è la directory da preferire per i binari che non fanno parte della distribuzione di base di Linux.

Alcune o tutte le suddette procedure ,nella maggior parte dei casi, dovrebbero essere effettuate automaticamente con un **make install**, e forse un **make install.man** o un **make install_man**. Se è così, i file di documentazione `README` o `INSTALL` lo specificheranno.

9 Primo esempio: Xscrabble

L'`Xscrabble` di Matt Chapman aveva l'aria di essere un programma che sarebbe stato interessante avere, poiché si dà il caso che io sia un accanito giocatore di ScrabbleTM. Lo scaricai, decompressi, e lo compilai seguendo la procedura nel file README:

```
xmkmf
make Makefiles
make includes
make
```

Ovviamente non funzionò...

```
gcc -o xscrab -O2 -O -L/usr/X11R6/lib
init.o xinit.o misc.o moves.o cmove.o main.o xutils.o mess.o popup.o
widgets.o display.o user.o CircPerc.o
-lXaw -lXmu -lXExExt -lXext -lX11 -lXt -lSM -lICE -lXExExt -lXext -lX11
-lXpm -L../Xc -lXc

BarGraf.o(.text+0xe7): undefined reference to `XtAddConverter'
BarGraf.o(.text+0x29a): undefined reference to `XSetClipMask'
BarGraf.o(.text+0x2ff): undefined reference to `XSetClipRectangles'
BarGraf.o(.text+0x375): undefined reference to `XDrawString'
BarGraf.o(.text+0x3e7): undefined reference to `XDrawLine'
etc.
etc.
etc...
```

Indagai su ciò nel newsgroup *comp.os.linux.x* , e qualcuno gentilmente mi indicò che, apparentemente, le librerie Xt, Xaw, Xmu, e X11 non erano state trovate nella fase di link. Hmmm...

C'erano due Makefile principali, e quello nella directory `src` catturò la mia attenzione. Una linea nel Makefile definita LOCAL_LIBS: LOCAL_LIBS = $(XAWLIB) $(XMULIB) $(XTOOLLIB) $(XLIB). Qui c'erano i riferimenti alle librerie non trovate dal linker.

Cercando il successivo riferimento a LOCAL_LIBS, vidi alla linea 495 di quel Makefile:

```
      $(CCLINK) -o $@ $(LDOPTIONS) $(OBJS) $(LOCAL_LIBS) $(LDLIBS)
$(EXTRA_LOAD_FLAGS)
```

Ora, cos'erano queste LDLIBS?

```
      LDLIBS = $(LDPOSTLIB) $(THREADS_LIBS) $(SYS_LIBRARIES)
$(EXTRA_LIBRARIES)
```

Le SYS_LIBRARIES erano:

```
SYS_LIBRARIES = -lXpm -L../Xc -lXc
```

Sì! Le librerie mancanti erano qui.

È possibile che il linker avesse bisogno di vedere le LDLIBS prima delle LOCAL_LIBS... Così, la prima cosa da provare era di modificare il Makefile invertendo le $(LOCAL_LIBS) e le $(LDLIBS) alla linea 495, dunque ora si dovrebbe leggere:

```
        $(CCLINK) -o $@ $(LDOPTIONS) $(OBJS) $(LDLIBS) $(LOCAL_LIBS)
$(EXTRA_LOAD_FLAGS)                          ^^^^^^^^^^^^^^^^^^^^^^
```

Provai ad eseguire di nuovo *make* con i suddetti cambiamenti e, guarda un po', stavolta funzionò. Xscrabble aveva ancora bisogno di qualche aggiustamento ed una messa a punto, ovviamente, come cambiare nome al dizionario e togliere il commento da qualche statement assert in uno dei file sorgenti, ma da allora mi ha fornito svariate ore di divertimento.

[Notate che ora è disponibile una nuova versione di Xscrabble in formato rpm, e questa si installa senza problemi.]

Potete contattare *Matt Chapman* `mailto:matt@belgarath.demon.co.uk` via e-mail, e scaricare *Xscrabble* dalla sua *home page* `http://www.belgarath.demon.co.uk/programs/index.html` .

```
Scrabble e un marchio registrato dalla Milton Bradley Co., Inc.
```

10 Secondo esempio: Xloadimage

Questo esempio pone un problema più facile. Il programma *xloadimage* sembrava un'utile aggiunta alla mia raccolta di attrezzi grafici. Ho copiato il file `xloadi41.gz` direttamente dalla directory sorgente sul CD, allegato all'eccellente libro 16 (X User Tools), di Mui e Quercia. Come c'era da aspettarsi, *tar xzvf* estrae i file dall'archivio. Il *make*, però, fornisce un antipatico errore e termina.

```
gcc -c -O -fstrength-reduce -finline-functions -fforce-mem
-fforce-addr -DSYSV  -I/usr/X11R6/include
-DSYSPATHFILE=\"/usr/lib/X11/Xloadimage\" mcidas.c

In file included from /usr/include/stdlib.h:32,
                 from image.h:23,
                 from xloadimage.h:15,
                 from mcidas.c:7:
/usr/lib/gcc-lib/i486-linux/2.6.3/include/stddef.h:215:
conflicting types for `wchar_t'
/usr/X11R6/include/X11/Xlib.h:74: previous declaration of
`wchar_t'
make[1]: *** [mcidas.o] Error 1
make[1]: Leaving directory
`/home/thegrendel/tst/xloadimage.4.1'
make: *** [default] Error 2
```

Il messaggio d'errore contiene l'indizio essenziale.

Guardando il file `image.h`, linea 23...

```
#include <stdlib.h>
```

Aha, da qualche parte nel sorgente per *xloadimage*, *wchar_t* è stato ridefinito in modo diverso da quanto specificato nel file include standard, `stdlib.h`. Proviamo prima a commentare la linea 23 in `image.h`, che forse l'*include stdlib.h*, dopo tutto, non è necessario.

A questo punto, la fase di compilazione procede senza nessun errore fatale. Il pacchetto *xloadimage* funziona correttamente ora.

11 Terzo esempio: Fortune

Questo esempio richiede qualche conoscenza di programmazione in C. La maggioranza del software UNIX/Linux è scritta in C, e imparare almeno un po' di C sarebbe certamente un bene per chiunque sia seriamente interessato all'installazione del software.

Il famoso programma *fortune* mostra una frase umoristica, un biscotto della fortuna, ad ogni avvio di Linux. Sfortunatamente (il gioco di parole è intenzionale), provare a costruir fortuna su una distribuzione Red Hat con un kernel 2.0.30 provoca degli errori fatali. (N.d.T: in inglese build fortune significa sia far fortuna che compilare il programma fortune)

```
~/fortune# make all


gcc -O2 -Wall -fomit-frame-pointer -pipe   -c fortune.c -o
fortune.o
fortune.c: In function ‘add_dir’:
fortune.c:551: structure has no member named ‘d_namlen’
fortune.c:553: structure has no member named ‘d_namlen’
make[1]: *** [fortune.o] Error 1
make[1]: Leaving directory ‘/home/thegrendel/for/fortune/fortune’
make: *** [fortune-bin] Error 2
```

Guardando `fortune.c`, le linee pertinenti sono queste.

```
   if (dirent->d_namlen == 0)
            continue;
        name = copy(dirent->d_name, dirent->d_namlen);
```

Ci serve di trovare la struttura `dirent`, ma essa non è dichiarata nel file *fortune.c*, e nemmeno un **grep dirent** la mostra in nessuno dei file sorgenti. Tuttavia, all'inizio di *fortune.c*, c'è la seguente linea.

```
#include <dirent.h>
```

Questo sembra essere un file include per la libreria di sistema, perciò, il posto più logico dove cercare *dirent.h* è in */usr/include*. Effettivamente esiste un file *dirent.h* in */usr/include*, ma quel file non contiene la dichiarazione della struttura `dirent`. C'è, però, un riferimento ad un altro file *dirent.h*.

```
#include <linux/dirent.h>
```

Alla fine, andando in */usr/include/linux/dirent.h*, troviamo la dichiarazione della struttura che ci serve.

```
struct dirent {
        long            d_ino;
        __kernel_off_t  d_off;
        unsigned short  d_reclen;
        char            d_name[256]; /* We must not include
limits.h! */
};
```

Poco ma sicuro, la dichiarazione della struttura non contiene nessun *d_namelen*, ma ci sono un paio di candidati come suo equivalente. Il più probabile di essi è *d_reclen*, poiché questo membro della struttura probabilmente rappresenta la lunghezza di qualcosa ed è uno short integer. L'altra possibilità, *d_ino*, potrebbe essere un numero di inode, a giudicare dal suo nome e tipo. A quanto pare, stiamo probabilmente trattando con una struttura di registrazione delle directory, e questi elementi rappresentano gli attributi di un file, il suo nome, il suo inode, e la sua lunghezza (in blocchi). Ciò sembrerebbe convalidare la nostra scelta.

Editiamo il file `fortune.c` e cambiamo i due riferimenti alle linee 551 e 553 da *d_namelen* a *d_reclen*. Proviamo di nuovo un *make all*. **Successo.** La compilazione finisce senza errori. Ora possiamo prenderci i nostri brividi a buon mercato da fortune.

12 Quarto esempio: Hearts

Ecco il vecchio canuto gioco Hearts, scritto per i sistemi UNIX da Bob Ankeney negli anni '80, rivisto nel 1992 da Mike Yang, ed attualmente mantenuto da *Jonathan Badger* `mailto:badger@phylo.life.uiuc.edu` . Il suo predecessore era un ancor più vecchio programma Pascal di Don Backus della Oregon Software, poi aggiornato da Jeff Hemmerling. Originariamente pensato come client per più giocatori, funziona bene anche per un solo giocatore contro avversari gestiti dal computer. La grafica è bella, benché il gioco manchi di caratteristiche più sofisticate e i giocatori computerizzati non siano molto forti. Nonostante tutto, sembra essere il solo Hearts decente disponibile per macchine UNIX e Linux ancora oggi.

A causa della sua stirpe ed età, questo pacchetto è particolarmente difficile da compilare su di un sistema Linux. Richiede la soluzione di una lunga e sconcertante serie di puzzle. È un esercizio di pazienza e determinazione.

Prima di iniziare, accertatevi di avere le librerie `motif` o le `lesstif` installate.

-

xmkmf

make

```
client.c: In function 'read_card':
client.c:430: '_tty' undeclared (first use in this function)
client.c:430: (Each undeclared identifier is reported only once
client.c:430: for each function it appears in.)
client.c: In function 'scan':
client.c:685: '_tty' undeclared (first use in this function)
make: *** [client.o] Error 1
```

Questi sono i colpevoli nel file `client.c`:

```
#ifndef SYSV
        (buf[2] != _tty.sg_erase) && (buf[2] != _tty.sg_kill)) {
 #else
        (buf[2] != CERASE) && (buf[2] != CKILL)) {
#endif
```

•

In `client.c`, aggiungiamo

```
#define SYSV
```

alla linea 39. Ciò bypasserà il riferimento a *_tty*.

make

```
client.c:41: sys/termio.h: No such file or directory
make: *** [client.o] Error 1
```

•

Il file include `termio.h` è nella directory `/usr/include` su di un sistema Linux, invece che nella `/usr/include/sys`, come era sulle vecchie macchine UNIX. Perciò, cambiamo la linea 41 di `client.c` da

```
#include <sys/termio.h>
```

a

```
#include <termio.h>
```

make

```
gcc -o hearts -g      -L/usr/X11R6/lib client.o hearts.o select.o connect.o
sockio.o start_dist.o  -lcurses -ltermlib
/usr/bin/ld: cannot open -ltermlib: No such file or directory
collect2: ld returned 1 exit status
make: *** [hearts] Error 1
```

•

Le moderne distribuzioni Linux usano il database *terminfo* e/o *termcap*, piuttosto che l'obsoleto *termlib*.

Editiamo il `Makefile`.

Linea 655:

```
CURSES_LIBRARIES = -lcurses -ltermlib
```

cambiamola in:

```
CURSES_LIBRARIES = -lcurses -ltermcap
```

make

```
gcc -o xmhearts -g      -L/usr/X11R6/lib xmclient.o hearts.o select.o
connect.o sockio.o start_dist.o gfx.o  -lXm_s -lXt -lSM -lICE -lXext -lX11
-lPW
/usr/bin/ld: cannot open -lXm_s: No such file or directory
collect2: ld returned 1 exit status
```

•

La principale libreria *lesstif* è `libXm`, piuttosto che `libXm_s`. Quindi, editiamo il `Makefile`.

Nella linea 653:

```
XMLIB = -lXm_s $(XTOOLLIB) $(XLIB) -lPW
```

cambia in:

```
XMLIB = -lXm $(XTOOLLIB) $(XLIB) -lPW
```

make

```
gcc -o xmhearts -g      -L/usr/X11R6/lib xmclient.o hearts.o select.o
connect.o sockio.o start_dist.o gfx.o  -lXm -lXt -lSM -lICE -lXext -lX11 -lPW
/usr/bin/ld: cannot open -lPW: No such file or directory
collect2: ld returned 1 exit status
make: *** [xmhearts] Error 1
```

•

Raduniamo i soliti sospetti.

Non c'è nessuna libreria `PW`. Editiamo il `Makefile`.

Linea 653:

```
XMLIB = -lXm $(XTOOLLIB) $(XLIB) -lPW
```

cambia in:

```
XMLIB = -lXm $(XTOOLLIB) $(XLIB) -lPEX5
```

(La libreria `PEX5` è quella più simile alla `PW`.)

make

```
rm -f xmhearts
gcc -o xmhearts -g      -L/usr/X11R6/lib xmclient.o hearts.o select.o
connect.o sockio.o start_dist.o gfx.o  -lXm -lXt -lSM -lICE -lXext -lX11 -lPEX5
```

Il `make` finalmente funziona (hurra!)

•

Installazione:

Come root,

```
[root@localhost hearts]# make install
install -c -s  hearts /usr/X11R6/bin/hearts
install -c -s  xmhearts /usr/X11R6/bin/xmhearts
install -c -s  xawhearts /usr/X11R6/bin/xawhearts
install in . done
```

-

Esecuzione di prova:

rehash

(Stiamo eseguendo la shell `tcsh`.)

xmhearts

```
localhost:~/% xmhearts
Can't invoke distributor!
```

-

Dal file `README` nel pacchetto `hearts`:

```
Put heartsd, hearts_dist, and hearts.instr in the HEARTSLIB
directory defined in local.h and make them world-accessible.
```

(Traducendo: Mettete Heartsd, hearts_dist, e hearts.instr nella directory HEARTSLIB definita in local.h e rendeteli accessibili a chiunque)

Dal file `local.h`:

```
/* where the distributor, dealer and instructions live */

#define HEARTSLIB "/usr/local/lib/hearts"
```

Questo è un classico caso: RTFM (leggete il fottuto manuale).

Come *root*,

cd /usr/local/lib

mkdir hearts

cd !$

Copiamo i file di `distributor` in questa directory.

cp /home/username/hearts/heartsd .

cp /home/username/hearts/hearts_dist .

cp /home/username/hearts/hearts.instr .

-

Lanciamo un'altra esecuzione di prova.

xmhearts

Qualche volta funziona, ma il più delle volte si pianta con un messaggio `dealer died!`.

•

Il distributor e il dealer esaminano le porte hardware. Dovremmo perciò sospettare che quei programmi abbiano bisogno dei privilegi di root.

Proviamo, come *root*,

chmod u+s /usr/local/lib/heartsd

chmod u+s /usr/local/lib/hearts_dist

(Osserviamo che, come discusso precedentemente, i binari *suid* possono creare dei buchi di sicurezza.)

xmhearts

Alla fine funziona!

Hearts è disponibile da *Sunsite* .

13 Quinto esempio: XmDipmon

```
Bullwinkle: Hey Rocky, guarda, tiro fuori un coniglio dal cappello.
Rocky:      Ma quel trucco non funziona mai.
Bullwinkle: Questa volta si.
            Voila!
            Beh, ci sono andato vicino.
Rocky:      Ed ora e il momento di un altro effetto speciale.
            --- "Rocky e i suoi amici"
```

XmDipmon è un'elegante applicazioncina che mostra un bottone indicante lo stato di una connessione Internet. Lampeggia e suona quando la connessione si interrompe, come troppo spesso accade nelle linee telefoniche di campagna. Sfortunatamente XmDipmon funziona solo con *dip*, il che lo rende inutile per tutti quelli, la maggioranza, che usano *chat* per collegarsi.

Compilare XmDipmon non è un problema. XmDipmon si linka con le librerie *Motif*, ma può essere compilato, e funziona bene, anche con le *Lesstif*. La sfida è di modificare il pacchetto per farlo funzionare con *chat*. Ciò in pratica richiede di armeggiare con il codice sorgente ed è quindi necessario avere delle conoscenze di programmazione.

```
"Quando xmdipmon parte, cerca un file chiamato /etc/dip.pid
 (potete fargli cercare un altro file usando l'opzione -pidfile
 dalla riga di comando). Tale file contiente il PID del demone
 dip (dip stesso si pone in modo demone, dopo che ha stabilito
 una connessione)."
               --- dal file README di XmDipmon
```

Usando l'opzione *-pidfile*, il programma può essere indotto, al suo avvio, a cercare un altro file, uno che esista solo dopo che un login con *chat* è stato effettuato con successo. Il candidato più ovvio è il file di lock del modem. Potremmo quindi provare a lanciare il programma con **xmdipmon -pidfile /var/lock/LCK..ttyS3** (assumendo che il modem sia sulla porta com numero 4, ttyS3). Comunque così si risolve solo una parte del problema. Il programma osserva continuamente il *demone dip* e dobbiamo invece fare in modo che controlli un processo che sia associato a *chat* o a *ppp*.

C'è un solo file sorgente e fortunatamente è ben commentato. Scorrendo il file `xmdipmon.c` troviamo la funzione *getProcFile*, la cui descrizione, all'inizio, riporta quanto segue:

```
/*****
* Name:               getProcFile
* Return Type:  Boolean
* Description:  tries to open the /proc entry as read from the dip pid file.
<snip>
*****/
```

(Traducendo la descrizione: prova ad aprire l'elemento di /proc secondo quanto letto dal file pid di dip)

Siamo sulla pista giusta. Guardando nel corpo della funzione...

```
/* we watch the status of the real dip deamon */
sprintf(buf, "/proc/%i/status", pid);
procfile = (String)XtMalloc(strlen(buf)*sizeof(char)+1);
strcpy(procfile, buf);
procfile[strlen(buf)] = '\0';
```

Il colpevole è la linea 2383:

```
sprintf(buf, "/proc/%i/status", pid);
              ^^^^^^^^^^^^^^^^^^^^^
```

Questo controlla se il processo demone dip è in esecuzione. Quindi, come possiamo cambiarlo in modo che controlli il demone pppd?

Guardando sulla pagina di manuale di *pppd*:

```
FILES
     /var/run/pppn.pid (BSD o Linux), /etc/ppp/pppn.pid (altri)
           Identificatore del processo (Process-ID) di pppd sull'unita
           d'interfaccia n di ppp.
```

Cambiamo la linea 2383 di `xmdipmon.c` in:

```
sprintf(buf, "/var/run/ppp0.pid" );
```

Ricompiliamo il pacchetto così modificato. Nessun problema con la compilazione. Ora proviamolo con il nuovo argomento della riga di comando. Funziona che è un incanto. Il bottoncino blu indica quando è stata stabilita una connessione ppp con l'ISP e lampeggia e suona quando la connessione si interrompe. Ora abbiamo un monitor per *chat* pienamente funzionante.

XmDipmon può essere scaricato da *Ripley Linux Tools* .

14 Dove trovare archivi sorgente

Ora che siete ansiosi di usare le conoscenze appena acquisite, per aggiungere delle utilità ed altre chicche al vostro sistema, potete trovarle online alla

Linux Applications and Utilities Page `http://www.redhat.com/linux-info/linux-app-list/linapps.html` , o in una delle raccolte su CD ROM, a prezzo molto ragionevole, di

Red Hat `http://www.redhat.com/` ,

InfoMagic `http://www.infomagic.com` ,

Linux Systems Labs `http://www.lsl.com` , *Cheap Bytes* `http://www.cheapbytes.com` , e altri.

Un vasto magazzino di codice sorgente è *comp sources UNIX archive* .

Molto codice sorgente UNIX viene pubblicato nel newsgroup *alt.sources* . Se state cercando particolari pacchetti di codice sorgente, potete chiedere sullo specifico newsgroup *alt.sources.wanted* . Un altro buon posto dove cercare è il newsgroup *comp.os.linux.announce* . Per entrare nella mailing list *Unix sources* , mandate alla stessa un messaggio *subscribe*.

Gli archivi del newsgroup *alt.sources* si trovano presso i seguenti siti ftp:

- *ftp.sterling.com/usenet/alt.sources/*
- *wuarchive.wustl.edu/usenet/alt.sources/articles*
- *src.doc.ic.ac.uk/usenet/alt.sources/articles*

15 Conclusioni

Riassumendo, la perseveranza fa tutta la differenza (e un'alta soglia alla frustrazione certamente aiuta). Come in tutti gli sforzi, imparare dagli errori è criticamente importante. Ogni passo falso, ogni fallimento, contribuisce alla costruzione della conoscenza che conduce alla padronanza dell'**arte della compilazione del software**.

16 Riferimenti e ulteriori letture


```
BORLAND C++ TOOLS AND UTILITIES GUIDE, Borland International, 1992,
pp. 9-42.
[Uno dei manuali distribuiti col Borland C++, ver. 3.1. Da una
abbastanza buona introduzione alla sintassi e ai concetti di make,
usando l'implementazione limitata al DOS della Borland.]

DuBois, Paul: SOFTWARE PORTABILITY WITH IMAKE, O'Reilly and Associates,
1996, ISBN 1-56592-226-3.
[E ritenuto il riferimento definitivo per imake, sebbene non lo
avevo a disposizione durante la scrittura di questo articolo.]

Frisch, Aeleen: ESSENTIAL SYSTEM ADMINISTRATION (2nd ed.), O'Reilly and
Associates, 1995, ISBN 1-56592-127-5.
[Questo, altrimenti ottimo, manuale per amministratori di sistema tratta
solo in modo sommario la compilazione del software.]

Hekman, Jessica: LINUX IN A NUTSHELL, O'Reilly and Associates, 1997, ISBN
1-56592-167-4.
[Un buon riferimento globale ai comandi di Linux.]

Lehey, Greg: PORTING UNIX SOFTWARE, O'Reilly and Associates, 1995, ISBN
1-56592-126-7.

Mayer, Herbert G.: ADVANCED C PROGRAMMING ON THE IBM PC, Windcrest Books,
1989, ISBN 0-8306-9363-7.
[Un libro zeppo di idee per programmatori C medi ed esperti. Superba
trattazione degli algoritmi, ricercatezza di linguaggio e anche
```

```
divertimento. Sfortunatamente non viene piu stampato.]

Mui, Linda and Valerie Quercia: X USER TOOLS, O'Reilly and Associates,
1994, ISBN 1-56592-019-8, pp. 734-760.

Oram, Andrew and Steve Talbott: MANAGING PROJECTS WITH MAKE, O'Reilly
and Associates, 1991, ISBN 0-937175-90-0.

Peek, Jerry and Tim O'Reilly and Mike Loukides: UNIX POWER TOOLS,
O'Reilly and Associates / Random House, 1997, ISBN 1-56592-260-3.
[Una meravigliosa fonte di idee, e tonnellate di utilita che potete
compilare dal codice sorgente, usando i metodi discussi in questo
articolo.]

Stallman, Richard M. and Roland McGrath: GNU MAKE, Free Software
Foundation, 1995, ISBN 1-882114-78-7.
[Da leggere]

Waite, Mitchell, Stephen Prata, and Donald Martin: C PRIMER PLUS, Waite Group
Press, ISBN 0-672-22090-3,.
[Probabilmente la miglior introduzione alla programmazione in C. Vasta
copertura per un "primo libro". Sono ora disponibili nuove edizioni.]

Welsh, Matt and Lar Kaufman: RUNNING LINUX, O'Reilly and Associates,
1996, ISBN 1-56592-151-8.
[Tuttora il miglior riferimento globale per Linux, sebbene manchi
di profondita in alcune aree.]
```

```
Le pagine di manuale di dpkg, gcc, gzip, imake, ldconfig, ldd, make, nm,
patch, rpm, shar, strip, tar, termcap, terminfo, e xmkmf.
```

```
Il BZIP2 HOWTO (tradotto), di David Fetter.

Il Glibc2 HOWTO (tradotto), di Eric Green

Il LINUX ELF HOWTO (tradotto), di Daniel Barlow.

L'RPM HOWTO, di Donnie Barnes.

Lo StarOffice miniHOWTO, di Matthew Borowski.
```


[Questi HOWTO dovrebbero trovarsi nella directory `/usr/doc/HOWTO` o nella `/usr/doc/HOWTO/mini` sul vostro sistema. Versioni aggiornate sono disponibili in formato testo, HTML e SGML nel *sito LDP* `http://metalab.unc.edu/LDP/HOWTO` , e di solito nei siti dei rispettivi autori.]

——————————————

Traduzioni: Per la versione in italiano degli HOWTO con l'indicazione (tradotto) vedere: *www.pluto.linux.it/ildp/HOWTO/HOWTO-INDEX-3.html* . Per altra documentazione su Linux *in italiano*, compresi eventuali HOWTO e/o mini-HOWTO citati ma al momento non ancora tradotti, si veda il

sito ILDP `http://www.pluto.linux.it/ildp/index.html` .

17 Crediti

L'autore di questo HOWTO desidera ringraziare le seguenti persone per gli utili suggerimenti, le correzioni e l'incoraggiamento.

- R. Brock Lynn
- Michael Jenner
- Fabrizio Stefani

Gloria va anche a quelle brave persone che hanno tradotto questo HOWTO in italiano e giapponese.

E naturalmente, ringraziamenti, lodi, benedizioni e osanna a Greg Hankins e Tim Bynum del *Linux Documentation Project* `http://metalab.unc.edu/LDP/` .