Le HOWTO ELF

Daniel Barlow <daniel.barlow@linux.org>

v1.29, 14 Juillet 1996

Ce document explique comment modifier votre système Linux afin qu'il puisse compiler et exécuter les programmes au format exécutable ELF. Il se décompose en trois sous parties (1) ce qu'est ELF et les raisons qui vous pousseront à vous y mettre, (2) comment mettre à jour votre système pour utiliser ELF et (3) ce qu'il vous sera alors possible de faire.

Table des matières

1 Qu'est-ce que ELF? Introduction.

ELF (Executable and Linking Format) est un format de données binaires originellement développé par USL (UNIX System Laboratories) et désormais utilisé dans Solaris et dans le Système 5 Version 4. De par sa flexibilité accrue par rapport à l'ancien format a.out que Linux utilisait précédemment, les développeurs de GCC et de librairies décidèrent l'an dernier de se mettre à ELF et d'ainsi modifier le format standard des fichiers binaires de Linux.

Cette flexibilité accrue a en fait deux intérêts essentiels pour le programmeur d'applications moyen :

- Il est bien plus simple de faire des librairies partagées avec ELF. Usuellement, il suffit de compiler tous les fichiers objets avec `-fPIC` et d'ensuite établir les liens avec une commande telle que :

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

  Si vous trouvez cela complexe, c'est que vous n'avez visiblement pas eu affaire à la procédure équivalente pour les librairies partagées avec a.out où il faut compiler la librairie par deux fois, en ayant réservé la place pour toutes les données dont vous pensez que la librairie va avoir besoin dans le futur et en ayant enregistré l'espace d'adressage auprès d'une tierce partie. Tout cela est décrit dans un document de plus de 20 pages (voir `(ftp://tsx-11.mit.edu/pub/linux/packages/GCC/src/tools-2.17.tar.gz)` pour les détails).

- Il est également plus simple de gérer le chargement dynamique en mémoire (c'est-à-dire les programmes qui peuvent charger des modules au moment de l'exécution). Ceci est utilisé par Perl 5, Python et Java (entre autres). D'autres possibilités pour le chargement dynamique : des Donjons Multi-Utilisateurs

(Multi User Dungeons) où le code supplémentaire pourrait être compilé et lié au programme en fonctionnement sans avoir à l'arrêter et à le relancer.

Cependant, il faut prendre en compte qu'ELF est peut-être un peu moins rapide. Les chiffres les plus couramment cités font état d'une baisse de performance de 1 à 5%, bien que les tests effectués jusqu'à présent indiquent que la différence est suffisamment faible pour se perdre dans le bruit dû autres événements qui arrivent au même moment. Si vous avez un visualiseur ou un moyen d'imprimer des fichiers TeX ou PostScript, vous pourrez lire `speed.comp-1.0.tar.gz` qui se trouve quelque part sur SunSite.

Le ralentissement vient du fait que les codes des librairies ELF doivent être indépendants de la position (c'est ce que signifie le commutateur `-fPIC` vu ci-dessus signifie) : un registre doit alors être dédié pour la conservation des offsets. C'est un de moins pour les variables, alors que les 80x86 ont déjà une pénurie de registres à usage général. Notez bien que les différences de vitesse ne s'appliquent qu'aux parties du code qui font partie des librairies partagées. Pour les applications ou le noyau, il n'y a aucune différence de vitesse entre a.out et ELF.

1.1 Ce qu'ELF n'est pas

Il y a bon nombre d'erreurs de commises quant à ce que ELF va apporter à votre système :

Ce n'est pas un moyen d'exécuter des programmes SVR4 ou Solaris

Bien que ce soit le même conteneur binaire que celui que les systèmes SVR4 utilisent, cela n'implique pas que les programmes SVR4 vont soudainement se mettre à fonctionner sous Linux. Il en va de même que pour un format de disque - vous pouvez conserver des programmes Linux sur des disques au format MSDOS ou Minix, et vice versa, mais cela ne signifie pas pour autant que ces systèmes seront alors en mesure d'exécuter les programmes des autres.

Il est possible d'exécuter une application prévue pour une autre implémentation d'Unix pour des systèmes x86 sous Linux (cela dépend de l'application) mais suivre les instructions de ce HOWTO *n'aura pas* cet effet-là. Commencez par essayer le module iBCS du noyau (quelque part sur `tsx-11.mit.edu`) et voyez s'il ne satisfait pas vos besoins.

Ce n'est pas intrinsèquement plus compact ou plus rapide

Vous pouvez très bien vous retrouver avec des fichiers exécutables plus petits compte tenu du fait que vous pouvez plus aisément créer des librairies partagées de code commun à de nombreux programmes. En général, si vous utilisez les mêmes options de compilation et que vous obtenez des exécutables plus petits qu'avec a.out, ce sera dû soit à un coup de chance soit à une version différente du compilateur. Pour ce qui est de la mention plus rapide, j'en serais surpris. Des augmentations de performances peuvent apparaître si vos fichier compilés sont plus petits, du fait de la baisse des transferts avec des fichiers d'échange ou des domaines fonctionnels plus grands qui rentrent dans le cache.

Il n'est pas nécessaire de remplacer chacun des exécutables de votre système

A la fin des procédures indiquées ci-après, vous aurez un système capable de compiler et d'exécuter à la fois des programmes a.out et ELF. Les nouveaux programmes seront compilés par défaut en ELF bien que cela puisse être modifié par l'intermédiaire d'un commutateur sur la ligne de commande. Il est communément admis qu'on encourt une perte de mémoire quand on dispose d'un système à la fois a.out et ELF : ainsi, si vous avez les deux sortes de programmes qui tournent en même temps, il va s'en suivre deux copies de la même librairie C en mémoire et ainsi de suite. Cela dit, il semblerait que cette différence de vitesse est imperceptible en utilisation normale sur un système avec 8Mo (je n'en ai en tout cas pas remarqué avec 8Mo). Vous perdez bien plus de mémoire chaque jour quand vous utilisez des programmes gourmands comme Emacs!!!!!! ou les exécutables statiques de Mosaic ou de Netscape :-).

Rien à voir avec Tolkien

Du moins pas dans ce contexte.

1.2 Pourquoi se convertir à ELF ?

Il y a essentiellement deux raisons pour mettre à jour votre système pour ELF, la première étant la flexibilité accrue décrite plus haut pour la programmation et la seconde étant que, au vu de la première, tout le monde le fera (ou l'a déjà fait). Les dernières versions de la librairie C et de GCC sont compilées seulement pour ELF et les autres développeurs se mettent également à ELF.

Beaucoup de personnes ont pour souci la stabilité (ce qui est légitime bien que terre-à-terre). ELF a été utilisé sous Linux depuis Août 1994 et a été publiquement disponible aux alentours de Mai ou Juin 1995 ; les problèmes les plus graves ne sont plus qu'un lointain souvenir. Vous devriez permettre quelques menus défauts – comme avec toute mise à jour d'importance - mais la technologie à laquelle vous allez adhérer n'est plus experimentale. Pour un système sur lequel on fait un tant soit peu de développement ou sur lequel vous avez l'intention d'exécuter les programmes précompilés d'autres personnes, ELF est presque devenu une nécessité. Pensez à vous y mettre quand vous ferez la mise à jour pour le noyau 2.0.

1.3 Comment se convertir à ELF

Au moment où ce HOWTO fut originellement écrit, il n'y avait qu'une seule manière : celle décrite ci-dessous. De nos jours, il y a des distributions de haute qualité faciles à mettre à jour – à moins que vous n'ayez investi un temps certain à peaufiner la configuration de votre machine, vous trouverez certainement que faire une sauvegarde de vos données personnelles et réinstaller à partir d'une distribution RedHat ou Debian est plus facile que de jouer avec les librairies et les compilateurs décrits ici.

Je me dois d'insister. L'installation décrite ici est un travail de plutôt faible envergure en elle-même (elle peut être faite en moins d'une heure, mis à part le temps de téléchargement des logiciels) mais elle peut donner lieu à de multiples erreurs qui risquent de vous laisser avec un système non bootable. Si vous ne vous sentez pas sûr de vous dans la mise à jour des librairies partagées, si les commandes `ldconfig` et `ldd` ne vous disent rien et si construire des paquetages à partir du code source ne vous enchante pas, vous devriez considérer la solution de facilité. Même si la description ne vous correspond pas, pensez y tout de même : si vous voulez un système totalement ELF, *quelqu'un* va devoir en compiler tous les exécutables.

Vous êtes toujours là ?

2 Installation

2.1 Prérequis

Le but de cette mise à jour est de vous fournir un système qui puisse compiler et exécuter aussi bien les programmes basés sur a.out que ELF, en s'assurant que chaque type de programme soit capable de trouver la version appropriée des bibliothèques partagées. Il apparaît aisément que cela demande plus de travail que simplement chercher dans `/lib`, dans `/usr/lib` ou n'importe où ailleurs selon le chemin de recherche indiqué dans la compilation, stratégie dont certains autres systèmes se satisfont.

Le travail nécessaire est centralisé au niveau d'un *chargeur dynamique* qui existe en un seul, ou deux, endroit du système. Pour les programmes a.out, il est appelé `/lib/ld.so` et, pour les programmes ELF, on fait appel à `/lib/ld-linux.so.1`. Le compilateur et l'éditeur de liens n'encodent pas les chemins absolus des librairies dans les programmes qu'ils produisent ; ils fournissent en effet le nom de la librairie et son chemin absolu au chargeur dynamique approprié et se contentent de ça pour apparier le nom de la bibliothèque au chemin correspondant lors de l'exécution. Cela a une conséquence d'importance : les bibliothèques peuvent être déplacées dans d'autres répertoires *sans recompiler le programme* dès lors que `ld.so` (ou `ld-linux.so.1` selon le cas) a été informé de chercher dans le nouveau répert!!!!!! oire. C'est une fonctionnalité essentielle pour l'échange de répertoires qui va suivre.

Le corollaire de ce qui précède est bien sûr que toute tentative d'effacer ou de déplacer `ld.so` ou `ld-linux.so.1` pourrait provoquer *le plantage de tout programme dynamiquement lié sur le système*. C'est généralement vu comme une Mauvaise Chose.

Le schéma général est alors de mettre tout ce qui concerne le développement ELF (compilateurs, bibliothèques et fichiers d'inclusions) dans `/usr/{bin,lib,include}` et de mettre ce qui concerne a.out dans `/usr/i486-linuxaout/{bin, lib, include}`. Le fichier `/etc/ld.so.conf` est la liste de tous les endroits du système où l'on s'attend à trouver les bibliothèques et `ldconfig` est suffisamment malin pour distinguer les versions ELF des versions a.out.

Il y a néanmoins quelques exceptions quant au placement de la bibliothèque :

– Certains vieux programmes ont été compilés sans l'utilisation de `ld.so`. Ceux-ci cesseraient alors de fonctionner si leurs librairies devaient être déplacées. Ainsi `libc.so*` et `libm.so*` doivent rester où ils sont dans `/lib` et les versions ELF ont vu leurs numéros majeurs mis à jour afin qu'ils n'écrasent pas les versions a.out. Les vieilles bibliothèques X (avant la version 6) feraient mieux de rester où elles sont alors que les plus récentes (`libX*so.6`) doivent être déplacées. Déplacer les anciennes bloquera xview et ne pas déplacer les nouvelles fera qu'elles seront écrasées lors de l'installation des bibliothèques ELF.

  Si vous avez des programmes non ld.so qui nécessitent des bibliothèques autre que celles citées ci-dessus (si vous savez de quels programmes il s'agit, vous pouvez exécuter ldd sur ceux-ci afin de déterminer quelles bibliothèques leurs sont nécessaires *avant* de les bloquer), deux solutions s'offrent à vous. La première est d'extraire la librairie ELF dans un répertoire temporaire, de voir si votre précieuse librairie va être écrasée et, si le cas se présente, de déplacer la version ELF de la bibliothèque dans, disons, `/usr/i486-linux/lib` en lieu et place de `/lib`.Assurez vous que `ld.so.conf` contient `/usr/i486-linux/lib` et exécutez `ldconfig` puis n'y pensez plus. La seconde est de recompiler ou d'obtenir une nouvelle version du programme incriminé. Cela n'est pas une mauvaise idée, si c'est possibl!!!!!! e.

- Si vous avez `/usr` et `/` sur des partitions différentes, toutes les bibliothèques que vous déplacerez de `/lib` devront se retrouver autre part sur le disque racine et pas sur `/usr`. J'ai utilisé `/lib-aout` pour les instructions qui suivent.

2.2 Avant de commencer : notes et mises en gardes

- Il va vous falloir tourner sous un noyau postérieur au 1.1.52 **avec support des fichiers exécutables ELF**. Le 1.2.13 ainsi que le 2.0.30 font l'affaire ainsi que la plupart des noyaux de la série 1.3 bien que l'intérêt de tourner sous des anciens noyaux expérimentaux est maintenant discutable avec l'arrivée du 2.0.
- Il est recommandé de se munir d'une disquette d'amorçage Linux telle que la disquette de sauvetage de Slackware. Vous n'en aurez probablement pas besoin mais si vous n'en avez pas et que vous en ayez besoin, vous serez dans le pétrin. Dans le même esprit "il vaut mieux prévenir que guérir", des copies statiquement liées de `mv`, `ln` ou d'autres commandes de manipulation de fichier pourraient vous aider à sortir des situations périlleuses dans lesquelles vous pourriez vous mettre.
- Si vous avez suivi le développement de ELF depuis le début ou si vous avez installé certaines versions de Slackware, vos bibliothèques ELF peuvent être dans `/lib/elf`. Les applications que vous avez compilées avec celles-ci doivent être recompilées avant d'effacer ce répertoire. `/lib/elf` n'est pas nécessaire!
- La plupart des installations Linux modernes se sont rejointes sur l'adoption du système de fichiers standard "FSSTND" mais il y en a sans nul doute quelques unes qui sont en retard. Si vous voyez des références à `/sbin/`*quelque chose* et que vous n'avez pas de répertoire `/sbin/`, vous trouverez probablement le programme en question dans `/bin` ou `/etc/`. Ceci est particulièrement important quand vous installez de nouveaux programmes: si `/etc` est placé avant `/sbin` dans le chemin de recherche, vous obtiendrez des erreurs parce que vous ferez appel aux anciennes versions alors que vous n'auriez pas dû.
- C'est une bonne idée de choisir un moment où nul n'utilise l'ordinateur ou de le lancer en mode mono-utilisateur. Relancez-le d'une disquette d'amorçage de manière à ne pas rester bloqué, même si, pour ma part, j'aime bien jouer avec le feu...

2.3 Notions

Tout ce qui est décrit ci-après par "sur `tsx-11`" peut être trouvé à `(ftp://tsx-11.mit.edu/pub/linux/packages/GCC/)`, `(ftp://sunsite.unc.edu/pub/Linux/GCC/)`, et d'autres sites miroir (NdT: comme `(ftp://ftp.lip6.fr/pub/linux/)` pour la France). Veuillez consulter le site miroir le plus proche de chez vous plutôt que les sites généraux dès que cela est possible. Cela ira plus vite, pour vous et pour les autres.

Ces paquetages (que ce soit la version listée ou une plus récente) sont requis. Pensez également à télécharger les notes concernant votre version pour chacun d'entre eux: elles sont de la forme `release.`*nom_du_paquetage*. Cela est particulièrement vrai pour les versions plus récentes que celles présentées ici (l'installation a pu changer).

Même si vous avez pour habitude de compiler à partir des sources, je vous recommande de prendre les versions binaires quand je l'indique à moins que vous n'ayez *réellement* plus besoin de vos cheveux. La plupart d'entre eux ne sont pas prévus pour la compilation sur un système mixte et vous vous exposez à de graves ennuis si vous essayez.

2.3.1 Les essentiels

- `ld.so-1.7.14.tar.gz` — le nouveau générateur de liens dynamiques. Il contient à la fois les sources et les versions compilées. Notez que la version à venir nécessitera une prise en charge d'ELF même pour les exécutables a.out; si vous récupérez la version 1.8.1 ou plus récente en lieu et place de la version mentionnée, vérifiez que votre noyau a été compilé avec l'option support d'ELF *avant* de l'installer.
- `libc-5.3.12.bin.tar.gz` — les images partagées pour ELF des bibliothèques C et de maths, plus les bibliothèques correspondantes en statique et les fichiers include nécessaires pour compiler les programmes. Si vous voulez le code source, préparez vous alors à attendre pendant des heures pendant la compilation, si jamais il se compile (à moins que vous n'ayez déjà un système ELF).
- `gcc-2.7.2.bin.tar.gz` — le paquetage du compilateur C ELF qui comprend également un compilateur C a.out qui tient compte de la nouvelle disposition des répertoires. Si vous vous voulez compiler gcc vous même (ce que vous trouverez certainement plus simple si vous utilisez déjà ELF), il est recommandé d'appliquer `gcc-2.7.2-linux.diff.gz` aux sources GNU avant de le faire.
- `binutils-2.6.0.12.bin.tar.gz` — les utilitaires GNU en version Linux. Ce sont des programmes tels que `gas`, `ld`, `strings` et ainsi de suite, la plupart d'entre eux étant requis pour l'exécution du compilateur C. Notez bien que les binutils GNU "vanilla" (à savoir ceux de `prep.ai.mit.edu`) ne sont pas un substitut acceptable. Si vous voulez réellment les compiler vous même, faites appel au paquetage modifié pour Linux `binutils-2.6.0.12.tar.gz` plutôt qu'à la version GNU.
- `ncurses-1.9.9e.tar.gz` — c'est une bibliothèque curses compatible SVR4, c'est donc la bibliothèque considérée comme standard concernant curses sous Linux. On peut en obtenir le code source sur des sites GNU tels que `(ftp://prep.ai.mit.edu/gnu/)` ou encore sur `(ftp://ftp.netcom.com/pub/zm/zmbenhal)`. Il y a aussi une version compilée du paquetage sur `tsx-11`. Avant que vous installiez cela, vous disposerez d'un système de développement ELF pleinement fonctionnel, je vous recommande donc le paquetage source si vous avez la moindre once de puissance pour la compilation.
- `gdbm-1.7.3.tar.gz` est un ensemble de routines de base de donnés qui fait appel au hachage et qui fonctionne de manière comparable aux routines dbm et ndbm standards d'UNIX. Le source est récupérable sur `(ftp://prep.ai.mit.edu/gnu/)` et vous aurez besoin du patch `(ftp://ftp.uk.linux.org/pub/Linux/libc/non-core/gdbm.patch)` pour en obtenir des bibliothèques partagées. Le patch corrige également quelques petites choses (une coquille dans le Makefile et une prédisposition à utiliser une mauvaise option de protection des fichiers).

2.3.2 Autres

Il y a d'autres bibliothèques et fichiers qui ne sont pas essentiels mais que vous voudrez probablement avoir de toute façon. Cette liste comporte seulement des paquetages qui nécessitent d'être mis à jour pour fonctionner de manière utile vis-à-vis d'ELF. Plus loin dans ce document, une nouvelle liste indique les programmes qui continueront à fonctionner mais qu'il vous faudra mettre à jour pour les recompiler en ELF.

- Le paquetage **compatibilité a.out** : `libc.so-4.7.6`. Il est marqué optionnel parce que vos bibliothèques a.out existantes continueront à marcher avec vos exécutables actuels. Vous ne le prendrez que si vous considérez de continuer à développer en a.out pour quelque raison que ce soit.

- **BSD curses**. Si vous trouvez des exécutables qui nécessitent `libcurses.so.1`, c'est l'ancienne version de la bibliothèque curses de BSD. Ils sont plutôt rares, ce qui est plutôt une bonne chose car je ne peux pour l'instant pas mettre la main sur une version en code source de cette bibliothèque. Il est probablement préférable de recompiler de tels programmes de manière à fonctionner avec ncurses. Vous trouverez une version compilée de `libcurses.so` ds `libc-5.0.9.bin.tar.gz` sur les sites miroirs de `tsx-11`.

- **Berkeley db**: les nouvelles routines BSD 4.4 `libdb` de base de données. Le source est téléchargeable `(ftp://ftp.cs.berkeley.edu/ucb/4bsd/db.1.85.tar.gz/)` et le patch pour les bibliothèques partagées sous Linux est `(ftp://ftp.uk.linux.org/pub/Linux/libc/non-core/db.patch)`.

- **C++**: le paquetage `gcc` est fourni avec `g++`, mais vous aurez également besoin de `libg++-2.7.1.4.bin.tar.gz` pour compiler n'importe quel programme C++ utile. Pour ma part, je n'utilise pas C++ mais je suis en mesure de comprendre qu'il n'est pas aisé de compiler cela à partir du code source d'où la recommandation quant aux versions compilées.

- **termcap compatible GNU**. Le passage à ncurses ne s'est pas fait simultanément au passage à ELF - vous pourriez avoir besoin d'exécuter des programmes d'autres personnes qui font appel à cette bibliothèque. `gdb` en est un bon exemple. Si vous avez l'intention de déboguer des bibliothèques partagées et que gdb est troublé par celles qui sont liées à elles-mêmes, vous voudrez certainement une copie statiquement liée de celle-ci. Dans ce cas, vous trouverez qu'un véritable termcap est bien moins encombrant que des routines compatibles termcap dans ncurses.

  `termcap-2.0.8.tar.gz` se trouve sur `tsx-11`. Ce *n'est pas* le Termcap GNU, mais il lui est compatible (les différences résident apparamment dans le contrôle des erreurs). C'est un paquetage en version code source.

- **MAKEDEV**. Dans certaines implémentations, cette utilitaire efface les entrées existantes des périphériques pour les recréer par la suite. Cela peut planter certaines versions de `ld-linux.so.1` s'il efface `/dev/zero`. Une nouvelle version est disponible à `(ftp://sunsite.unc.edu/pub/Linux/system/Admin/MAKEDEV-C-1.5.tar.gz)` ou `(ftp://sunsite.unc.edu/pub/Linux/system/Admin/MAKEDEV-2.2.tar.gz)`.

- `modules-2.0.0`. Si vous utilisez des modules, la mise à jour des binutils que vous allez bientôt effectuer va empêcher le fonctionnement de tous les utilitaires de modules plus vieux que la 1.3.69. Les nouvelles versions des modules peuvent être obtenues sur `(http://www.pi.se/blox/)`.

- Le système **X window** comporte un grand nombre de bibliothèques partagées. Comme vos nouveaux programmes seront en ELF, et que les programmes ELF ne peuvent tirer partie des bibliothèques a.out, vous allez avoir besoin d'une nouvelle installation de X si vous voulez faire un tant soit peu de développement sous X. XFree86 3.1.2 est diponible sous les formats ELF et a.out. Allez par `ftp` sur `ftp.xfree86.org`, lisez le message "too many users" que vous n'allez pas manquer de lire et choisissez le site miroir le plus proche de chez vous. Une fois que vous avez récupéré les répertoires `common` et `elf`, vous aurez alors à éditer `/usr/X11R6/lib/X11/config/linux.cf` pour modifier les lignes suivantes:

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

en mettant `YES` à la place. Autrement, une compilation de xpm conduira à utiliser les anciens utilitaires du passé. Notez que les exécutables de Xfree86 nécessitent maintenant qu'une bibliothèque termcap partagée en ELF (`libtermcap.so.2`) soit installée.

Si vous utilisez Motif, contactez votre fournisseur pour savoir s'il est en mesure de vous fournir des librairies partagées Motif en ELF. Ne l'utilisant pas, je ne vous suis d'aucune utilité à ce sujet.

– Si vous passez à Linux 2.0 en même temps que votre mise à jour à ELF, n'oubliez pas de consulter le fichier `Documentation/Changes` qui est fourni avec les sources du noyau pour voir de quoi vous aurez besoin par ailleurs.

2.4 Réarrangement de votre système de fichiers.

Bien! Pour la suite, veuillez noter que quand j'écris "effacer", je veux dire "sauvegarder puis effacer" :-). Une bonne inspiration et on y va...

1. **primordial — installation des exécutables**
2. Créez les nouveaux répertoires dans lesquels vous allez déplacer ce qui concerne a.out

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

3. Détarrez le paquetage de l'éditeur de liens dynamique `ld.so-1.7.14` dans le répertoire dans lequel vous mettez habituellement le code source et lisez de suite le script `ld.so-1.7.14/instldso.sh` qui vient d'être décompressé. Si vous avez réellement un système standard, lancez-le en tapant `sh instldso.sh`, mais si il y a quoi que ce soit d'anormal, installez-le à la main. Par anormal, je veux dire:
   – utiliser zsh comme shell (certaines versions de zsh définissent `$VERSION`, ce qui semble gêner `instldso.sh`)
   – avoir des liens symboliques de `/lib/elf` vers `/lib` (ce dont vous ne devriez pas avoir besoin mais c'est une bien piètre consolation quand vous êtes à la recherche de votre disquette de sauvetage)
4. Editez `/etc/ld.so.conf` pour y ajouter le nouveau répertoire `/usr/i486-linuxaout/lib` (et `/lib-aout` si vous allez en avoir besoin). Relancez ensuite `/sbin/ldconfig -v` pour vérifier qu'il prend en compte les nouveaux répertoires.
5. Déplacez toutes les bibliothèques a.out dans `/usr/lib` et `/usr/*/lib` vers `/usr/i486-linuxaout/lib`. Notez que j'ai écrit les bibliothèques et non pas tous les fichiers. Ce sont les fichiers qui correspondent à la spécification `lib*.so*` , `lib*.sa*`, ou `lib*.a`. Ne commencez pas à déplacer par exemple `/usr/lib/gcc-lib`.

6. Regardez maintenant dans `/lib`. Laissez tranquille les fichiers `libc.so*`, `libm.so*`, et `libdl.so*`. Si vous disposez de liens symboliques vers les bibliothèques X (`libX*.so.3*`), laissez-les également là car Xview et d'autres paquetages en ont besoin. Laissez les fichiers `ld.so*`, `ld-linux.so*` et tous les autres fichiers commençant par `ld`. Pour ce qui est des autres bibliothèques (s'il en reste), si vous avez `/usr` sur la partition racine, mettez-les dans `/usr/i486-linuxaout/lib`. Si `/usr` est monté séparément, mettez-les dans `/lib-aout`. Lancez maintenant `ldconfig -v`.

7. Effacez le répertoire `/usr/lib/ldscripts` s'il existe, en vue de l'installation des binutils (qui va le recréer)

8. Effacez toutes les copies de `ld` et de `as` (*excepté* `ld86` et `as86`) que vous trouverez dans `/usr/bin`.

9. Vous devez nettoyez votre arborescence `/usr/include`. Sur un système normal, certains des fichiers qui se trouvent ici sont des fonctionnalités de base et sont fournis avec libc, alors que d'autres proviennent d'autres paquetages que vous ou l'auteur de votre distribution avez installé. En tenant compte de cela, je suggère que vous la reconstruisiez à partir de rien: renommez-la en `/usr/include.old`, et extractez `libc-5.2.18.bin.tar.gz` à partir du répertoire racine.

10. Installez le paquetage binutils. `tar -xvzf binutils-2.6.0.12.bin.tar.gz -C /` est une bonne manière de le faire.

11. Le paquetage gcc doit être extracté à partir de la racine. Il installe quelques fichiers dans `/usr/bin` et bien plus encore dans `/usr/lib/gcc-lib/i486-linux/2.7.2` et `/usr/lib/gcc-lib/i486-linuxaout/2.7.2`. Tapez:

    ```
    $ tar ztf gcc-2.7.2.bin.tar.gz
    ```

    pour voir ce qu'il contient, sauvegardez tout ce qu'il va écraser et que vous voudriez conseerver (par exemple, si vous avez installé Gnu ADA, vous voudrez conserver `/usr/bin/gcc`), et tapez juste:

    ```
    # tar -zxf gcc-2.7.2.bin.tar.gz -C /
    ```

    A cette étape, vous devriez être en mesure d'exécuter `gcc -v` et de compiler des programmes tests. Essayez:

    ```
    $ gcc -v
    Reading specs from /usr/lib/gcc-lib/i486-linux/2.7.2/specs
    gcc version 2.7.2
    $ gcc -v -b i486-linuxaout
    Reading specs from /usr/lib/gcc-lib/i486-linuxaout/2.7.2/specs
    gcc version 2.7.2
    $ ld -V
    ld version 2.6 (with BFD 2.6.0.2)
      Supported emulations:
       elf_i386
       i386linux
       i386coff
    ```

suivi bien sûr du programme traditionnel "Hello, world". Essayez-le avec `gcc` et avec `gcc -b i486-linuxaout` pour vérifier que les compilateurs a.out et ELF sont bien configurés.

Terminé? Pas exactement. Les bibliothèques qui ne sont pas de base ne sont pas encore installées et de nombreux liens symboliques sont encore à établir. Courage...

Liens symboliques

12. Certains programmes (notamment de nombreux programmes X) utilisent `/lib/cpp`, qui sous Linux est généralement un lien vers `/usr/lib/gcc-lib/i486-linux/`*version*`/cpp`. Comme l'étape précédente a très certainement effacé la version de `cpp` vers laquelle il pointait, vous allez devoir recréer le lien:

```
# cd /lib
# ln -s /usr/lib/gcc-lib/i486-linux/2.7.2/cpp .
```

13. Quand vous avez déplacé `/usr/include` vers `/usr/include.old`, vous avez perdu les liens symboliques à l'intérieur des sources du noyau. Faites:

```
# cd /usr/include
# ln -s ../src/linux/include/linux .
# ln -s ../src/linux/include/asm .
```

(en considérant que vos sources du noyau sont dans `/usr/src/linux`).

14. Les gens de FSSTND se sont justifiés de l'avoir gardé en déplaçant les fichiers `utmp` et `wtmp` de `/var/adm` vers `/var/run` et `/var/log` respectivement. Vous allez devoir créer des liens en fonction de leurs emplacements et vous aurez peut-être à créer les répertoires `/var/log` et `/var/adm`. Je reproduis ci-dessous le résultat de `ls -l` sur mon système:

```
$ 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
```

Consultez FSSTND (à partir des archives de la LDP telles que (`ftp://sunsite.unc.edu/pub/Linux/docs/fsstnd/`)) pour les détails.

Réjouissez-vous!

A cet instant, vous devriez disposer d'un environnement de développement ELF pleinement fonctionnel (enfin plus ou moins). Relaxez vous et fêtez cela pendant quelques minutes.

Paquetages essentiels en code source

15. **L' installation de ncurses** est un travail de longue haleine bien que vous puissiez profiter du temps de compilation pour lire l'Usenet. Après avoir décompressé le fichier tar, lisez le fichier `INSTALL` en vous considérant comme "un constructeur de distribution Linux ou de paquetages", à savoir que vous allez probablement le configurer avec une commande du style

```
$ ./configure --with-normal --with-shared --disable-termcap --enable-overwrite --prefix=/usr
```

Prenez également garde aux commentaires qui concernent le type de terminal par défaut: dans les noyaux 1.3 et 2.0, on le définit par `linux` au moment du boot mais vous aurez peut-être à éditer `/etc/inittab` pour éviter de le redéfinir en tant que `console` par `getty`.

Si `/usr/lib/terminfo` ne se trouve pas sur le disque racine (qui contient /), vous allez avoir affaire au "support du fallback" avec ncurses. Ceci est documenté dans le fichier `INSTALL` mentionné ci-dessus: c'est simple mais pénible car vous allez devoir compiler la bibliothèque deux fois. Si le fait d'avoir `linux` et `vt100` comme fallbacks ne vous dérange pas, vous pouvez copier `fallback.c` que vous trouverez à (`ftp://ftp.uk.linux.org/pub/Linux/libc/non-core/fallback.c`) sur l'existant.

Après avoir installé ncurses, vous allez devoir bidouiller dans `/usr/lib` car il y fait des choses non optimales qu'il est plus simple de réparer à la main. Notez que les contradictions entre les numéros de version, bien que peu agréables, ne sont d'aucun danger pour la santé humaine.

(a) `/usr/lib/libncurses.so.1.9.9e` devrait être déplacé vers `/lib` de manière à ce que les programmes curses qui fonctionnent en mode mono-utilisateur continuent à le faire. Si `/usr/lib` se trouve sur votre partition racine, cela n'est pas nécessaire bien que cela ne fasse pas de mal.

(b) Dans `/lib`, établissez un lien vers `libncurses.so.1.9.9e` appelé `libncurses.so.3.0`.

(c) Vous aurez également besoin des liens `/usr/lib/libncurses.so`, `/usr/lib/libcurses.so` et `/usr/lib/libtermcap.so` qui doivent tous pointer vers `/lib/libncurses.so.3.0`.

Pour résumer, cela donne:

```
# cd /lib
# mv /usr/lib/libncurses.so.1.9.9e .
# ln -s libncurses.so.1.9.9e libncurses.so.3.0
# cd /usr/lib
# ln -s /lib/libncurses.so.3.0 libncurses.so
# ln -s /lib/libncurses.so.3.0 libcurses.so
# ln -s /lib/libncurses.so.3.0 libtermcap.so
```

16. Installation de **gdbm**. Décompressez le code source dans un répertoire de code source, appliquez le patch `gdbm.patch`, et consultez les fichiers `README` et `INSTALL`.

La procédure de compilation sera alors du genre:

```
$ tar zxf gdbm-1.7.3.tar.gz
$ patch -p0 < gdbm.patch
$ cd gdbm-1.7.3
$ ./configure --prefix=/usr
$ make
$ make progs
$ su
# make install
# make install-compat
# cd /usr/lib
# ln -s libgdbm.so.1 libgdbm.so
# ln -s libgdbm.so.1 libgdbm.so.2
# ldconfig
```

La dernière étape est pour la compatibilité ascendante: certaines distributions utilisent `libgdbm.so.2` qui contient le même code que `libgdbm.so.1` mais mal numéroté pour des raisons historiques.

Paquetages optionnels en code source. En général, vous pouvez vous contenter de les installer en suivant les procédures qu'ils indiquent que je ne vais donc pas répéter. Il y a toutefois deux exceptions:

17. Si vous voulez **termcap version GNU** (optionnel à strictement parler mais nécessaire pour utiliser les exécutables XFree86), il doit être compilé à partir du code source mais cela ne devrait pas être plus compliqué que

```
$ tar zxf termcap-2.0.8.tar.gz
$ cd termcap-2.0.8
$ make
$ su
# cp libtermcap.so.2.0.8 /usr/lib
# ldconfig
```

    Je vous recommande de *ne pas* faire `make install` pour ne pas compromettre l'installation antérieure de ncurses. Si vous avez besoin de compiler des choses à partir de cette bibliothèque plutôt que simplement exécuter des exécutables faits avec elle, pensez à placer en lieu sûr les fichiers d'en-tête et les bibliothèques statiques et à utiliser les commutateurs `-I` et `-L` quand vous compilerez les choses en question. L'aspect vague de cette description devrait rendre clair que l'utilisation de termcap est déconseillée à moins que vous n'ayez de bonnes raisons.

18. Pour `libdb`, c'est quelque chose du genre:

```
$ tar zxf db.1.85.tar.gz
$ patch -p0 <db.patch
$ cd db.1.85/PORT/linux
$ make
$ su
# mkdir /usr/include/db
# ldconfig
# cp libdb.so.1.85.3 /usr/lib ; ( cd /usr/lib && ln -s libdb.so.1 libdb.so )
# cp ../../include/*.h /usr/include/db
```

    Veuillez noter

    - vous n'appliquez pas `PORT/linux/OTHER_PATCHES` car il est contenu dans ce patch
    - vous installez les fichiers d'en-tête autre part que dans `/usr/include` — il y a en effet conflit avec ceux que gdbm utilise. Pour compiler les programmes qui nécessitent `libdb`, vous devrez ajouter `-I/usr/include/db` à la ligne de commande du compilateur C.

2.5 A quoi cela devrait ressembler (les grandes lignes de l'arborescence des répertoires)

Ceci est un guide délibérément vague qui indique en gros ce que sont les fichiers que vous venez d'installer. Cela peut être utile dans un contexte de dépannage ou si vous décidez d'effacer quelque chose.

2.5.1 `/lib`

- L'editeur dynamique de liens`ld.so` (a.out) et `ld-linux.so.1` (ELF). Chacun d'entre eux peut être un lien symbolique mais vérifiez que les fichiers vers lesquels ils pointent existent.
- Les bibliothèques partagées de base `libc.so.4` et `libm.so.4` (a.out). Ce sont des liens symboliques dont il faut vérifier qu'ils pointent vers des fichiers réels.
- Les bibliothèques partagées de base `libc.so.5`, `libm.so.5`, `libdl.so.1`,`libncurses.so.1`,`libtermcap.so.2`, (ELF). Encore des liens symboliques, même remarque que ci-dessus

2.5.2 `/usr/lib`

- Tous les fichiers qui ne sont pas des bibliothèques et les répertoires qui étaient là auparavant.
- `libbfd.so*`,`libdb.so*`, `libgdbm.so*`: bibliothèques partagées ELF.
- Encore des liens symboliques. Pour chacune des bibliothèques dans `/lib` ou `/usr/lib`, il devrait y avoir un lien symbolique ici. Le nom du lien devrait être le véritable nom du fichier en enlevant le numéro de version. Par exemple `libc`,

```
lrwxrwxrwx   1 root     root           14 May  2 20:09 /lib/libc.so.5 -> libc.so.5.3.12
-rwxr-xr-x   1 bin      bin        583795 Apr 25 06:15 /lib/libc.so.5.3.12
lrwxrwxrwx   1 root     root           12 Oct 27  1995 /usr/lib/libc.so -> /lib/libc.so.5
```

  Ces liens sont utilisés par `ld` au moment de l'édition des liens.
- `libbsd.a`, `libgmon.a`, `libmcheck.a`, `libmcheck.a` et un fichier `lib*.a` pour chacune des bibliothèques partagées ELF dans `/lib` et `/usr/lib`. Les bibliothèques ELF statiques. Celles qui sont l'équivalent des librairies partagées ne seront pas d'une grande utilité pour la plupart des gens — quand vous utilisez ELF, vous pouvez employer le commutateur `gcc -g` avec les bibliothèques partagées, il n'y a donc plus de raison d'utiliser les versions statiques. *Vous en aurez besoin* si vous voulez déboguer les bibliothèques elles-mêmes.
- `crt0.o`, `gcrt0.o`. fichiers de "début de programme" en a.out; l'un d'entre eux est lié en tant que premier fichier dans tout programme a.out que vous compilez, à moins que vous ne vous arrangiez pour qu'il n'en soit pas ainsi.
- `crt1.o`, `crtbegin.o`, `crtbeginS.o`, `crtend.o`, `crtendS.o`, `crti.o`, `crtn.o`, `gcrt1.o`. Fichiers de démarrage ELF. Ils font la même chose que les fichiers `*crt0.o` ci-dessus pour les programmes ELF.

2.5.3 `/usr/lib/ldscripts`

- C'est là que les scripts de gestionnaires de périphériques pour `ld` vont, comme le nom le suggère. Cela devrait ressembler à:

```
$ 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. Ce sont des liens symboliques vers les binutils réels de /usr/bin.

2.5.5 /usr/i486-linuxaout/bin

– as — l'assembleur a.out, and gasp, son préprocesseur de macro

– ar, ld, nm, ranlib, strip — liens symboliques vers les binutils réels de /usr/bin

2.5.6 /usr/i486-linux/lib

– ldscripts est un lien symbolique vers /usr/lib/ldscripts.

2.5.7 /usr/i486-linuxaout/lib

– lib*.so*. Images de la bibliothèque partagée a.out. Nécessaire pour exécuter des programmes a.out.

– lib*.sa. Bases de la bibliothèque partagée a.out. Nécessaires pour compiler les programmes a.out qui utilisent des bibliothèques partagées. Si vous n'en avez pas l'intention, vous pouvez sans problème les effacer.

– lib*.a. Bibliothèques statiques a.out. Nécessaires pour compiler les programmes statiques a.out (à savoir quand vous compilez avec -g). Vous pouvez aussi les effacer si vous n'avez pas l'intention de compiler de tels programmes.

– ldscripts est un lien symbolique vers /usr/lib/ldscripts

2.5.8 /usr/lib/gcc-lib/i486-linux/2.7.2

– Ce répertoire contient une version de gcc 2.7.2 configurée pour compiler les programmes ELF.

2.5.9 /usr/lib/gcc-lib/i486-linuxaout/2.7.2

– Ce répertoire contient une version de gcc 2.7.2 configurée pour compiler les programmes a.out qui tient compte de la nouvelle arborescence de répertoires. Si vous n'avez pas l'intention de compiler quoi que ce soit en a.out, l'effacer devrait libérer environ 4Mo. Notez que vous devez les conserver si vous voulez compiler des noyaux 1.2 non-patchés.

2.6 Erreurs communes — Ne paniquez pas!

– Vous avez déplacé la mauvaise chose et maintenant plus rien ne fonctionne

Votre shell tourne toujours et en vous donnant un peu de mal, vous pouvez faire beaucoup avec les commandes intégrées du shell. Souvenez vous que `echo *` remplace très bien `ls`, et que `echo >>filename` peut être utilisé pour ajouter des lignes à un fichier. N'oubliez pas non plus que `ldconfig` est lié statiquement. Si vous avez déplacé, par exemple, `libc.so.4` vers `/lib-aout` par erreur, vous pouvez faire `echo "lib-aout"` >>/etc/ld.so.conf; ldconfig -v/ et vous retrouver sur vos pieds. Si vous avez déplacé `/lib/ld.so`, vous pourrez sûrement faire `sln /silly/place/ld.so /lib/ld.so`, si ln est lié statiquement, et une fois de plus vous retrouver sur vos pieds.

– `bad address`

quand vous essayez d'exécuter quoi que ce soit d'ELF. Vous utilisez un noyau 1.3.x, où x<3. Arrêtez tout de suite. Ce sont probablement les noyaux les plus bogués de la planète. Passez au 2.0 ou revenez au 1.2.13. Certaines personnes font également état de "kernel panics" dans des circonstances similaires; je ne me suis pas penché sur la question principalement parce que je ne vois pas de raisons d'utiliser les noyaux de développement sans suivre les mises à jour.

– `gcc: installation problem, cannot exec` *quelque chose*`: No such file or directory`

quand vous essayez de faire des compilations a.out (*quelque chose* est généralement `cpp` ou `cc1`). Ou bien cela est vrai ou bien vous avez tapé

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

quand vous auriez du taper

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

Le "i486" ne commence *pas* par un tiret.

– `make: *** No targets specified and no makefile found. Stop.`

indique que vous n'avez pas patché ou recompilé `make`, ou que vous en avez toujours une vieille version quelque part sur le système.

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

(ou n'importe quel autre fichier que vous essayez d'exécuter) quand vous *savez* qu'un tel fichier existe. Cela veut généralement dire que le chargeur dynamique ELF `/lib/ld-linux.so.1` n'est pas installé ou est illisible pour une raison ou pour une autre. Vous auriez du l'installer antérieurement, aux alentours de l'étape 2.

– `not a ZMAGIC file, skipping`

selon `ldconfig`. Vous avez une vieille version du paquetage ld.so, donc récupérez-en une plus récente. Voir étape 2 de l'installation.

– `_setutent: Can't open utmp file`

Ce message apparaît souvent en groupe de trois quand vous lancez un xterm. Veuillez consulter la tirade sur FSSTND vers la fin des instructions d'installation.

3 Compilation des programmes

3.1 Programmes ordinaires

Pour compiler un programme en ELF, utilisez `gcc` comme toujours. Pour compiler en a.out, utilisez `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
```

Le moment est peut-être venu de répondre à la question "si l'option par défaut des compilateurs a.out est de produire un programme appelé `a.out`, quel nom un compilateur ELF donne-t-il en sortie?". La réponse est toujours `a.out`.

3.2 Compilation des bibliothèques

Pour compiler `libfoo.so` en une bibliothèque partagée, les étapes fondamentales sont du genre:

```
$ 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
```

Cela générera une bibliothèque partagée appelée `libfoo.so.1.0`, et les liens appropriés pour que ld (`libfoo.so`) et l'éditeur dynamique de liens (`libfoo.so.1`) la trouvent. Comme test, ajoutons le répertoire courant à `LD_LIBRARY_PATH`.

Quand vous êtes satisfaits du fonctionnement de la bibliothèque, vous pourrez la déplacer vers `/usr/local/lib` par exemple, et recréer les liens appropriés. Notez que le lien `libfoo.so` devrait pointer vers `libfoo.so.1`, ainsi n'est-il pas nécessaire de le mettre à jour à chaque changement du numéro mineur de version. Le lien de `libfoo.so.1` vers `libfoo.so.1.0` est gardé à jour par `ldconfig` que la plupart des systèmes exécutent lors du processus d'amorçage.

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

3.3 Compiler en a.out

Vous pourriez avoir besoin de continuer à compiler des programmes dans l'ancien format a.out. Pour les programmes normaux, tout ce que vous avez à faire pour utiliser le compilateur a.out est de spécifier l'option `-b i486-linuxaout` quand vous appellerez gcc, et `-m i386linux` quand (si) vous appellerez ld. Si vous devez toujours compiler des bibliothèques partagées DLL a.out, vous avez toute ma sympathie. A ce que j'en sais, la réponse la plus courte est que cela ne marche pas. Envoyez-moi un mail si vous pensez différemment.

4 Patches et exécutables

Arrivé à ce point, vous pouvez, si vous le voulez, vous arrêter. Vous avez installé tout ce qui est nécessaire à la compilation et à l'exécution de programmes ELF.

Vous pourriez cependant vouloir recompiler des programmes en ELF, que ce soit par amour du travail bien fait ou pour minimiser l'usage de la mémoire. Pour la plupart des applications destinées à l'utilisateur final, c'est plutôt simple. Cependant certains paquetages se reposent trop sur le système sur lequel ils tournent et pourraient planter pour l'une ou plusieurs des raisons qui suivent:

- Conventions différentes pour le soulignement dans l'assembleur: dans un exécutable a.out, les labels externes reçoivent le préfixe _; pas en ELF. Cela ne fait pas de différence à moins que vous n'intégriez des instructions assembleur écrites à la main: toutes les étiquettes de la forme `_foo` doivent être traduites en `foo`, ou (si vous voulez conserver la portabilité) en `EXTERNAL(foo)` où `EXTERNAL` est une macro qui retourne soit son argument (si __ELF__ est défini) soit _ concaténé avec son argument dans le cas contraire.
- Différences de libc 5 par rapport à libc 4. L'interface pour le support local a changé, pour une fois.
- L'application ou le processus de compilation selon la connaissance du format binaire utilisé — emacs, par exemple, écrit l'image de sa mémoire sur le disque sous format binaire, vous avez donc besoin de savoir quel format vos fichiers compilés utilisent.
- L'application est ou comprend des bibliothèques partagées (X11 en est un exemple trivial). Il faudra de manière évidente faire des modifications pour adapter les différentes méthodes de création de bibliothèques partagées en ELF.

Voici maintenant deux listes: la première présente les programmes qui nécessitaient des modifications pour pouvoir être compilés sous ELF (c'est-à-dire dont vous devez obtenir une nouvelle version pour pouvoir les compiler en ELF) et la seconde comporte des programmes qui nécessitent toujours des patches extérieurs.

4.1 Mise à jour:

- **Dosemu**. Maintenant, dosemu tourne avec ELF. Les versions actuelles de dosemu sont disponibles sur `(ftp://tsx-11.mit.edu/pub/linux/ALPHA/dosemu/)`
- **e2fsutils**. Les utilitaires pour e2fs versions 0.5c et plus se compilent sans modifications en ELF.

- **Emacs**. Il y a potentiellement deux problèmes: (i) Emacs a une manière tout à fait particulière de se construire, à savoir qu'il faut en exécuter une version minimale, charger toutes les parties utiles en tant que lisp, et écrire l'image de sa mémoire sur le disque en tant que fichier binaire. (FSF) Emacs 19.29 et XEmacs 19.12 (anciennement Lucid Emacs) peuvent tous deux détecter que vous compilez en ELF et s'y adapter automatiquement. (ii) Si vous compilez une version de emacs sous ncurses, cela va rater à moins que vous n'éditiez au préalable `src/s/linux.h` dans la distribution emacs pour y ajouter la ligne `#define TERMINFO` quelque part près du début. Cela n'est pas nécessaire pour la version 19.31 mais l'est pour la version XEmacs 19.13. Il semblerait que cela soit réparé dans la 19.14.

- **gdb 4.16**. Votre copie actuelle de gdb continuera de fonctionner comme par le passé mais le support de la bibliothèque partagée dans la 4.16 est bien meilleur, donc si vous voulez déboguer des programmes qui manipulent bizarrement cette zone, c'est une mise à jour qui vaut le coup.

- **Le Noyau**. Les versions 2.0 du noyau (ou plus récentes) fonctionnent à merveille avec ELF; vous devez répondre "yes" à chacune des questions:

  ```
  Kernel support for ELF binaries (CONFIG_BINFMT_ELF) [Y/m/n/?]
  Compile kernel as ELF - if your GCC is ELF-GCC (CONFIG_KERNEL_ELF) [Y/n/?]
  ```

  quand vous exécutez `make config` (c'est également valable pour la plupart des noyaux de la série 1.3). Si vous utilisez encore le 1.2, veuillez consulter la liste qui suit.

- **perl 5**. Perl 5.001m et + se compilent sans modification sur un système ELF complet avec chargement dynamique. Les versions actuelles de Perl peuvent être obtenues sur le CPAN (Comprehensive Perl Archive Network): voir `(ftp://ftp.funet.fi/pub/mirrors/perl/CPAN)` pour le site le plus proche de chez vous.

- **ps** et **top**. Procps 0.98 et + marcheront avec ELF (les précédentes versions aussi mais elles ne sont pas en mesure de lire les WCHAN si vous en avez besoin). Prenez en considération que les noyaux de la série 2.0 ont de toute manière besoin de procps 0.99a ou plus.

- Le programme `cal` dans **util-linux 2.2** ne fonctionne pas. Passez à *version 2.5* `(ftp://tsx-11.mit.edu/pub/linux/packages/utils)` ou plus récent.

- **Mosaic**. Je ne suis pas en mesure de le compiler par moi-même mais l'exécutable Mosaic 2.7b1 est disponible par l'intermédiaire de NCSA en ELF. Il a été lié sous une configuration X étrange, avec pour résultat sur les systèmes normaux qu'il dit ne pas trouver `libXpm.so.4.5`. Une manière simple de réparer cela est de l'éditer avec précaution avec emacs ou une autre éditeur qui prend en charge les fichiers exécutables. Cherchez l'occurrence de la chaîne `libXpm.so.4.5^@` (où `^@` est un zéro ASCII — caractère NUL)), effacez le `.5` et ajoutez deux caractères de plus après le NUL pour ne pas modifier la longueur du fichier.

4.2 Patch

- **file**. Il fonctionne de toute manière mais peut être amélioré: `(ftp://ftp.uk.linux.org/pub/Linux/libc/non-core/file.patch)`.

– `make-3.74` — soit vous récupérez le code source sur un site GNU et appliquez le patch qui accompagne libc-5.3.12, soit vous récupérez l'exécutable `make-3.74.gz` sur `tsx-11`. Il y a un bogue dans le make de GNU qui ne se manifeste qu'avec les nouvelles versions ELF de libc — cela vient en fait d'un bogue dans les anciennes versions de la libc de GNU, qui était aussi présent dans la libc Linux jusqu'à peu. Si vous conservez votre vieux programme `make` en a.out, il continuera de fonctionner mais si vous en voulez une version ELF il vous faut le patch.

  Les développeurs GNU de Make sont au courant du bogue et devraient un jour sortir une version corrigée.

– **Le noyau 1.2.x** . 3 options s'offrent à vous:

  1. patcher légèrement le Makefile de manière à utiliser le compilateur a.out. `cd /usr/src/linux/`, coupez le patch qui suit et mettez-le dans `patch -p1`. Ou encore éditez le Makefile manuellement en utilisant ce qui suit comme guide; c'est suffisamment clair (effacez les lignes comportant un `-` et rajoutez les ligne avec un `+`.

```
diff -u linux-1.2.13/Makefile.orig linux/Makefile
--- linux-1.2.13/Makefile.orig  Wed Aug 16 20:53:26 1995
+++ linux/Makefile      Fri Dec  8 16:19:49 1995
@@ -12,9 +12,9 @@
 TOPDIR := $(shell if [ "$$PWD" != "" ]; then echo $$PWD; else pwd; fi)

-AS     =as
-LD     =ld
-HOSTCC =gcc -I$(TOPDIR)/include
-CC     =gcc -D__KERNEL__ -I$(TOPDIR)/include
+AS     =/usr/i486-linuxaout/bin/as
+LD     =ld  -m i386linux
+HOSTCC =gcc -b i486-linuxaout -I$(TOPDIR)/include
+CC     =gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
 MAKE   =make
 CPP    =$(CC) -E
 AR     =ar
```

  Autrement,

  2. Appliquez le patch de H J Luqui permet de compiler le noyau en ELF (et ajoute également comme possibilité de faire des core dumps ELF). On peut le trouver à (`ftp://ftp.cdrom.com/pub/linux/slackware_source/kernel-source/v1.2/linuxelf-1.2.13.diff.gz`).

     Si vous utilisez une distribution ELF (RedHat 2.1, Slackware 3) qui est fournie avec un noyau de la série 1.2, vous vous rendrez certainement compte que ce patch ou un qui lui est similaire a déjà été appliqué.

     La meilleure idée est cependant de probablement

  3. Passer à la 2.0! 1.2 n'a jamais été prévu pour ELF après tout.

Un autre problème va se poser pour la compilation du 1.2.13 avec gcc 2.7.2 et supérieur; il y avait un bogue dans `asm/io.h` qui est seulement détecté par gcc 2.7.2. Vous devez vous munir du patch (`ftp://ftp.uk.linux.org/pub/Linux/libc/misc/io.h`).

5 Informations complémentaires

- *GCC-HOWTO* (GCC-HOWTO.html) comporte un grand nombre d'informations utiles quant au développement de Linux (c'est du moins ce que je pense, j'en suis le régisseur). Il devrait pouvoir être obtenu au même endroit que là où vous avez trouvé ce HOWTO, c'est pourquoi le lien ci-dessus est un lien relatif.
- La liste de diffusion `linux-gcc` (qui est également le newsgroup `linux.dev.gcc`, si vous avez un filtre de news `linux.*`) est le meilleur endroit pour se tenir au courant de ce qui se passe, même sans y poster. N'oubliez pas que ce n'est pas Usenet: gardez vos questions à moins que vous ne fassiez du développement. Pour savoir comment rejoindre cette liste de diffusion, envoyez un message contenant le mot `help` à `majordomo@vger.rutgers.edu`. Les archives de cette liste sont dans (http://www.linux.ncm.com/linux-gcc/).
- Il y a une certaine masse d'information relative à ce que fait cette liste de diffusion à l'adresse *linux-gcc web page* (http://ftp.uk.linux.org/ barlow/linux/gcc-list.html), quand je pense à la mettre à jour. Vous y trouverez également un lien vers la dernière version de ce HOWTO et les patches auquels il se réfère. Pour les personnes des USA et ceux qui ont de mauvais accès sur les sites universitaires de Grande-Bretagne (à savoir quiconque est en dehors de l'université), il y a un miroir à (http://www.blackdown.org/elf/elf.html)
- D'autres documents sur le format de fichiers sont sur *tsx-11* (ftp://tsx-11.mit.edu/pub/linux/packages/GCC/ELF.doc.tar.gz). Ce sera probablement d'une grande utilité pour les personnes qui veulent comprendre, déboguer ou bien réécrire des programmes qui ont directement à voir avec des objets exécutables.
- Le document de H J Lu *ELF: From The Programmer's Perspective* (ftp://tsx-11.mit.edu/pub/linux/packages/GCC/elf.latex.tar.gz) contient un grand nombre d'informations très utiles et plus détaillées quant à la programmation avec ELF. Si vous n'avez pas de capacités LaTeX, il existe également au format PostScript.
- Des informations sur la bibliothèque **ncurses** et la base de données terminfosont disponibles sur *Eric Raymond's ncurses resource page* (http://www.ccil.org/ esr/ncurses.html).
- Une page de manuel traite de `dlopen(3)` et des fonctions afférentes, elle est fournie avec le paquetage `ld.so`.

6 Généralités

6.1 Comment me contacter

Envoyez moi un mail à daniel.barlow@linux.org. Ma clé publique PGP (ID 5F263625) se trouve sur mes *pages web* (http://ftp.uk.linux.org/ barlow/), si vous tenez à confidentialité.

Pour toute question auquel ce document aurait du répondre mais ne l'a pas fait, écrivez moi. Si vous avez une question qui n'a pas sa place ici mais dont vous pensez que j'en possède la réponse, veuillez tout d'abord la poster sur le newsgroup `comp.os.linux.*` approprié; je réponds généralement au mail mais il m'est arrivé d'oublier.

Quiconque s'autoriserait à me rajouter à des listes d'adresse publicitaires s'expose à de graves conséquences.

6.2 Traductions

Si vous désirez traduire ce document, faites-le mais parlez m'en! Les chances que je parle la langue dans laquelle vous voulez le traduire sont tristement faibles mais cela mis à part, je serai heureux de vous aider de quelque manière que ce soit.

Les traductions dont j'ai connaissance sont:

- *Italian* (http://www.psico.unipd.it/ildp/docs/HOWTO/ELF-HOWTO.html) par Favro Renata. (d'autres HOWTOs sont disponibles en italien sur (http://www.psico.unipd.it/ildp/docs/HOWTO/INDEX.html).
- Kojima Mitsuhiro a fait une traduction japonaise, disponible sur (http://jf.gee.kyoto-u.ac.jp/JF/index.html).
- Traduction de Pierre Tane (ftp://ftp.lip6.fr/pub/linux/french)

6.3 Notions légales