GCC HOWTO pour Linux

par Daniel Barlow <dan@detached.demon.co.uk> v1.17, 28 février 1996

(Adaptation française par Eric Dumas <dumas@freenix.fr>, 8 Avril 1996). Ce document présente la manière de configurer le compilateur GNU C et les bibliothèques de développement sous Linux. Il donne un aperçu de la compilation, de l'édition de liens, de l'exécution et du débogage de programmes sous Linux. Bon nombre de passages de ce document sont empruntés à la FAQ GCC rédigée par Mitch D'Souza's et au HowTo ELF. Ceci est la première version publique (en dépit du numéro de version : en fait, ça vient de RCS). N'hésitez pas à me joindre pour toute remarque.

1 Préliminaires

1.1 ELF et a.out

Le développement de Linux est actuellement dans une phase de transition. En résumé, il existe deux formats de binaires que Linux reconnaît et exécute, et cela dépend de la manière dont votre système est configuré : vous pouvez avoir les deux, l'un ou l'autre. En lisant ce document, vous pourrez savoir quels binaires votre système est capable de gérer.

Comment le savoir ? Utilisez la commande `file` (par exemple, `file /bin/bash`). Pour un programme ELF, cette commande va vous répondre quelque chose dans lequel se trouve le mot ELF. Dans le cas d'un programme en a.out, il vous indiquera quelque chose comme `Linux/i386`.

Les différences entre ELF et a.out sont détaillées plus tard dans ce document. ELF est le nouveau format et il est considéré comme étant meilleur.

1.2 Du côté du copyright

Le copyright et autres informations légales peuvent être trouvés à la *fin* de ce document, avec les avertissements conventionnels concernant la manière de poser des questions sur Usenet pour éviter d'avoir à révéler votre ignorance du langage C en annonçant des bogues qui n'en sont pas, etc.

1.3 Typographie

Si vous lisez ce document au format Postscript, dvi, ou HTML, vous pouvez voir quelques différence entre les styles d'écriture alors que les gens qui consultent ce document au format texte pur ne verront aucune différence. En particulier, les noms de fichiers, le nom des commandes, les messages donnés par les programmes et les codes sources seront écrits avec le style suivant : `style d'écriture`, alors que les noms de variables entre autres choses seront en *italique*.

Vous aurez également un index. Avec les formats dvi ou postscript, les chiffres dans l'index correspondent au numéros de paragraphes. Au format HTML, il s'agit d'une numérotation séquentielle pour que vous puissiez cliquer dessus. Avec le format texte, ce ne sont que des nombres. Il vous est donc conseillé de prendre un autre format que le format texte !

L'interpréteur de commande (*shell*) utilisé dans les exemples sera la Bourne shell (plutôt que le C-Shell). Les utilisateurs du C-Shell utiliseront plutôt :

```
% setenv soif JD
```

là où j'ai écrit

```
$ soif=JD; export soif
```

Si l'invite (*prompt* dans la langue de Shakespeare) est `#` plutôt que `$`, la commande ne fonctionnera que si elle est exécutée au nom de Root. Bien sur, je décline toute responsabilité de ce qui peut se produire sur votre système lors de l'exécution de ces exemples. Bonne chance `:-)`

2 Où récupérer de la documentation et les programmes ?

2.1 Ce document

Ce document fait partie de la série des HOWTO pour Linux, et il est donc disponible ainsi que ces collègues dans les répertoires HowTo pour Linux, comme sur `http://sunsite.unc.edu/pub/linux/docs/HOWTO/` . La version HTML peut également être consultée sur `http://ftp.linux.org.uk/~barlow/howto/gcc-howto.html` .

Note du traducteur : vous pouvez obtenir tous les HowTos en langue anglaise et française sur `ftp.ibp.fr:/pub/linux`. Les versions françaises se trouvent dans le répertoire `/pub/linux/french/HOWTO`.

2.2 Autres documentation

La documentation officielle pour gcc se trouve dans les sources de la distribution (voir plus bas) sous la forme de fichiers texinfo et de fichiers `.info`. Si vous possédez une connexion rapide, un CD-ROM ou une certaine patience, vous pouvez désarchiver la documentation et l'installer dans le répertoire `/usr/info`. Sinon, vous pouvez toujours les trouver sur *tsx-11* `ftp://tsx-11.mit.edu:/pub/linux/packages/GCC/` , mais ce n'est pas nécessairement toujours la dernière version.

Il existe deux sources de documentation pour la libc. La libc GNU est fournie avec des fichiers info qui décrivent assez précisément la libc Linux sauf pour la partie des entrées-sorties. Vous pouvez également trouver sur *sunsite* `ftp://sunsite.unc.edu/pub/Linux/docs/` des documents écrits pour Linux ainsi que la description de certaines appels systèmes (section 2) et certaines fonctions de la libc (section 3).

Note du traducteur : un bémol concernant cette partie... La libc Linux n'est pas GNU et tend à être relativement différente sur certains points.

2.3 GCC

Il existe deux types de réponses

(a) La distribution officielle de GCC pour Linux peut toujours être récupérée sous la forme de binaires (déjà compilée) sur `ftp://tsx-11.mit.edu:/pub/linux/packages/GCC/` . Vous pouvez la trouver sur le miroir français `ftp://ftp.ibp.fr:/pub/linux/packages/GCC/` . A l'heure où j'écris ces lignes, la dernière version est gcc 2.7.2 (`gcc-2.7.2.bin.tar.gz`).

(b) La dernière distribution des sources de GCC de la *Free Software Foundation* peut-être récupérée sur

prep.ai.mit.edu `ftp://prep.ai.mit.edu/pub/gnu/` ou

ftp.ibp.fr `ftp://ftp.ibp.fr/pub/gnu/` . Ce n'est pas toujours la même version que celle présentée ci-dessus. Les mainteneurs de GCC pour Linux ont rendu la compilation de GCC plus facile grâce à l'utilisation du script `configure` qui effectue la configuration d'une manière automatique. Regardez dans *tsx-11* `ftp://tsx-11.mit.edu:/pub/linux/packages/GCC/` ou *ftp.ibp.fr* `ftp://ftp.ibp.fr:/pub/linux/packages/GCC/`

pour récupérer d'éventuels patches.

Quelle que soit la complexité de votre programme, vous aurez également besoin de la *libc*.

2.4 Les fichiers d'en-tête et la bibliothèque C

Ce que vous allez trouver dans ce paragraphe dépend

- de votre système (ELF ou a.out) ;
- du type de binaire que vous désirez générer.

Si vous êtes en train de mettre à jour votre libc 4 en libc 5, vous devriez consulter le ELF HowTo qui se trouve au même endroit que ce document.

Les libc sont disponibles sur *tsx-11* `ftp://tsx-11.mit.edu:/pub/linux/packages/GCC/` ou

ftp.ibp.fr `ftp://ftp.ibp.fr:/pub/linux/packages/GCC/` . Voici une description des fichiers situés dans ce répertoire :

`libc-5.2.18.bin.tar.gz`

— bibliothèques dynamiques et statiques ELF plus les fichiers d'en-tête pour la bibliothèque C et la bibliothèque mathématique.

`libc-5.2.18.tar.gz`

— Code source pour la bibliothèque ci-dessus. Vous aurez également besoin du paquetage `.bin.` pour avoir les fichiers d'en-tête. Si vous hésitez entre compiler la bibliothèque C vous-même et utiliser les binaires, la bonne réponse est dans la majorité des cas est d'utiliser les binaires. Toutefois, si vous désirer utiliser NYS (NdT : NYS != NIS) ou bien les mots de passe *shadow*, vous devrez recompiler la libc par vous-même.

`libc-4.7.5.bin.tar.gz`

— bibliothèques dynamiques et statiques a.out pour la version 4.7.5 de la libc. Cette bibliothèque a été conçue pour pouvoir coexister avec le paquetage de la libc 5 décrit ci-dessus, mais c'est uniquement nécessaire si vous désirez utiliser ou développer des programmes au format a.out.

2.5 Outils associés (as, ld, ar, strings, etc.)

Ces outils se trouvent comme les bibliothèques dans le répertoire

tsx-11 `ftp://tsx-11.mit.edu:/pub/linux/packages/GCC/` , et

ftp.ibp.fr `ftp://ftp.ibp.fr:/pub/linux/packages/GCC/` . La version actuelle est `binutils-2.6.0.2.bin.tar.gz`.

Il est utile de remarquer que ces outils ne sont disponibles qu'au format ELF, que la libc actuelle est ELF et que la libc a.out ne pose pas de problème lorsqu'elle est utilisée avec la libc ELF. Le développement de la libc est relativement rapide et à moins que n'ayez de bonnes raisons pour utiliser le format a.out, vous êtes encouragés à suivre le mouvement.

3 Installation et configuration de GCC

3.1 Les versions de GCC

Vous pouvez savoir quelle est la version de GCC que vous possédez en tapant `gcc -v` lors de l'invite. C'est également une bonne technique pour savoir si votre configuration est ELF ou a.out. Sur mon système, cela donne ceci :

```
$ gcc -v
Reading specs from /usr/lib/gcc-lib/i486-zorglub-linux/2.7.2/specs
gcc version 2.7.2
```

Les mots-clefs à remarquer

- `i486`. Cela vous indique que la version de gcc que vous utilisez a été compilée pour être utilisée sur un processeur 486 — mais vous pouvez avoir un autre processeur comme un 386 ou un Pentium (586). Tous ces processeurs peuvent exécuter le code compilé avec n'importe quel processeur. La seule différence réside dans le fait que le code 486 rajoute un peu de code à certains endroits pour aller plus vite sur un 486. Cela n'a pas d'effet néfaste côté performance sur un 386 mais cela rend les exécutables un peu plus importants.
- `zorglub`. Ce n'est pas réellement important, et il s'agit généralement d'un commentaire (comme `slackware` or `debian`) ou même, cela peut-être vide (lorsque vous avez comme nom de répertoire `i486-linux`). Si vous construisez votre propre gcc, vous pouvez fixer ce paramètre selon vos désirs, comme je l'ai fait. `:-)`
- `linux`. Cela peut être à la place `linuxelf` ou `linuxaout` et en fait, la signification varie en fonction de la version que vous possédez.
  - `linux` signifie ELF si la version est 2.7.0 ou supérieure, sinon, c'est du a.out.
  - `linuxaout` signifie a.out. Cela a été introduit comme cible lorsque le format des binaires a changé de a.out vers ELF dans **Linux**. Normalement, vous ne verrez plus de `linuxaout` avec une version de gcc supérieure à 2.7.0.
  - `linuxelf` est dépassé. Il s'agit généralement de gcc version 2.6.3 configuré pour générer des exécutables ELF. Notez que gcc 2.6.3 est connu pour générer de nombreuses erreurs lorsqu'il produit du code ELF — une mise à jour est très fortement recommandée.
- `2.7.2` est le numéro de la version de GCC.

Donc, en résumé, nous possédons gcc 2.7.2 qui génère du code ELF. *Quelle surprise* (NdT: En français dans le texte) !

3.2 A quel endroit s'installe GCC ?

Si vous avez installé gcc sans regarder, ou bien si vous l'avez eu à partir d'une distribution, vous pouvez avoir envie de savoir où il se trouve dans votre arborescence. Les mots clefs permettant cela sont

- `/usr/lib/gcc-lib/`*machine-cible*`/`*version*`/` (et ses sous-répertoires) est généralement l'endroit où se trouve le plus souvent le compilateur. Ceci inclut les exécutables qui réalisent la compilation ainsi que certaines bibliothèques et quelques fichiers d'en-tête.

- `/usr/bin/gcc` est le lanceur du compilateur — c'est en fait le programme que vous lancez. Il peut être utilisé avec plusieurs versions de gcc lorsque vous possédez plusieurs répertoires installés (voir plus bas). Pour trouver la version par défaut utilisée, lancez `gcc -v`. Pour forcer l'utilisation d'une autre version, lancez `gcc -V` *version*. Par exemple,

```
# gcc -v
Reading specs from /usr/lib/gcc-lib/i486-zorglub-linux/2.7.2/specs
gcc version 2.7.2
# gcc -V 2.6.3 -v
Reading specs from /usr/lib/gcc-lib/i486-zorglub-linux/2.6.3/specs
gcc driver version 2.7.2 executing gcc version 2.6.3
```

- `/usr/`*machine-cible*`/(bin|lib|include)/`. Si vous avez installé plusieurs cibles possibles (par exemple a.out et elf, ou bien un compilateur croisé, les bibliothèques, les binutils (`as`, `ld`, etc.) et les fichiers d'en-tête pour les cibles différente de celle par défaut peuvent être trouvés à cet endroit. Même si vous n'avez qu'une seule version de gcc installée, vous devriez toutefois trouver à cet endroit un certain nombre de fichiers. Si ce n'est pas la cas, regardez dans `/usr/(bin|lib|include)`.

- `/lib/`, `/usr/lib` et autres sont les répertoires pour les bibliothèques pour le système initial. Vous aurez également besoin du programme `/lib/cpp` pour un grand nombre d'applications (X l'utilise beaucoup) — soit vous le copiez à partir de `/usr/lib/gcc-lib/`*machine-cible*`/`*version*`/`, soit vous faites pointer un lien symbolique dessus.

3.3 Où se trouvent les fichiers d'en-tête ?

Si l'on excepte les fichier fichiers d'en-tête que vous installez dans le répertoire `/usr/local/include`, il y a en fait trois types de fichiers d'en-tête :

- La grande majorité des fichiers situés dans le répertoire `/usr/include/` et dans ses sous-répertoires proviennent du paquetage de la libc dont s'occupe H.J. Lu. Je dis bien la "grande majorité" car vous pouvez avoir également certains fichiers provenant d'autres sources (par exemple des bibliothèques `curses` et `dbm`), ceci est d'autant plus vrai si vous possédez une distribution de la libc récente (où les bibliothèques curses et dbm ne sont pas intégrées).

- Les répertoires `/usr/include/linux` et `/usr/include/asm` (pour les fichiers `<linux/*.h>` et `<asm/*.h>`) doivent être des liens symboliques vers les répertoires `linux/include/linux` et `linux/include/asm` situés dans les sources du noyau. Vous devrez installer ces sources si vous désirez pouvoir développer : ces sources ne sont pas utilisés uniquement pour compiler le noyau.

  Il est probable que vous ayez besoin de lancer la commande suivante `make config` dans le répertoire des sources du noyau après les avoir installés. Beaucoup de fichiers ont besoin du fichier d'en-tête `<linux/autoconf.h>` qui n'existe pas sans cette commande. Il est à noter que dans certaines versions du noyau, le répertoire `asm` est en fait un lien symbolique qui n'est créé qu'avec l'exécution de `make config`.

  Donc, si vous installez les sources du noyau dans le répertoire `/usr/src/linux`, il suffit de faire :

```
$ cd /usr/src/linux
$ su
# make config
[repondez aux questions. A moins que vous ne recompiliez votre
noyau, les reponses importent peu]
# cd /usr/include
# ln -s ../src/linux/include/linux .
# ln -s ../src/linux/include/asm .
```

- Les fichiers tels que <float.h>, <limits.h>, <varargs.h>, <stdarg.h> et <stddef.h> changent en fonction de la version du compilateur, et peuvent être trouvés dans le répertoire /usr/lib/gcc-lib/i486-box-linux/2.7.2/include/ pour la version 2.7.2.

3.4 Construire un compilateur croisé

3.4.1 Linux comme plate-forme de destination

Nous supposons que vous avez récupéré les sources de gcc, et normalement, il vous suffit de suivre les instructions données dans le fichier INSTALL situé dans les sources de gcc. Ensuite, il suffit de lancer configure -target=i486-linux -host=XXX sur une plateforme XXX, puit un make devrait compiler gcc correctement. Il est à noter que vous aurez besoin des fichiers d'en-tête de Linux, ainsi que les sources de l'assembleur et du l'éditeur de liens croisés que vous pouvez trouver sur ftp://tsx-11.mit.edu/pub/linux/packages/GCC/ ou

ftp://ftp.ibp.fr/pub/linux/GCC/ .

3.4.2 Linux comme plate-forme origine et MSDOS comme destination

Arggg. Apparemment, cela est possible en utilisant le paquetage « emx » ou l'extension « go ». Regardez ftp://sunsite.unc.edu/pub/Linux/devel/msdos pour plus d'informations.

Je n'ai pas testé cela et je ne pense pas le faire !

4 Portage et compilation

4.1 Symboles définis automatiquement

Vous pouvez trouver quels symboles votre version de gcc définit automatiquement en le lançant avec l'option -v. Par exemple cela donne ça chez moi :

```
$ echo 'main(){printf("Bonjour !\n");}' | gcc -E -v -
Reading specs from /usr/lib/gcc-lib/i486-box-linux/2.7.2/specs
gcc version 2.7.2
 /usr/lib/gcc-lib/i486-box-linux/2.7.2/cpp -lang-c -v -undef
-D__GNUC__=2 -D__GNUC_MINOR__=7 -D__ELF__ -Dunix -Di386 -Dlinux
-D__ELF__ -D__unix__ -D__i386__ -D__linux__ -D__unix -D__i386
-D__linux -Asystem(unix) -Asystem(posix) -Acpu(i386)
-Amachine(i386) -D__i486__ -
```

Si vous écrivez du code qui utilise des spécificités Linux, il est souhaitable d'implémenter le code non portable de la manière suivante

```
#ifdef __linux__
/* ... code linux ... */
#endif /* linux */
```

Utilisez __linux__ pour cela, et *pas* linux. Bien que cette macro soit définie, ce n'est pas une spécification POSIX.

4.2 Options de compilation

La documentation des options de compilation se trouve dans les pages *info* de gcc (sous Emacs, utilisez `C-h i` puis sélectionnez l'option 'gcc'). Votre distribution peut ne pas avoir installé la documentation ou bien vous pouvez en avoir une ancienne. Dans ce cas, la meilleure chose à faire est de récupérer les sources de gcc depuis `ftp://prep.ai.mit.edu/pub/gnu` ou l'un des ses nombreux miroirs dont `ftp://ftp.ibp.fr/pub/gnu` .

La page de manuel gcc (`gcc.1`) est en principe, complètement dépassée. Cela vous met en garde si vous désirez la consulter.

4.2.1 Options de compilation

gcc peut réaliser un certain nombre d'optimisations sur le code généré en ajoutant l'option `-O`*n* à la ligne de commandes, où *n* est un chiffre. La valeur de *n*, et son effet exact, dépend de la version de gcc, mais s'échelonne normalement entre 0 (aucune optimisation) et 2 (un certain nombre) ou 3 (toutes les optimisations possibles).

En interne, gcc interprète les options telles que `-f` et `-m`. Vous pouvez voir exactement ce qu'effectue le niveau spécifié dans l'option `-O` en lançant gcc avec l'option `-v` et l'option (non documentée) `-Q`. Par exemple, l'option `-O2`, effectue les opérations suivantes sur ma machine :

```
enabled: -fdefer-pop -fcse-follow-jumps -fcse-skip-blocks
-fexpensive-optimizations
         -fthread-jumps -fpeephole -fforce-mem -ffunction-cse -finline
         -fcaller-saves -fpcc-struct-return -frerun-cse-after-loop
         -fcommon -fgnu-linker -m80387 -mhard-float -mno-soft-float
         -mno-386 -m486 -mieee-fp -mfp-ret-in-387
```

Utiliser un niveau d'optimisation supérieur à celui que le compilateur supporte (par exemple `-O6`) aura le même effet qu'utiliser le plus haut niveau géré. Distribuer du code où la compilation est configurée de cette manière est une très mauvaise idée – si d'autres optimisations sont incorporées dans de versions futures, vous (ou d'autres utilisateurs) pouvez vous apercevoir que cela ne compile plus, ou bien que le code généré ne fait pas les actions désirées.

Les utilisateurs de gcc 2.7.0 à 2.7.2 devraient noter qu'il y a un bogue dans l'option `-O2`. Plus précisément, la *strength reduction* ne fonctionne pas. Un patch a été implémenté pour résoudre ce problème, mais vous devez alors recompiler gcc. Sinon, vous devrez toujours compiler avec l'option `-fno-strength-reduce`.

Spécification du processeur Il existe d'autres options `-m` qui ne sont pas positionnées lors de l'utilisation de `-O` mais qui sont néanmoins utiles dans certains cas. C'est le cas pour les options `-m386` et `-m486`, qui indiquent à gcc de générer un code plus ou moins optimisé pour l'un ou l'autre type de processeur. Le code continuera à fonctionner sur les deux processeurs. Bien que le code pour 486 soit plus important, il ne ralentit pas l'exécution du programme sur 386.

Il n'existe pas actuellement de `-mpentium` ou `-m586`. Linus a suggéré l'utilisation des options `-m486 -malign-loops=2 -malign-jumps=2 -malign-functions=2`, pour exploiter les optimisations du 486 tout en perdant de la place due aux problèmes d'alignements (dont le Pentium n'a que faire). Michael Meissner (de Cygnus) nous dit :

> « Mon avis est que l'option `-mno-strength-reduce` permet d'obtenir un code plus rapide sur un x86 (nota : je ne parle pas du bogue *strength reduction*, qui est un autre problème). Cela s'explique en raison du peu de registres dont disposent ces processeurs (et la méthode de GCC qui consiste à grouper les registres dans l'ordre inverse au lieu d'utiliser d'autres registres

> n'arrange rien). La *strength reduction* consiste en fait à rajouter des registres pour remplacer les multiplications par des additions. Je suspecte également `-fcaller-saves` de ne pas arranger la situation. »
>
> Une autre idée est que `-fomit-frame-pointer` n'est pas obligatoirement une bonne idée. D'un côté, cela peut signifier qu'un autre registre est disponible pour une allocation. D'un autre côté, vue la manière dont les processeurs x86 codent leur jeu d'instruction, cela peut signifier que la pile des adresses relatives prend plus de place que les adresses de fenêtres relatives, ce qui signifie en clair que moins de cache est disponible pour l'exécution du processus. Il faut préciser que l'option `-fomit-frame-pointer`, signifie que le compilateur doit constamment ajuster le pointeur de pile après les appels, alors qu'avec une fenêtre, il peut laisser plusieurs appels dans la pile.

Le mot final sur le sujet provient de Linus :

> Remarquez que si vous voulez des performances maximales, ne me croyez pas : testez ! Il existe tellement d'options de gcc, et il est possible que cela ne soit une réelle optimisation que pour vous.

4.2.2 `Internal compiler error: cc1 got fatal signal 11`

Signal 11 correspond au signal SIGSEGV, ou bien *segmentation violation*. Normalement, cela signifie que le programme s'est mélangé les pointeurs et a essayé d'écrire là où il n'en a pas le droit. Donc, cela pourrait être un bug de gcc.

Toutefois, gcc est un logiciel assez testé et assez remarquable de ce côté. Il utilise un grand nombre de structures de données complexes, et un nombre impressionnant de pointeurs. En résumé, c'est le plus pointilleux des testeurs de mémoire existants. Si vous *n'arrivez pas à reproduire le bogue* — si cela ne s'arrête pas au même endroit lorsque vous retentez la compilation — c'est plutôt un problème avec votre machine (processeur, mémoire, carte mère ou bien cache). **N'annoncez pas** la découverte d'un nouveau bogue si votre ordinateur traverse tous les tests du BIOS, ou s'il fonctionne correctement sous Windows ou autre : ces tests ne valent rien. Il en va de même si le noyau s'arrête lors du '`make zImage`' ! '`make zImage`' doit compiler plus de 200 fichiers, et il en faut bien moins pour arriver à faire échouer une compilation.

Si vous arrivez à reproduire le bogue et (mieux encore) à écrire un petit programme qui permet de mettre en évidence cette erreur, alors vous pouvez envoyer le code soit à la FSF, soit dans la liste linux-gcc. Consultez la documentation de gcc pour plus de détails concernant les informations nécessaires.

4.3 Portabilité

Cette phrase a été dite un jour : si quelque chose n'a pas été porté vers Linux alors ce n'est pas important de l'avoir :-).

Plus sérieusement, en général seules quelques modifications mineures sont nécessaires car Linux répond à 100% aux spécifications POSIX. Il est généralement sympathique d'envoyer à l'auteur du programme les modifications effectuées pour que le programme fonctionne sur Linux, pour que lors d'une future version, un 'make' suffise pour générer l'exécutable.

4.3.1 Spécificités BSD (notamment `bsd_ioctl`, `daemon` et `<sgtty.h>`)

Vous pouvez compiler votre programme avec l'option `-I/usr/include/bsd` et faire l'édition de liens avec `-lbsd` (en ajoutant `-I/usr/include/bsd` à la ligne `CFLAGS` et `-lbsd` à la ligne `LDFLAGS` dans votre fichier

`Makefile`). Il est également nécessaire de ne **pas** ajouter `-D__USE_BSD_SIGNAL` si vous voulez que les signaux BSD fonctionnent car vous les avez inclus automatiquement avec la ligne `-I/usr/include/bsd` et en incluant le fichier d'en-tête <`signal.h`>.

4.3.2 Signaux *manquants* (`SIGBUS`, `SIGEMT`, `SIGIOT`, `SIGTRAP`, `SIGSYS`, etc.)

Linux respecte les spécifications POSIX. Ces signaux n'en font pas partie (cf. ISO/IEC 9945-1:1990 - IEEE Std 1003.1-1990, paragraphe B.3.3.1.1) :

> « Les signaux SIGBUS, SIGEMT, SIGIOT, SIGTRAP, et SIGSYS ont été omis de la norme POSIX.1 car leur comportement est dépendant de l'implémentation et donc ne peut être répertorié d'une manière satisfaisante. Certaines implémentations peuvent fournir ces signaux mais doivent documenter leur effet »

La manière la plus élégante de régler ce problème est de redéfinir ces signaux à `SIGUNUSED`. La manière *normale* de procéder est d'entourer le code avec les `#ifdef` appropriés :

```
#ifdef SIGSYS
/* ... code utilisant les signaux non posix  .... */
#endif
```

4.3.3 Code K & R

GCC est un compilateur ANSI, or il existe beaucoup de code qui ne soit pas ANSI.

Il n'y a pas grand chose à faire, sauf rajouter l'option `-traditional` lors de la compilation. Il effectue certaines vérifications supplémentaires. Consultez les pages info gcc.

Notez que l'option `-traditional` a pour unique effet de changer la forme du langage accepté par gcc. Par exemple, elle active l'option `-fwritable-strings`, qui déplace toutes les chaînes de caractères vers l'espace de données (depuis l'espace de texte, où elle ne peuvent pas être modifiées). Ceci augmente la taille de la mémoire occupée par le programme.

4.3.4 Les symboles du préprocesseur produisent un conflit avec les prototypes du code

Un des problèmes fréquents se produit lorsque certaines fonctions standards sont définies comme macros dans les fichiers d'en-tête de Linux et le préprocesseur refusera de traiter des prototypes identiques. Par exemple, cela peut arriver avec `atoi()` et `atol()`.

4.3.5 `sprintf()`

Parfois, soyez prudent lorsque vous effectuez un portage à partir des sources de programmes fonctionnant sous SunOs, surtout avec la fonction `sprintf(string, fmt, ...)` car elle renvoie un pointeur sur la chaîne de caractères alors que Linux (suivant la norme ANSI) retourne le nombre de caractères recopiés dans la chaîne de caractères.

4.3.6 `fcntl` et ses copains. Où se trouve la définition de `FD_*` et compagnie ?

Dans <`sys/time.h`>. Si vous utilisez `fcntl` vous voudrez probablement inclure <`unistd.h`> également, pour avoir le prototype de la fonction.

D'une manière générale, la page de manuel pour une fonction donne la liste des fichiers d'en-tête à inclure.

4.3.7 Le timeout de `select()`. Les programmes commencent dans un état d'attente active

A une certaine époque, le paramètre timeout de la fonction `select()` était utilisé en lecture seule. C'est pourquoi la page de manuel comporte une mise en garde :

> select() devrait retourner normalement le temps écoulé depuis le timeout initial, s'il s'est déclenché, en modifiant la valeur pointée par le paramètre `time`. Cela sera peut-être implémenté dans les versions ultérieures du système. Donc, il n'est pas vraiment prudent de supposer que les données pointées ne seront pas modifiées lors de l'appel à select().

Mais tout arrive avec le temps ! Lors d'un retour de `select()`, l'argument `timeout` recevra le temps écoulé depuis la dernière réception de données. Si aucune donnée n'est arrivée, la valeur sera nulle, et les futurs appels à cette fonction utilisant le même `timeout` auront pour résultat un retour immédiat.

Pour résoudre le problème, il suffit de mettre la valeur `timeout` dans la structure à chaque appel de `select()`. Le code initial était

```
struct timeval timeout;
timeout.tv_sec = 1;
timeout.tv_usec = 0;
while (some_condition)
      select(n,readfds,writefds,exceptfds,&timeout);
```

et doit devenir :

```
struct timeval timeout;
while (some_condition)
{
      timeout.tv_sec = 1;
      timeout.tv_usec = 0;
      select(n,readfds,writefds,exceptfds,&timeout);
}
```

Certaines versions de Mosaic étaient connues à une certaine époque pour avoir ce problème.

La vitesse de rotation du globe terrestre était inversement proportionnelle à la vitesse de transfert des données !

4.3.8 Appels systèmes interrompus

Symptomes : Lorsqu'un processus est arrêté avec un Ctrl-Z et relancé - ou bien lorsqu'un autre signal est déclenché dans une situation différente : par exemple avec un Ctrl-C, la terminaison d'un processus, etc, on dit qu'il y a « interruption d'un appel système » , ou bien « write : erreur inconnue » ou des trucs de ce genre.

Problèmes : Les systèmes POSIX vérifient les signaux plus souvent que d'autres Unix plus anciens. Linux peux lancer les gestionnaires de signaux :

- d'une manière asynchrone (sur un top d'horloge)
- lors d'un retour de n'importe quel appel système

- pendant l'exécution des appels systèmes suivants : `select()`, `pause()`, `connect()`, `accept()`, `read()` sur des terminaux, des sockets, des pipes ou des fichiers situés dans `/proc`, `write()` sur des terminaux, des sockets, des pipes ou des imprimantes, `open()` sur des FIFOs, des lignes PTYs ou séries, `ioctl()` sur des terminaux, `fcntl()` avec la commande `F_SETLKW`, `wait4()`, `syslog()`, et toute opération d'ordre TCP ou NFS.

Sur d'autres systèmes d'exploitation, il est possible que vous ayez à inclure dans cette catégorie les appels systèmes suivants : `creat()`, `close()`, `getmsg()`, `putmsg()`, `msgrcv()`, `msgsnd()`, `recv()`, `send()`, `wait()`, `waitpid()`, `wait3()`, `tcdrain()`, `sigpause()`, `semop()`.

Si un signal (que le programme désire traiter) est lancé pendant l'exécution d'un appel système, le gestionnaire est lancé. Lorsque le gestionnaire du signal se termine, l'appel système détecte qu'il a été interrompu et se termine avec la valeur -1 et `errno = EINTR`. Le programme n'est pas forcément au courant de ce qui s'est passé et donc s'arrête.

Vous pouvez choisir deux solutions pour résoudre ce problème.

(1)Dans tout gestionnaire de signaux que vous mettez en place, ajoutez l'option `SA_RESTART` au niveau de *sigaction*. Par exemple, modifiez

```
  signal (signal_id, mon_gestionnaire_de_signaux);
```

en

```
  signal (signal_id, mon_gestionnaire_de_signaux);
  {
        struct sigaction sa;
        sigaction (signal_id, (struct sigaction *)0, &sa);
#ifdef SA_RESTART
        sa.sa_flags |= SA_RESTART;
#endif
#ifdef SA_INTERRUPT
        sa.sa_flags &= ~ SA_INTERRUPT;
#endif
        sigaction (signal_id, &sa, (struct sigaction *)0);
  }
```

Notez que lors de certains appels systèmes vous devrez souvent regarder si `errno` n'a pas été positionnée à `EINTR` par vous même comme avec `read()`, `write()`, `ioctl()`, `select()`, `pause()` et `connect()`.

(2) A la recherche de `EINTR` :

Voici deux exemples avec `read()` et `ioctl()`,

Voici le code original utilisant `read()`

```
int result;
while (len > 0)
{
  result = read(fd,buffer,len);
  if (result < 0)
        break;
  buffer += result;
  len -= result;
}
```

et le nouveau code

```
int result;
while (len > 0)
{
  result = read(fd,buffer,len);
  if (result < 0)
  {
        if (errno != EINTR)
                break;
  }
  else
  {
        buffer += result;
        len -= result;
  }
}
```

Voici un code utilisant `ioctl()`

```
int result;
result = ioctl(fd,cmd,addr);
```

et cela devient

```
int result;
do
{
   result = ioctl(fd,cmd,addr);
}
while ((result == -1) && (errno == EINTR));
```

Il faut remarquer que dans certaines versions d'Unix de type BSD on a l'habitude de relancer l'appel système. Pour récupérer les interruptions d'appels systèmes, vous devez utiliser les options `SV_INTERRUPT` ou `SA_INTERRUPT`.

4.3.9 Les chaînes et leurs accès en écritures (ou les programmes qui provoquent des « segmentation fault » d'une manière aléatoire)

GCC a une vue optimiste en ce qui concerne ses utilisateurs, en croyant qu'ils respectent le fait qu'une chaîne dite constante l'est réellement. Donc, il les range dans la zone *texte(code)* du programme, où elles peuvent être chargées puis déchargées à partir de l'image binaire de l'exécutable située sur disque (ce qui évite d'occuper de l'espace disque). Donc, toute tentative d'écriture dans cette chaîne provoque un « segmentation fault ».

Cela peut poser certains problèmes avec d'anciens codes, par exemple ceux qui utilisent la fonction `mktemp()` avec une chaîne constante comme argument. `mktemp()` essaye d'écrire dans la chaîne passée en argument.

Pour résoudre ce problème,

1. compilez avec l'option `-fwritable-strings` pour indiquer à gcc de mettre les chaînes constantes dans l'espace de données

2. réécrire les différentes parties du code pour allouer une chaîne non constante puis effectuer un strcpy des données dedans avant d'effectuer l'appel.

4.3.10 Pourquoi l'appel à `execl()` échoue ?

Tout simplement parce que vous l'utilisez mal. Le premier argument d'`execl` est le programme que vous désirez exécuter. Le second et ainsi de suite sont en fait le éléments du tableau `argv` que vous appelez. Souvenez-vous que `argv[0]` est traditionnellement fixé même si un programme est lancé sans argument. Vous devriez donc écrire :

```
execl("/bin/ls","ls",NULL);
```

et pas

```
execl("/bin/ls", NULL);
```

Lancer le programme sans argument est considéré comme étant une demande d'affichage des bibliothèques dynamiques associées au programme, si vous utilisez le format a.out. ELF fonctionne d'une manière différente.

(Si vous désirez ces informations, il existe des outils plus simples; consultez la section sur le chargement dynamique, ou la page de manuel de `ldd`).

5 Déboguer et optimiser

5.1 Etude préventive du code (lint)

Il n'existe pas de lint qui soit réellement utilisable, tout simplement parce que la grande majorité des développeurs sont satisfaits des messages d'avertissement de gcc. Il est probable que l'option la plus utile est l'option `-Wall` — qui a pour effet d'afficher tous les avertissements possibles.

Il existe une version du domaine public du programme lint que vous pouvez trouver à l'adresse suivante : `ftp://larch.lcs.mit.edu/pub/Larch/lclint` . Je ne sais pas ce qu'elle vaut.

5.2 Déboguer

5.2.1 Comment rendre débogable un programme ?

Vous devez compiler et effectuer l'édition de liens avec l'option `-g`, et sans l'option `-fomit-frame-pointer`. En fait, vous ne devez compiler que les modules que vous avez besoin de déboguer.

Si vous possédez un système a.out, les bibliothèques dynamiques sont compilées avec l'option `-fomit-frame-pointer`, que gcc ne peut pas gérer. Lorsque vous compilez avec l'option `-g`, alors par défaut vous effectuez une édition de liens statique, ce qui permet de résoudre le problème.

Si l'éditeur de liens échoue avec un message disant qu'il n'arrive pas à trouver la bibliothèque libg.a, c'est que vous ne possédez pas la bibliothèque `/usr/lib/libg.a`, qui est la bibliothèque C standard permettant le débogage. Cette bibliothèque est fournie dans le paquetage des binaires de la libc., ou (dans les nouvelles versions) vous aurez besoin de récupérer le source et de le compiler vous-même. Vous n'avez pas réellement besoin de cela en fait, vous pouvez faire un lien logique vers `/usr/lib/libc.a`

Comment réduire la taille des exécutables ? Bon nombre de produits GNU sont fournis pour compiler avec l'option `-g`, ce qui génère des exécutables d'une taille très importante (et souvent l'édition de liens s'effectue d'une manière statique). Ce n'est pas une idée lumineuse...

Si le programme possède le script `configure` généré par autoconf, vous pouvez modifier les options de débogage en effectuant un `./configure CFLAGS=` ou `./configure CFLAGS=-O2`. Sinon, vous pouvez aller modifier le Makefile. Bien sûr, si vous utilisez le format ELF, l'édition de liens sera effectuée de manière dynamique même avec l'option `-g`. Dans ce cas, vous pouvez effectuer un strip sur l'exécutable.

5.2.2 Programmes disponibles

Beaucoup de gens utilisent **gdb**, que vous pouvez récupérer sur le site

prep.ai.mit.edu `ftp://prep.ai.mit.edu/pub/gnu` , sous une forme binaire sur *tsx-11* `ftp://tsx-11.mit.edu/pub/linux/packages/GCC` ou sur sunsite. **xxgdb** est une surcouche X de gdb (c.a.d. que vous avez besoin de gdb pour utiliser xxgdb). Les sources peuvent être récupérés sur `ftp://ftp.x.org/contrib/xxgdb-1.08.tar.gz`

Il existe également le débogueur **UPS** qui a été porté par Rick Sladkey. Il fonctionne sous X également, mais à la différence d'xxgdb, ce n'est qu'une surcouche X pour un débogueur en mode en texte. Il possède certaines caractéristiques très intéressantes et si vous utilisez beaucoup ce genre d'outils, vous l'essayerez sûrement. Les patches ainsi que des versions précompilées pour Linux peuvent être trouvées sur

`ftp://sunsite.unc.edu/pub/Linux/devel/debuggers/` , et les sources peuvent être récupérés sur `ftp://ftp.x.org/contrib/ups-2.45.2.tar.Z` .

Un autre outil que vous pouvez trouver utile pour déboguer est « **strace** » , qui affiche les appels systèmes que le processus lance. Il possède d'autres caractéristiques telles que donner les chemins d'accès où ont été compilés les binaires, donner les temps passés dans chacun des appels systèmes, et il vous permet également de connaître les résultats des appels. La dernière version de strace (actuellement la version 3.0.8) peut être trouvée sur

`ftp://ftp.std.com/pub/jrs/` .

5.2.3 Programmes en tâche de fond (démon)

Les démons lancent typiquement un `fork()` dès leur lancement et terminent donc le père. Cela fait une session de déboguage très courte.

La manière la plus simple de résoudre ce problème est de poser un point d'arrêt sur `fork`, et lorsque le programme s'arrête, forcer le retour à 0.

```
(gdb) list
1       #include <stdio.h>
2
3       main()
4       {
5         if(fork()==0) printf("child\n");
6         else printf("parent\n");
7       }
(gdb) break fork
Breakpoint 1 at 0x80003b8
(gdb) run
Starting program: /home/dan/src/hello/./fork
Breakpoint 1 at 0x400177c4

Breakpoint 1, 0x400177c4 in fork ()
(gdb) return 0
Make selected stack frame return now? (y or n) y
```

```
#0  0x80004a8 in main ()
    at fork.c:5
5         if(fork()==0) printf("child\n");
(gdb) next
Single stepping until exit from function fork,
which has no line number information.
child
7       }
```

5.2.4 Fichiers core

Lorsque Linux se lance, il n'est généralement pas configuré pour produire des fichiers core. Si vous les voulez vous devez utiliser votre shell pour ça en faisant sous csh (ou tcsh) :

```
% limit core unlimited
```

avec sh, bash, zsh, pdksh, utilisez

```
$ ulimit -c unlimited
```

Si vous voulez pousser le vice à nommer votre fichier core (par exemple si vous utilisez un débogueur bogué... ce qui est un comble) vous pouvez simplement modifier le noyau. Editez les fichiers `fs/binfmt_aout.c` et `fs/binfmt_elf.c` (dans les nouveaux noyaux, vous devrez chercher ailleurs) :

```
        memcpy(corefile,"core.",5);
#if 0
        memcpy(corefile+5,current->comm,sizeof(current->comm));
#else
        corefile[4] = '\0';
#endif
```

et changez les `0` par des `1`.

5.3 Caractéristiques du programme

Il est possible d'examiner un peu le programme pour savoir quels sont les appels de fonctions qui sont effectués le plus souvent ou bien qui prennent du temps. C'est une bonne manière d'optimiser le code en déterminant là où l'on passe le plus de temps. Vous devez compiler tous les objets avec l'option `-p`, et pour mettre en forme la sortie écran, vous aurez besoin du programme `gprof` (situé dans les `binutils`). Consultez les pages de manuel `gprof` pour plus de détails.

6 Edition de liens

Entre les deux formats de binaires incompatibles, bibliothèques statiques et dynamiques, on peut comparer l'opération d'édition de lien en fait à un jeu ou l'on se demanderait qu'est-ce qui se passe lorsque je lance le programme ? Cette section n'est pas vraiment simple...

Pour dissiper la confusion qui règne, nous allons nous baser sur ce qui se passe lors d'exécution d'un programme, avec le chargement dynamique. Vous verrez également la description de l'édition de liens dynamiques, mais plus tard. Cette section est dédiée à l'édition de liens qui intervient à la fin de la compilation.

6.1 Bibliothèques partagées contre bibliothèques statiques

La dernière phase de construction d'un programme est de réaliser l'édition de liens, ce qui consiste à assembler tous les morceaux du programme et de chercher ceux qui sont manquants. Bien évidement, beaucoup de programmes réalisent les mêmes opérations comme ouvrir des fichiers par exemple, et ces pièces qui réalisent ce genre d'opérations sont fournies sous la forme de bibliothèques. Sous Linux, ces bibliothèques peuvent être trouvées dans les répertoires `/lib` et`/usr/lib/` entre autres.

Lorsque vous utilisez une bibliothèque statique, l'éditeur de liens cherche le code dont votre programme a besoin et en effectue une copie dans le programme physique généré. Pour les bibliothèques partagées, c'est le contraire : l'éditeur de liens laisse du code qui lors du lancement du programme chargera automatiquement la bibliothèque. Il est évident que ces bibliothèques permettent d'obtenir un exécutable plus petit; elles permettent également d'utiliser moins de mémoire et moins de place disque. Linux effectue par défaut une édition de liens dynamique s'il peut trouver les bibliothèques de ce type sinon, il effectue une édition de liens statique. Si vous obtenez des binaires statiques alors que vous les voulez dynamiques vérifiez que les bibliothèques existent (`*.sa` pour le format a.out, et `*.so` pour le format ELF) et que vous possédez les droits suffisants pour y accéder (lecture).

Sous Linux, les bibliothèques statiques ont pour nom `libnom.a`, alors que les bibliothèques dynamiques sont appelées `libnnom.so.x.y.z` où `x.y.z` représente le numéro de version. Les bibliothèques dynamiques ont souvent des liens logiques qui pointent dessus, et qui sont très importants. Normalement, les bibliothèques standards sont livrées sous la double forme dynamique et statique.

Vous pouvez savoir de quelles bibliothèques dynamiques un programme a besoin en utilisant la commande `ldd` (*List Dynamic Dependencies*)

```
$ ldd /usr/bin/lynx
        libncurses.so.1 => /usr/lib/libncurses.so.1.9.6
        libc.so.5 => /lib/libc.so.5.2.18
```

Cela indique sur mon système que l'outil `lynx` (outil WWW) a besoin des bibliothèques dynamiques `libc.so.5` (la bibliothèque C) et de `libncurses.so.1` (nécessaire pour le contrôle du terminal). Si un programme ne possède pas de dépendances, `ldd` indiquera '*statically linked*' (édition de liens statique).

6.2 A la recherche des fonctions... ou dans quelle bibliothèque se trouve la fonction `sin()` ?')

`nm` *nomdebibliothèque* vous donne tous les symboles référencés dans la bibliothèque. Cela fonctionne que cela soit du code statique ou dynamique. Supposez que vous vouliez savoir où se trouve définie la fonction `tcgetattr()` :

```
$ nm libncurses.so.1 |grep tcget
         U tcgetattr
```

La lettre `U` vous indique que c'est indéfini (*Undefined*) — cela indique que la bibliothèque ncurses l'utilise mais ne la définit pas. Vous pouvez également faire :

```
$ nm libc.so.5 | grep tcget
00010fe8 T __tcgetattr
00010fe8 W tcgetattr
00068718 T tcgetpgrp
```

La lettre 'W' indique que le symbole est défini mais de telle manière qu'il peut être surchargé par une autre définition de la fonction dans une autre bibliothèque (W pour *weak* : faible). Une définition normale est marquée par la lettre 'T' (comme pour `tcgetpgrp`).

La réponse à la question située dans le titre est `libm.(so|a)`. Toutes les fonctions définies dans le fichier d'en-tête `<math.h>` sont implémentées dans la bibliothèque mathématique donc vous devrez effectuer l'édition de liens grâce à `-lm`.

6.3 Trouver les fichiers

Supposons que vous ayez le message d'erreur suivant de la part de l'éditeur de liens :

```
ld:  Output file requires shared library 'libfoo.so.1'
```

La stratégie de recherche de fichiers de ld ou de ses copains diffère de la version utilisée, mais vous pouvez être sûr que les fichiers situés dans le répertoire `/usr/lib` seront trouvés. Si vous désirez que des fichiers situés à un endroit différent soient trouvés, il est préférable d'ajouter l'option `-L` à gcc ou ld.

Si cela ne vous aide pas clairement, vérifiez que vous avez le bon fichier à l'endroit spécifié. Pour un système a.out, effectuer l'édition de liens avec `-ltruc` implique que ld recherche les bibliothèques `libtruc.sa` (bibliothèques partagées), et si elle n'existe pas, il recherche `libtruc.a` (statique). Pour le format ELF, il cherche `libtruc.so` puis `libtruc.a`. `libtruc.so` est généralement un lien symbolique vers `libtruc.so.x`.

6.4 Compiler votre propre bibliothèque

6.4.1 Numéro de la version

Comme tout programme, les bibliothèques ont tendance à avoir quelques bogues qui sont corrigés au fur et à mesure. De nouvelles fonctionnalités sont ajoutées et qui peuvent changer l'effet de celles qui existent ou bien certaines anciennes peuvent êtres supprimées. Cela peut être un problème pour les programmes qui les utilisent.

Donc, nous introduisons la notion de numéro de version. Nous répertorions les modifications effectuées dans la bibliothèques comme étant soit mineures soit majeures. Cela signifie qu'une modification mineure ne peut pas modifier le fonctionnement d'un programme (en bref, il continue à fonctionner comme avant). Vous pouvez identifier le numéro de la version de la bibliothèque en regardant son nom (en fait c'est un mensonge pour les bibliothèques ELF... mais continuez à faire comme si !) : `libtruc.so.1.2` a pour version majeure 1 et mineure 2. Le numéro de version mineur peut être plus ou moins élevé — la bibliothèque C met un numéro de patch, ce qui produit un nom tel que `libc.so.5.2.18`, et c'est également courant d'y trouver des lettres ou des blancs soulignés ou tout autre caractère ASCII affichable.

Une des principales différences entre les formats ELF et a.out se trouve dans la manière de construire la bibliothèque partagée. Nous traiterons les bibliothèques partagées en premier car c'est plus simple.

6.4.2 ELF, qu'est-ce que c'est ?

ELF (*Executable and Linking Format*) est format de binaire initialement conçu et développé par USL (*UNIX System Laboratories*) et utilisé dans les systèmes Solaris et System R4. En raison de sa facilité d'utilisation par rapport à l'ancien format dit a.out qu'utilisait Linux, les développeurs de GCC et de la bibliothèque C ont décidé l'année dernière de basculer tout le système sous le format ELF. ELF est désormais le format binaire standard sous Linux.

ELF, le retour ! Ce paragraphe provient du groupe '/news-archives/comp.sys.sun.misc'.

> ELF (*Executable Linking Format*) est le « nouveau et plus performant » format de fichier introduit dans SVR4. ELF est beaucoup plus puissant que le sacro-saint format COFF, dans le sens où il est extensible. ELF voit un fichier objet comme une longue liste de sections (plutôt qu'un tableau de taille fixe d'éléments). Ces sections, à la différence de COFF ne se trouvent pas à un endroit constant et ne sont pas dans un ordre particulier, etc. Les utilisateurs peuvent ajouter une nouvelle section à ces fichiers objets s'il désirent y mettre de nouvelles données. ELS possède un format de débogage plus puissant appelé DWARF (*Debugging With Attribute Record Format*) - par encore entièrement géré par Linux (mais on y travaille !). Une liste chaînée de « DWARF DIEs » (ou *Debugging Information Entries* - NdT... le lecteur aura sûrement noté le jeu de mot assez noir : dwarf = nain; dies = morts) forment la section *.debug* dans ELF. Au lieu d'avoir une liste de petits enregistrements d'information de taille fixes, les DWARF DIEs contiennent chacun une longue liste complexe d'attributs et sont écrits sous la forme d'un arbre de données. Les DIEs peuvent contenir une plus grande quantité d'information que la section *.debug* du format COFF ne le pouvait (un peu comme les graphes d'héritages du C++).
>
> Les fichiers ELF sont accessibles grâce à la bibliothèque d'accès de SVR4 (Solaris 2.0 peut-être ?), qui fournit une interface simple et rapide aux parties les plus complexes d'ELF. Une des aubaines que permet la bibliothèque d'accès ELF est que vous n'avez jamais besoin de connaître les méandres du format ELF. Pour accéder à un fichier Unix, on utilise un Elf *, retourné par un appel à elf_open(). Ensuite, vous effectuez des appels à elf_foobar() pour obtenir les différents composants au lieu d'avoir à triturer le fichier physique sur le disque (chose que beaucoup d'utilisateurs de COFF ont fait...).

Les arguments pour ou contre ELF, et les problèmes liés à la mise à jour d'un système a.out vers un système ELF sont décrits dans le ELF-HOWTO et je ne veux pas effectuer de copier coller ici (NdT: ce HowTo est également traduit en français). Ce HowTo se trouve au même endroit que les autres.

Les bibliothèque partagées ELF Pour construire `libtruc.so` comme une bibliothèque dynamique, il suffit de suivre les étapes suivantes :

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

Cela va générer une bibliothèque partagée appelée `libtruc.so.1.0`, les liens appropriés pour ld (`libtruc.so`) et le chargeur dynamique (`libtruc.so.1`) pour le trouver. Pour tester, nous ajoutons le répertoire actuel à la variable d'environnement `LD_LIBRARY_PATH`.

Lorsque vous êtes satisfait et que la bibliothèque fonctionne, vous n'avez plus qu'à la déplacer dans le répertoire par exemple, `/usr/local/lib`, et de recréer les liens appropriés. Le lien de `libtruc.so.1` sur `libtruc.so.1.0` est enregistré par `ldconfig`, qui sur bon nombre de systèmes est lancé lors du processus d'amorçage. Le lien `libfoo.so` doit être mis à jour à la main. Si vous faites attention lors de la mise à jour de la bibliothèque la chose la plus simple à réaliser est de créer le lien `libfoo.so -> libfoo.so.1`, pour que ldconfig conserve les liens actuels. Si vous ne faites pas cela, vous aurez des problèmes plus tard. Ne me dites pas que l'on ne vous a pas prévenu !

```
$ /bin/su
```

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

Les numéros de version, les noms et les liens Chaque bibliothèque possède un nom propre (*soname*). Lorsque l'éditeur de liens en trouve un qui correspond à un nom cherché, il enregistre le nom de la bibliothèque dans le code binaire au lieu d'y mettre le nom du fichier de la bibliothèque. Lors de l'exécution, le chargeur dynamique va alors chercher un fichier ayant pour nom le nom propre de la bibliothèque, et pas le nom du fichier de la bibliothèque. Par exemple, une bibliothèque ayant pour nom `libtruc.so` peut avoir comme nom propre `libbar.so`, et tous les programmes liés avec vont alors chercher `libbar.so` lors de leur exécution.

Cela semble être une nuance un peu pointilleuse mais c'est la clef de la compréhension de la coexistence de plusieurs versions différentes de la même bibliothèque sur le même système. On a pour habitude sous Linux d'appeler une bibliothèque `libtruc.so.1.2` par exemple, et de lui donner comme nom propre `libtruc.so.1`. Si cette bibliothèque est rajoutée dans un répertoire standard (par exemple dans `/usr/lib`), le programme `ldconfig` va créer un lien symbolique entre `libtruc.so.1 -> libtruc.so.1.2` pour que l'image appropriée soit trouvée lors de l'exécution. Vous aurez également besoin d'un lien symbolique `libtruc.so -> libtruc.so.1` pour que ld trouve le nom propre lors de l'édition de liens.

Donc, lorsque vous corrigez des erreurs dans la bibliothèque ou bien lorsque vous ajoutez de nouvelles fonctions (en fait, pour toute modification qui n'affecte pas l'exécution des programmes déjà existants), vous reconstruisez la bibliothèque, conservez le nom propre tel qu'il était et changez le nom du fichier. Lorsque vous effectuez des modifications que peuvent modifier le déroulement des programmes existants, vous pouvez tout simplement incrémenter le nombre situé dans le nom propre — dans ce cas, appelez la nouvelle version de la bibliothèque `libtruc.so.2.0`, et donnez-lui comme nom propre `libtruc.so.2`. Maintenant, faites pointer le lien de `libfoo.so` vers la nouvelle version et tout est bien dans le meilleur des mondes !

Il est utile de remarquer que vous n'êtes pas obligé de nommer les bibliothèques de cette manière, mais c'est une bonne convention. Elf vous donne une certaine liberté pour nommer des bibliothèques tant et si bien que cela peut perturber certains utilisateurs, mais cela ne veut pas dire que vous êtes obligé de le faire.

Résumé : supposons que choisissiez d'adopter la méthode traditionnelle avec les mises à jour majeures qui peuvent ne pas être compatibles avec les versions précédentes et les mises à jour mineures qui ne posent pas ce problème. Il suffit de créer la bibliothèque de cette manière :

```
gcc -shared -Wl,-soname,libtruc.so.majeur -o libtruc.so.majeur.mineur
```

et tout devrait être parfait !

6.4.3 a.out. Le bon vieux format

La facilité de construire des bibliothèque partagées est la raison principale de passer à ELF. Ceci dit, il est toujours possible de créer des bibliothèques dynamiques au format a.out. Récupérez le fichier archive

`ftp://tsx-11.mit.edu/pub/linux/packages/GCC/src/tools-2.17.tar.gz`

et lisez les 20 pages de documentation que vous trouverez dedans après l'avoir désarchivé. Je n'aime pas avoir l'air d'être aussi partisan, mais il est clair que je n'ai jamais aimé ce format :-).

ZMAGIC contre QMAGIC QMAGIC est le format des exécutables qui ressemble un peu aux vieux binaires a.out (également connu comme ZMAGIC), mais qui laisse la première page libre. Cela permet plus facilement de récupérer les adresses non affectées (comme NULL) dans l'intervalle 0-4096 (NdT : Linux utilise des pages de 4Ko).

Les éditeurs de liens désuets ne gèrent que le format ZMAGIC, ceux un peu moins rustiques gèrent les deux, et les plus récents uniquement le QMAGIC. Cela importe peu car le noyau gère les deux types.

La commande `file` est capable d'identifier si un programme est de type QMAGIC.

Gestion des fichiers Une bibliothèque dynamique a.out (DLL) est composée de deux fichiers et d'un lien symbolique. Supposons que l'on utilise la bibliothèque *truc*, les fichiers seraient les suivants : `libtruc.sa` et `libtruc.so.1.2`; et le lien symbolique aurait pour nom `libtruc.so.1` et pointerait sur le dernier des fichiers. Mais à quoi servent-ils ?

Lors de la compilation, `ld` cherche `libtruc.sa`. C'est le fichier de description de la bibliothèque : il contient toutes les données exportées et les pointeurs vers les fonctions nécessaires pour l'édition de liens.

Lors de l'exécution, le chargeur dynamique cherche `libtruc.so.1`. C'est un lien symbolique plutôt qu'un réel fichier pour que les bibliothèques puissent être mise à jour sans avoir à casser les applications qui utilisent la bibliothèque. Après la mise à jour, disons que l'on est passé à la version `libfoo.so.1.3`, le lancement de ldconfig va positionner le lien. Comme cette opération est atomique, aucune application fonctionnant n'aura de problème.

Les bibliothèques DLL (Je sais que c'est une tautologie... mais pardon !) semblent être très souvent plus importantes que leur équivalent statique. En fait, c'est qu'elles réservent de la place pour les extensions ultérieures sous la simple forme de trous qui sont fait de telle manière qu'ils n'occupent pas de place disque (NdT : un peu comme les fichiers `core`). Toutefois, un simple appel à `cp` ou à `makehole` les remplira... Vous pouvez effectuer une opération de `strip` après la construction de la bibliothèque, comme les adresses sont à des endroits fixes. **Ne faites pas la même opération avec les bibliothèques ELF !**

« libc-lite » ? Une « libc-lite » (contraction de *libc* et *little*) est une version épurée et réduite de la bibliothèque libc construite de telle manière qu'elle puisse tenir sur une disquette avec un certain nombre d'outil Unix. Elle n'inclut pas curses, dbm, termcap, ... Si votre `/lib/libc.so.4` est liée avec une bibliothèque de ce genre il est très fortement conseillé de la remplacer avec une version complète.

6.4.4 Edition de liens : problème courants

Envoyez-les moi !

Des programmes statiques lorsque vous les voulez partagés

Vérifiez que vous avez les bons liens pour que `ld` puisse trouver les bibliothèques partagées. Pour ELF cela veut dire que `libtruc.so` est un lien symbolique sur son image, pour a.out un fichier `libtruc.sa`. Beaucoup de personnes ont eu ce problème après être passés des outils ELF 2.5 à 2.6 (`binutils`) — la dernière version effectue une recherche plus intelligente pour les bibliothèques dynamiques et donc ils n'avaient pas créé tous les liens symboliques nécessaires. Cette caractéristique avait été supprimée pour des raisons de compatibilité avec d'autres architectures et parce qu'assez souvent cela ne marchait pas bien. En bref, cela posait plus de problèmes qu'autre chose.

Le programme 'mkimage' n'arrive pas à trouver libgcc

Comme `libc.so.4.5.x` et suivantes, libgcc n'est pas une bibliothèque partagée. Vous devez remplacer les '`-lgcc`' sur la ligne de commande par `` `gcc -print-libgcc-file-name` `` (entre quotes)

Egalement, détruisez tous les fichiers situés dans `/usr/lib/libgcc*`. C'est important.

Le message `__NEEDS_SHRLIB_libc_4 multiply defined`

Sont une conséquence du même problème.

Le message "Assertion failure" apparaît lorsque vous reconstruisez une DLL

Ce message énigmatique signifie qu'un élément de votre table *jump* a dépassé la table car trop peu de place était réservée dans le fichier `jump.vars` file. Vous pouvez trouver le(s) coupable(s) en lançant la commande `getsize` fournie dans le paquetage tools-2.17.tar.gz. La seule solution est de passer à une nouvelle version majeure, même si elle sera incompatible avec les précédentes.

`ld: output file needs shared library libc.so.4`

Cela arrive lorsque vous effectuez l'édition de liens avec des bibliothèques différentes de la libc (comme les bibliothèques X) et que vous utilisez l'option `-g` sans utiliser l'option `-static`.

Les fichiers `.sa` pour les bibliothèques dynamiques ont un symbole non résolu `_NEEDS_SHRLIB_libc_4` qui est défini dans `libc.sa`. Or, lorsque vous utilisez `-g` vous faites l'édition de liens avec `libg.a` ou `libc.a` et donc ce symbole n'est jamais défini.

Donc, pour résoudre le problème, ajoutez l'option `-static` lorsque vous compilez avec l'option `-g`, ou n'utilisez pas `-g` lors de l'édition de liens !

7 Chargement dynamique

Ce paragraphe est en fait un peu court : il sera étendu dans une version ultérieure dès que j'aurai récupéré le HowTo ELF

7.1 Concepts

Linux possède des bibliothèques dynamiques, comme on vous le répète depuis le début de ce document ! Or, il existe un système pour reporter le travail d'association des noms des symboles et de leur adresse dans la bibliothèque, qui est normalement effectué lors de l'édition de liens en l'effectuant lors du chargement du programme.

7.2 Messages d'erreur

Envoyez moi vos erreurs ! Je n'en fait pas grand chose sauf les insérer dans ce paragraphe...

`can't load library: /lib/libxxx.so, Incompatible version`

(seulement a.out) Cela signifie que vous n'avez pas la version correcte de la bibliothèque (numéro dit majeur). Non, il n'est pas possible d'effectuer un lien symbolique sur la bibliothèque que vous possédez : si vous avez de la chance, vous obtiendrez un *segmentation fault*. Récupérez la nouvelle version. Un message un peu équivalent existe également sur les systèmes ELF :

```
ftp: can't load library 'libreadline.so.2'
```

`warning using incompatible library version xxx`

(seulement a.out) Vous avez un numéro de version de bibliothèque (mineur) inférieur à la version avec laquelle a été compilé le programme. Le programme fonctionnera sûrement. Une mise à jour est toutefois conseillée.

7.3 Contrôler l'opération de chargement dynamique

Il existe certaines variables d'environnements que le chargeur dynamique utilise. Beaucoup sont exploitées par le programme `ldd` lorsqu'il s'agit de particularités de l'environnement de l'utilisateur, ce qui peuvent être positionnées pour lancer ldd avec des options particulières. Voici une description des différentes variables d'environnement que vous pouvez rencontrer :

- `LD_BIND_NOW` — normalement, les fonctions ne sont pas cherchées dans les bibliothèques avant leur appel. En positionnant cette option, vous vérifiez que toutes les fonctions employées dans votre programmes se trouvent bien dans la bibliothèque lors de son chargement, ce qui ralentit le lancement du programme. C'est utile lorsque vous voulez tester que l'édition de liens s'est parfaitement déroulée et que tous les symboles sont bien associés.

- `LD_PRELOAD` peut être défini avec un nom de fichier qui contient des fonctions surchargeant des fonctions déjà existantes. Par exemple, si vous testez une stratégie d'allocation mémoire, et que vous voulez remplacer le malloc de la bibliothèque C par le vôtre situé dans un module ayant pour nom `malloc.o`, il vous suffit de faire :

  ```
  $ export LD_PRELOAD=malloc.o
  $ test_mon_malloc
  ```

  `LD_ELF_PRELOAD` et `LD_AOUT_PRELOAD` sont similaires, mais leur utilisation est spécifique au type de binaire utilisé. Si `LD_`*TypeBinaire*`_PRELOAD` et `LD_PRELOAD` sont positionnés, celui correspondant le mieux à la machine est utilisé.

- `LD_LIBRARY_PATH` contient une liste de répertoires contenant les bibliothèques dynamiques. Cela n'affecte pas l'édition de liens : cela n'a qu'un effet lors de l'exécution. Il faut noter qu'elle est désactivée pour des programmes qui s'exécutent avec un setuid ou un setgid. Enfin, `LD_ELF_LIBRARY_PATH` et `LD_AOUT_LIBRARY_PATH` peuvent être utilisés pour orienter le mode de compilation du binaire. `LD_LIBRARY_PATH` ne devrait pas être nécessaire en principe : ajoutez les répertoires dans le fichier `/etc/ld.so.conf/` et relancez ldconfig.

- `LD_NOWARN` s'applique au format a.out uniquement. Lorsqu'elle est positionnée (c.a.d si elle existe par exemple avec `LD_NOWARN=true; export LD_NOWARN`) cela arrête le chargeur du programme même sur des avertissements insignifiants (tels que des messages d'incompatibilités de numéros mineurs de version).

- `LD_WARN` s'applique à ELF uniquement. Lorsqu'elle est positionnée, on transforme le message habituellement fatal *Can't find library* en un avertissement. Ce n'est pas positionné par défaut mais c'est important pour un programme comme ldd.

- `LD_TRACE_LOADED_OBJECTS` s'applique à ELF uniquement, et permet de simuler l'exécution des programmes comme s'ils l'étaient par `ldd` :

  ```
  $ LD_TRACE_LOADED_OBJECTS=true /usr/bin/lynx
          libncurses.so.1 => /usr/lib/libncurses.so.1.9.6
          libc.so.5 => /lib/libc.so.5.2.18
  ```

7.4 Ecrire des programmes en utilisant le chargement dynamique

Cela ressemble énormément au système de chargement dynamique utilisé sous Solaris 2.x. Ce système est décrit d'une manière précise dans le document expliquant la programmation avec ELF écrit par H J Lu et dans la page de manuel `dlopen(3)`, qui se trouve dans le paquetage ld.so. Voici un exemple simple : pensez à faire l'édition de liens avec `-ldl`

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

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

  if(libc=dlopen("/lib/libc.so.5",RTLD_LAZY))
  {
    printf_call = dlsym(libc,"printf");
    (*printf_call)("Bonjour ! Ha ben ca marche pil poil sous Linux !\n");
  }

}
```

8 Contacter les développeurs

8.1 Annoncer des bogues

Commencez par mettre en doute le problème. Est-ce spécifique à Linux ou bien cela arrive avec gcc mais sur d'autres plates-formes ? Est-ce spécifique à la version du noyau ? A la version de la bibliothèque C ? Est-ce que ce problème disparaît lorsque vous effectuez une édition de liens statique ? Pouvez-vous produire un code très court mettant en évidence le problème ?

Après avoir répondu après ces quelques questions, vous saurez quel programme est à l'origine du problème. Pour un problème direct avec GCC, le mieux est de consulter le fichier d'information livré avec : la procédure pour rapporter un bogue y est détaillé. Pour un problème avec ld.so, la bibliothèque C ou mathématique, envoyez un courrier électronique à `linux-gcc@vger.rutgers.edu`. Si possible, donnez un court exemple mettant en évidence le problème ainsi qu'une courte description indiquant ce que le programme aurait normalement dû faire, et ce qu'il fait en réalité.

8.2 Participer au développement

Si vous désirez participer au développement de GCC ou de la bibliothèque C, la première chose à faire est de rejoindre la liste de diffusion `linux-gcc@vger.rutgers.edu`. Si vous désirez uniquement savoir de quoi ça parle, il existe des archives à l'adresse `http://homer.ncm.com/linux-gcc/` . Tout dépend de ce que vous désirez faire ou apporter à ce projet !

9 Divers

9.1 Ce document

Ce HowTo est basé sur la FAQ de Mitchum DSouza's. Bon nombre des informations en proviennent. D'une manière générale, il est fréquent de dire une phrase du genre « je n'ai pas tout testé et donc ne me blâmez pas si vous cassez votre disque, votre système ou si vous rompez avec votre épouse ».

Le nom des contributeurs à ce document sont donnés par ordre alphabétique : Andrew Tefft, Axel Boldt, Bill Metzenthen, Bruce Evans, Bruno Haible, Daniel Barlow, Daniel Quinlan, David Engel, Dirk Hohndel, Eric Youngdale, Fergus Henderson, H.J. Lu, Jens Schweikhardt, Kai Petzke, Michael Meissner, Mitchum DSouza,

Olaf Flebbe, Paul Gortmaker, Rik Faith, Steven S. Dick, Tuomas J Lukka, et bien sûr Linus Torvalds, sans qui ce genre d'exercice aurait été difficile, voir impossible :-)

Ne soyez pas offensé si votre nom n'apparaît pas dans la liste et que vous ayez contribué à ce document (sous la forme d'un HowTo ou d'une FAQ). Envoyez-moi un courrier électronique et j'effectuerai la correction.

9.2 Traduction

A l'heure ou j'écris ces lignes, je ne connais pas de traduction de ce document. Si vous en réalisez une, s'il vous plaît dites-le moi. Je suis disponible pour toute aide concernant l'explication du texte, je serai très content d'y répondre.

Note du traducteur : **Cocorico !** La version française est la première traduction de ce document.

9.3 Contacts

Tout contact est le bienvenu. Envoyez-moi un courrier électronique à l'adresse suivante : *dan@detached.demon.co.uk* . Ma clef publique PGP (ID 5F263625) est disponible sur mes *pages WWW* `http://ftp.linux.org.uk/~barlow/` , Si vous souhaitez rendre confidentiel certains messages.

9.4 Copyright

10 Index

Les entrées de cet index sont triées dans l'ordre alphabétique.