Portabilidade e flexibilidade em software livre: a experiência do GoboLinux

Hisham H. Muhammad, Rafael G. Jeffman

[1] GoboLinux.org

{lode,rafasgj}@gobolinux.org

Resumo. *Este artigo discute alguns dos problemas encontrados em diversas aplicações distribuídas como software livre, no que se refere a flexibilidade e portabilidade destas aplicações e as implicações das restrições encontradas na adaptação destas aplicações ao GoboLinux, uma distribuição Linux que não segue alguns padrões convencionais como a hierarquia de diretórios e o processo de inicialização. São abordadas algumas ferramentas utilizadas para o desenvolvimento de software livre e alguns problemas encontrados tanto nas ferramentas quanto no mau uso destas. São apontadas soluções possíveis e é proposta uma nova forma de encarar a flexibilidade e a portabilidade durante o desenvolvimento de software livre.*

Abstract. *This article discusses some of the problems existing in several applications distributed as free software, in terms of flexibity and portability and the implications of those restrictions in the adaptation of these applications to GoboLinux, a Linux distribution that does not follow some of the established conventions such as the directory hierarchy and the start-up process. Some of the tools used in the development of free software are covered in this work, as well as some of the problems that arise in their use and misuse. Possible solutions are pointed and, most importantly, a new way to view flexibility and portability in the development of free software.*

1 Introdução

O conceito de portabilidade, adquire, com o modelo de desenolvimento de software livre, novas proporções. Software proprietários normalmente são desenvolvidos visando as arquiteturas de hardware e software dominantes no mercado, e manter uma equipe de desenvolvimento e testes para arquiteturas menos utilizadas é inviável para a maioria das empresas, mesmo que na prática, garantir a portabilidade de um programa muitas vezes envolva apenas cuidados triviais. Com a disponibilidade do código-fonte, este tipo de verificação acaba sendo feita pela própria comunidade de usuários, trazendo benefícios para ambas as partes: por um lado, a comunidade de usuários potenciais da aplicação aumenta; por outro lado, a plataforma é beneficiada com a disponibilidade de mais aplicações.

Apesar disso, a maioria das questões referentes a portabilidade que são enfrentadas pelos usuários poderiam ser evitadas. Uma série de premissas assumidas pelos autores das aplicações, com base nas características dos sistemas utilizados por eles, não são verdadeiras em outros ambientes. Não estamos falando aqui de funcionalidades do sistema operacional que realmente demandariam bibliotecas de compatibilização, mas simplesmente de peculiaridades como, por exemplo, localização de arquivos.

Este artigo relata a experiência obtida nesse tema a partir do desenvolvimento do *GoboLinux*, uma distribuição GNU/Linux baseada em uma hierarquia de diretórios alternativa ([7]). A partir desta experiência, uma série de deficiências de portabilidade e flexibilidade freqüentemente ocorrentes em softwares livres foram encontradas. O artigo discute como estes problemas podem ser evitados, de modo a tornar a comunidade de desenvolvedores e usuários de software livre menos centrada em torno de um conjunto pequeno de distribuições, ou em última análise, de um único sistema operacional.

2 Padronização

O estabelecimento de padrões, à primeira vista, visa diminuir as diferenças entre os ambientes. De fato, tentativas de estabelecimento de padrões como o Linux Standard Base ([6]) têm este como seu objetivo, arbitrando características que os sistemas devem seguir de modo a buscar uma certa homogeneização. Padrões como estes acabam apenas por limitar as possibilidades, e restringir opções.

	Total de arquivos	Contendo *hard-coded*
`/System/Links/Executables`	1760	701 (39,82%)
`/System/Links/Libraries`	13878	3138 (22,61%)
`/System/Links/Headers`	6351	5 (0,07%)

Tabela 1: Quantidade de arquivos contendo caminhos hard-coded

Padrões devem, na verdade, ter por objetivo *especificar interoperabilidade*. Somente desta forma é possível manter a liberdade de implementação, para que alternativas melhores possam sempre ser experimentadas e, possivelmente, adotadas. Um caminho nesta direção é a adoção das GNU AutoTools ([8]) por um número crescente de aplicações. Ao utilizar estas ferramentas, o desenvolvedor permite que a aplicação seja utilizável em diversos ambientes, uma vez que o processo de detecção e configuração da aplicação para o ambiente-alvo é realizado pelo AutoConf.

O uso de GNU AutoTools por parte das aplicações permite que elas sejam utilizadas no GoboLinux sem adaptação alguma, o que facilitou largamente a instalação de aplicações. Ainda assim, é necessária uma ressalva em relação ao uso correto das ferramentas. Foi desenvolvido para o GoboLinux uma ferramenta chamada FIBOSANDBOX que realiza de forma transparente a instalação de aplicações utilizando um ambiente com permissões de escrita no sistema de arquivos restringidas. Através do FIBOSANDBOX, detectou-se que diversas aplicações, apesar de utilizarem GNU AutoTools para configurar o ambiente de compilação, não o fazem corretamente para a configuração dos caminhos de instalação.

3 Configurações

Em um grande número de programas, os autores cometem o erro de pressupor como fixas determinadas características que, especialmente dada a natureza do software livre, são flexíveis. O modelo de software livre estimula a modificação. Mesmo tendo desenvolvido um programa para um determinado sistema operacional, o autor deve estar ciente de que ele eventualmente será portado.

3.1 Tipo de ambiente

No passado, as grandes diferenças entre UNIX e VMS faziam com que um programa portável entre estas duas arquiteturas fosse facilmente portável para qualquer outra. Atualmente, com a popularização do GNU/Linux e da família BSD, a maior parte dos softwares livres visa apenas a plataforma UNIX. O surgimento do Cygwin ([1]), ambiente UNIX-like para Windows, tenta remediar esta situação. Infelizmente esta abordagem representa uma solução apenas para a plataforma Windows, e técnica é, de certa forma, invasiva, uma vez que de certa forma é o ambiente que é portado, e não a aplicação. O projeto Fink ([3]) lida com questões de portabilidade de softwares livres para o sistema operacional Mac OS X, que atualmente é oficialmente considerado um sistema UNIX. A necessidade da existência de projetos como o Fink apontam a pobre portabilidade existente em um grande número de aplicações, mesmo entre diferentes sistemas UNIX.

3.2 Caminhos no sistema de arquivos

O erro mais comum encontrado são os caminhos *hard-coded* no código-fonte dos programas. O GoboLinux possui uma constituição de caminhos totalmente diferente, o que demanda da aplicação um nível de portabilidade de caminhos similar à necessário para a *instalação* em ambientes não-UNIX. Devido à hierarquia de compatibilidade apresentada em [7], o GoboLinux assemelha-se, em execução, a um sistema UNIX tradicional. Através de opções como `prefix` e `sysconfdir`, as GNU AutoTools permitem o uso de caminhos arbitrários. A ferramenta FIBOSANDBOX, entretanto, detectou que uma série de aplicações não abstraem todos os caminhos referenciados pelas ferramentas.

Caminhos hard-coded dentro do código binário das aplicações, embora não caracterizem um problema de portabilidade sério para software livre (uma vez que o código-fonte é disponível), podem causar incoveniências de interoperabilidade entre sistemas arquiteturalmente compatíveis, como diferentes distribuições Linux. A tabela 1 demonstra a quantidade de caminhos hard-coded encontrados em uma instalação GoboLinux com 275 pacotes instalados, entre aplicações, ferramentas e bibliotecas. É importante

ressaltar que os números listados na tabela referem-se apenas caminhos hard-coded relativos ao caminho-alvo da compilação do pacote — isto é, apenas em relação a caminhos *hard-coded* baseados em `/Programs` — isoladamente de eventuais caminhos *default* incluídos nos código-fonte dos programas. Este tipo de avaliação não seria possível em uma distribuição Linux tradicional, uma vez que caminhos inseridos na compilação e caminhos *default* supridos pela aplicação se confundiriam. Os resultados mostram que caminhos hard-coded são um problema considerável para a flexibilidade de executáveis e bibliotecas, mas são um problema raro, mas ocorrente, em arquivos de *header*s C/C++.

3.3 Identificação do super-usuário

Tradicionalmente o usuário com privilégios em relação a acesso aos dispositivos e arquivos em sistemas multi-usuário (o super-usuário) tem sido chamado de `root` por convenção. Mas não existe uma regra que obrigue os sistemas a utilizar este nome, é apenas prática comum. O única restrição existente para o super-usuário é que este usuário possua um identificador 0 (zero). Para o sistema, o usário com identificador 0 (zero) é o super-usuário, não importando qual o nome seja dado para ele.

Muitos desenvolvedores assumem que o nome do super-usuário é `root`, o que não é uma boa prática, principalmente em relação à portabilidade. Outros sistemas compatíveis com UNIX, por exemplo o BeOS ([5]), não possuem um usuário `root`, mas possuem um super-usuário. Programas escritos de forma a não assumir um usuário `root`, mas identificando o usuário pelo seu identificador numérico, tem menos restrições para que possam ser portados para outras plataformas.

A utilização de outros nomes ao invés de `root` apresenta algumas vantagens como nomes mais representativos, menos ambíguos (usuário `root`, diretório `/root`, sistema de arquivos ROOTFS), e mais seguros, uma vez que é pratica comum tentar invadir sistemas utilizando o login do usuário `root`.

Pelos motivos citados existem diversas vantagens na utilização de nomes alternativos para o super-usuário, tanto para a administração do sistema, quanto para o desenvolvimento de aplicações, neste caso não assumindo um nome pré-definido. No GoboLinux, o nome do super-usuário é `gobo`, e este pode ser facilmente alterado pelo usuário.

3.4 Scripts de inicialização

Não existe um padrão para scripts de inicialização. As diferentes distribuições Linux utilizam uma série de variantes dos dois principais modelos históricos de *boot scripts* (System V e BSD). Deste modo, uma aplicação não deve assumir nada a respeito destes scripts. Ainda assim, uma série de pacotes distribuem scripts para inclusão de serviços na seqüência de inicialização do sistema. Distribuir estes scripts como uma conveniência não é um problema. Todavia, realizar a instalação destes scripts durante o processo padrão de instalação do programa é extremamente problemático, além de prejudicar a manutenção do próprio programa, uma vez que ele estará sujeito às mudanças ocorridas, por exemplo, até em diferentes versões de uma mesma distribuição. Além disso, existem diversos projetos para bootscripts alternativos, não associados a nenhuma distribuição específica, por exemplo baseados em dependências entre os serviços a serem inicializados ([4]).

Outro aspecto a ser levado em conta nos scripts de inicialização é o tipo de aplicação ao qual a máquina serádestinada. Tipicamente, servidores precisam de um número de serviços bem maior do que em máquinas utilizadas como estações de trabalho. Em um servidor o tempo gasto com a inicializaçõa do sistema não é tão crítico, uma vez que são máquinas das quais se espera um uso ininterrupto (24x7). O mesmo não acontece em estações de trabalho, onde se espera um tempo de inicialização bem menor. Para este tipo de máquina o número de serviços executados é bem menor e não faz sentido a utilização de grandes scripts como os BSD ou a complexidade do System V.

4 Ferramentas para compilação de programas

4.1 Libtool

O trabalho de desenvolvimento do GoboLinux detectou uma falha fundamental de projeto da ferramenta LIBTOOL, parte das GNU AutoTools. A falha se origina do fato da ferramenta ter sido desenvolvida tendo em mente os sistemas UNIX, que possuem uma série de repositórios centralizados de bibliotecas (`/lib`,

`/usr/lib`, `/usr/local/lib`), contendo uma série de bibliotecas de diversas procedências. O programa `libtool`, ao gerar um arquivo de dependência para bibliotecas durante a sua compilação (arquivo de extensão `.la`), especifica, neste arquivo, os caminhos das bibliotecas utilizadas na *compilação* da biblioteca nova. Isto é uma prática errada, pois referencia, no sistema onde a biblioteca será instalada, caminhos que possivelmente só existem no sistema onde a biblioteca foi compilada. Em tempo de execução, estes arquivos não são relevantes, pois o carregador de bibliotecas dinâmicas `ld.so` procura as bibliotecas com base em uma variável de ambiente (`LD_LIBRARY_PATH`), a solução mais portável.

4.2 Imake

Uma das ferramentas de compilação de programas com uma das maiores bases de instalação existentes é o IMAKE ([2]), uma vez que faz parte da distribuição do XFREE86, a implementação livre do X WINDOW SYSTEM, o ambiente gráfico padrão do mundo UNIX. Esta ferramenta, apesar de haver sido criada com a finalidade de melhorar a portabilidade do X WINDOW SYSTEM, possui um dos mais sérios problemas de flexibilidade existentes: ela assume que a aplicação sendo compilada será instalada dentro da mesma hierarquia de diretórios que o XFREE86. Isto limita em muito a sua aplicabilidade, e por esta razão o IMAKE raramente é utilizado em novos projetos. Entretanto, existe um série de programas importantes no mundo do software livre que o utilizam. Em uma lista de discussão dedicada à questões de projeto do XFree86[1], este foi também apontado recentemente como um problema.

5 Conclusão

Este artigo descreveu os principais problemas relativos à portabilidade e flexibilidade em software livre enfrentados durante o desenvolvimento da distribuição GNU/Linux GoboLinux. Além de apontar soluções, espera-se que este artigo possa ser utilizado como referência pelos desenvolvedores de software livre de modo que estas limitações sejam reduzidas e que a interoperabilidade do software livre em geral seja aumentada. É importante notar que todas estas questões surgiram dentro de uma distribuição Linux, utilizando uma arquitetura x86, o que leva a crer que problemas de portabilidade em diferentes arquiteturas de hardware, utilizando bases de software alternativas seja ainda maior. Características peculiares da distribuição GoboLinux tornaram possível realizar uma avaliação da disseminação de caminhos *hard-coded* no sistema.

O artigo apresentou ainda uma ferramenta livre incluída no GoboLinux denominada FIBOSANDBOX que permite o isolamento das permissões de escrita a uma área restrita sem o isolamento das permissões de leitura como ocorre normalmente com `chroot`. Ferramentas como esta apontam um caminho onde a flexibilidade do software em relação a diferentes ambientes possa testada de forma automática. A instalação de aplicações no ambiente GoboLinux pode ser considerada também uma forma de emulação da instalação em ambientes não-UNIX, o que caracteriza uma forma nova de teste de portabilidade.

Referências

[1] Cygwin Information and Installation. `http://www.cygwin.com`

[2] Dubois, Paul. *Software Portability with Imake*, O'Reilly & Associates, 1996.

[3] Projeto Fink. `http://fink.sourceforge.net`

[4] Gooch, R. Linux Boot Scripts. `http://www.atnf.csiro.au/people/rgooch/linux/boot-scripts/`

[5] Hacker, S. et al. *The BeOS Bible*. Peachpit Press, 1999.

[6] Linux Standards Base. `http://www.linuxbase.org`

[7] Muhammad, H. Detsch, A. *"Uma nova proposta para a árvore de diretórios UNIX"*, III Workshop de Software Livre, Porto Alegre, 2002.

[8] Vaughan, G., Elliston, B., Tromey, T., Taylor, I. *"GNU Autoconf, Automake and Libtool"*, New Riders Publishing, 2000.

[1] `forum@xfree86.org`