## Compilando software para várias plataformas

# Produto de exportação


Apesar de ser um conceito incomum para a maior parte das pessoas, aqueles que desenvolvem para outras plataformas rapidamente sentem uma necessidade de bons ambientes e suporte para cross-compiling. POR PITER PUNK

Não sou o que se pode chamar de "desenvolvedor multiplataforma", mas já dei meus pulinhos, compilando um kernel para PlayStation (desenvolvido por uma empresa que não existe mais, a *runix.ru*), um para ARM, outro para PowerPC e uma meia dúzia de softwares para Athlon64.

Se eu tenho todas essas máquinas? Não, não tenho. E, mesmo que tivesse, como eu iria compilar o kernel do PlayStation no próprio PlayStation, que não tem sequer linha de comando, quanto mais um compilador?

É nessas horas que usamos um "cross-compiler", ou *compilador cruzado*. Fazer uma compilação cruzada nada mais é que compilar programas de uma plataforma em outra. Aproveitando o exemplo que citei acima, podemos compilar um kernel para MIPS R3000A (processador do PlayStation 1) dentro do meu PC, que usa um processador x86.

Outro uso típico da compilação cruzada é em máquinas com baixa capacidade de processamento. Por exemplo, um PowerMac 5500 (com um processador PowerPC 603e a 250 MHz) demora cerca de uma hora para compilar um kernel da série 2.4, mas em menos de 20 minutos meu Athlon já "velhinho" resolve o problema. Ou seja, apesar de gastar um tempo inicial configurando o Athlon para geração de binários para máquinas PowerPC, terei uma grande economia de tempo depois. Em alguns casos, o dispositivo alvo sequer tem memória suficiente para compilar coisa alguma, como o PlayStation com seus parcos 2MB de RAM.

## OK, você me convenceu...

Agora que já consegui convencê-lo a usar um cross-compiler, vamos às más notícias:

1. O processo envolve aproximadamente meia dúzia de passos exóticos;
2. Não funciona sempre, estando condicionado a diversos fatores externos, incluindo as fases da lua, o humor do presidente do Banco Central e a flutuação no valor de títulos voláteis da dívida externa do Zimbábue.

Quanto à primeira má notícia, a solução é razoavelmente simples: basta montar um "algoritmo" com os tais passos e seguilos sempre que necessário. A segunda má notícia envolve bem mais trabalho, incluindo buscas em listas de discussão, no Google, em ChangeLogs, em documentos obscuros na Biblioteca Nacional e até mesmo o sacrifício de virgens.

A primeira coisa que você vai precisar é baixar e instalar os softwares necessários para a compilação. Basicamente você vai precisar das *binutils*, o código fonte do kernel e o *gcc*, que são instalados por padrão na maioria das distribuições Linux.

Como estudo de caso, realizaremos a compilação de um binário para a arquitetura PowerPC em uma máquina x86. Isso irá determinar as versões de cada software que serão utilizadas (bem como os patches necessários). Sim, graças à segunda má notícia, não são todas as versões de software que funcionam com todas as plataformas de destino. Aliás, nem mesmo é possível a compilação cruzada entre certas plataformas – cada caso é um caso.

## ...mãos à obra!

Vale lembrar que todos os softwares utilizados são extremamente temperamentais. Vejamos os procedimentos básicos e dicas do que você pode precisar se quiser compilar programas para outras arquiteturas.

Utilizamos neste artigo o *binutils-2.15.90.0.3*, o *gcc-3.3.4*, a *glibc-2.3.2* e o kernel *2.4.18*. O código fonte dos três primeiros foi retirado do Slackware 10.0 e o kernel veio do Slackintosh 8.1. Você pode tentar com versões diferentes, mas é uma questão de "sorte".

## Preparando o terreno

Descompacte o kernel que você baixou. Uma boa dica é remover o link */usr/src/linux* e descompactar seu kernel no */usr/src*. A seguir, entre no diretório do kernel e execute:

```
# make ARCH=$ARQUITETURA symlinks ➥
include/linux/version.h
```

Esse comando irá corrigir os links que estão dentro do */usr/src/linux/include* e apontá-los para a arquitetura correta. No meu caso, eu fiz:

```
# make ARCH=ppc symlinks ➥
include/linux/version.h
```

Com isso, o link */usr/src/linux/include/asm* que, originalmente, apontava para o *asm-i386* (instruções em assembler do x86) passa a apontar para o asm-ppc, com as instruções em assembler para o PowerPC.

## O binutils

Agora começa a diversão. O binutils gera vários programas importantes para a compilação. Entre eles, dois essenciais:

- *as* – o assembler, que pega a saída do gcc e a converte para instruções da máquina; em seguida, as passa para o linker.
- *ld* – a função do linker é juntar vários arquivos objeto em um só, o que possibilita dividir programas extensos em pequenos pedaços e juntá-los no fim do processo em um único executável.

O *as* e o *ld* são específicos para cada arquitetura. Ou seja, o *as* para x86 não pode ser usado para gerar binários de PowerPC e vice-versa. Por isso, vamos compilá-los (junto com outros programas importantes) para ser executados em x86 e gerar código PowerPC.

Antes de descompactar o binutils, vamos prezar um pouco a organização do sistema e criar um diretório */opt/ppc*, para não misturar os nossos binários com os do sistema. Após criar esse diretório, descompacte o código-fonte do binutils, entre no diretório criado e digite:

```
# ./configure --prefix=/opt/ppc ➥
--target=powerpc-linux ➥
--disable-nls
```

Esse comando configura o binutils para ser instalado no diretório */opt/ppc*, tendo como plataforma alvo uma máquina PowerPC rodando Linux. O *--disable-nls* avisa para não compilar o suporte para outras linguagens além do inglês. Depois de tudo configurado, vamos aos já clássicos:

```
# make
# make install
```

Com isso podemos já compilar programas em assembly do PowerPC. Como a maior parte dos programas está em C, é uma boa idéia ter um compilador C, como por exemplo o gcc. Ah! Antes de passar para o próximo passo, inclua o diretório */opt/ppc/bin* no seu *PATH*, para que possamos utilizar os comandos que acabamos de compilar:

```
export PATH=$PATH:/opt/ppc/bin
```

## O compilador C

Este é o momento de compilar o gcc. Na teoria é extremamente simples. A seguinte linha de comando faz a configuração correta:

```
# ./configure --target=powerpc-➥
linux --prefix=/opt/ppc ➥
--disable-nls --disable-threads ➥
--disable-shared --enable-➥
languages=c --with-newlib
```

As opções *--target*, *--prefix* e *--disable-nls* já são nossas conhecidas de quando compilamos o binutils. Como novidades temos o *--disable-threads* e o *--disable-shared*, que desabilitam, respectivamente, o suporte a threads (que depende da glibc, que nós ainda não temos) e a bibliotecas compartilhadas (idem).

O *--enable-languages* diz quais das linguagens do gcc iremos compilar. No nosso caso, pelo menos nesse primeiro momento, compilaremos apenas o C, não tendo motivo para dar suporte a outras linguagens. O último argumento faz com que o gcc não use a glibc (que nós ainda não temos). Depois de tudo isso, poderíamos terminar o assunto com um:

```
# make
# make install
```

Não é? Pode até ser que para outras arquiteturas isso seja verdade, mas para o PowerPC é necessário utilizar um patch. E, sim, é muito comum ter que usar patches diversos: lembre-se da má notícia número 2; ela nos perseguirá ainda por bastante tempo. No nosso caso, devemos aplicar o patch *gcc-3.3.1-crosspc.diff*, encontrado em [1]. Para aplicar o patch basta digitar:

```
# patch -p1 < gcc-3.3.1-➥
crosspc.diff
```

Depois disso, podemos rodar novamente o *make* e o *make install*. Dessa vez, tudo vai correr bem e teremos nosso compilador C funcionando.

Um bom teste para ele é compilar o kernel do Linux, que é complexo e não usa a glibc. A compilação é bem simples: o primeiro passo é editar o arquivo */usr/src/linux/Makefile*. Comente a linha que começa com *ARCH=$(sh...)* e inclua uma linha com o seguinte conteúdo:

```
ARCH=ppc
```

Ah! Os nomes utilizados na compilação de outros programas nem sempre são os mesmos nomes utilizados no kernel, como vimos agora com *powerpc* e *ppc*. Também devemos editar a linha com:

```
CROSS_COMPILE =
```

E colocar o parâmetro powerpc-linux- (é, com o "-" no final) após o sinal de igualdade. Agora siga a ordem normal de compilação:

```
# make menuconfig
# make dep
# make clean
# make vmlinuz
```

Opa! Essa última linha não era para ser *make bzImage*? Bom, uma das coisas que temos que fazer quando pensamos em compilar para outras arquiteturas é tentar se livrar dos vícios x86. O *zImage* e o *bzImage* só existem porque não é possível ler diretamente arquivos de um certo tamanho no PC. A solução encontrada foi compactar o arquivo e fazer com que ele fosse descompactado durante a inicialização. Daí vêm os arquivos zImage (*Zipped Image*) e o bzImage (*Big Zipped Image*).

No PowerPC, e na maioria das outras arquiteturas, geramos diretamente a imagem do kernel, o arquivo *vmlinux*. Podemos verificar se o binário foi gerado corretamente utilizando o comando "file" para ver o tipo do arquivo:

```
# file vmlinux
vmlinux: ELF 32-bit MSB executable,➥
PowerPC or cisco 4500, version➥
1 (SYSV), statically linked,➥
not stripped.
```

Pois bem: se quisermos compilar apenas o kernel, terminamos por aqui o nosso toolkit para compilação cruzada. Porém, nem só de kernel vive o homem, mas de todo software que puder ser compilado...

## Compilando a glibc

Sim, se quisermos compilar outros softwares além do kernel, vamos precisar de uma libc. A mais utilizada no mundo Linux é a GNU libc, conhecida como *glibc*. Usei o código-fonte da *glibc-*

*2.3.2*, encontrado nos CDs do Slackware 10. Cuidado: o procedimento de compilação dela não é tão trivial.

Primeiro, devemos descompactar o código (arquivo *glibc-2.3.2.tar.bz2*); em seguida, entramos no diretório gerado (*glibc-2.3.2*). Lá dentro, devemos descompactar o código-fonte da linuxthreads (*glibc-linuxthreads-2.3.2.tar.bz2*):

```
# tar -xvjf ../glibc-➥
linuxthreads-2.3.2.tar.bz2
```

Ainda no diretório *glibc-2.3.2*, aplicamos um patch para possibilitar a compilação da glibc com o gcc 3.3.x:

```
# zcat ../glibc.gcc33x.diff.gz ➥
| patch -p1
```

Agora, vamos começar a trabalhar. Saia do diretório *glibc-2.3.2* e crie outro chamado "build" (ou o nome que você achar melhor). Entre nesse diretório e vamos configurar a glibc:

```
# ../glibc-2.3.2/configure ➥
--prefix=/opt/ppc --target=➥
powerpc-linux --host=powerpc-➥
linux --enable-add-ons=➥
linuxthreads --with-headers=➥
/usr/src/linux/include ➥
--with-binutils=/opt/ppc/bin
```

Os parâmetros significam:

**--enable-add-ons=linuxthreads**, que adiciona e faz com que seja compilado o suporte a threads;
**--with-headers**, que aponta para onde estão os includes do kernel que estamos utilizando; e
**--with-binutils**, que avisa onde estão instalados nossos binutils.

Depois de configurada a glibc, podemos compilá-la: *make all*. De preferência faça isso pouco antes de dormir, ou quando estiver passando um filme bom na TV, já que o processo demora bastante. Quando tudo tiver terminado, digite: *make install*.

Por algum motivo, a glibc instala vários dos seus arquivos em */opt/ppc/lib* e o gcc os procura em */opt/ppc/powerpc-linux/lib*. Resolvi isso copiando todos os arquivos de */opt/ppc/lib* para o */opt/ppc/powerpc-linux/lib*.

Aproveitando que estamos copiando coisas, vamos copiar os includes do kernel para o */opt/ppc/include*. Vamos fazer isso porque os programas devem ser compilados com os mesmos includes de kernel com que foi compilada a glibc, e nada melhor para garantir isso que copiar esses arquivos agora, logo depois de criarmos a glibc.

```
# cd /opt/ppc/include
# cp -a /usr/src/linux/include/➥
linux .
# cp -a /usr/src/linux/include/➥
asm-generic .
# cp -a /usr/src/linux/include/➥
asm-ppc asm
```

## Repeteco

O gcc completo usa a glibc. Enquanto não havia a glibc, não era possível compilar o gcc com ela. Por isso usamos o parâmetro *--with-newlib* na configuração do gcc. Agora que temos uma glibc, podemos também ter um gcc completo.

Entre de novo no diretório onde você descompactou o gcc, e digite:

```
# ./configure --target=powerpc-➥
linux --prefix=/opt/ppc --enable-➥
shared --enable-threads ➥
--enable-languages=c
# make all
# make install
```

O gcc será compilado, mas agora com suporte a threads, bibliotecas compartilhadas e o que mais você achar interessante. Agora você também pode compilar outras linguagens além de C e fazer um sistema de desenvolvimento completo! Atingimos nosso objetivo!

## Testando...

Depois de tudo pronto, o ideal é testar com um programa:

```
/* hello.c */
#include <stdio.h>
int main() {
  printf("Hello World!");
  return(0);
}
```

Esse programa imprime na tela *"Hello World!"* (sim, não estou sendo nem um pouco original). Antes de compilar, vamos exportar as seguintes variáveis:

```
# export C_INCLUDE_PATH=/opt/➥
ppc/include
# export CC=powerpc-linux-gcc
```

O primeiro comando indica onde devem ser encontrados os headers do sistema; afinal, queremos que o compilador use aqueles que nós criamos para PowerPC. Acredite, a falta dessa linha leva a desespero e noites mal dormidas. O segundo comando serve apenas para facilitar a vida se formos compilar várias coisas. A maioria dos programas e Makefiles já reconhece a variável CC e por isso usa o compilador indicado. O teste é simples:

```
# $CC -Wall hello.c
```

Ele vai dar algumas mensagens de erro, reclamando que o "main" não pode ser "void" (e não pode mesmo), mas vai compilar. Agora basta checar se o arquivo gerado é para PowerPC, o que pode ser feito com o comando "file", assim como fizemos com o kernel (*vmlinux*) que compilamos anteriormente.

Para garantir que nosso sistema funciona com programas "de verdade" vamos compilar um software qualquer. Escolhi o tar e peguei os sources do slackware 10.0. Vamos descompactar o arquivo *tar-1.14.tar.bz2* e compilá-lo:

```
# tar -xvjf tar-1.14.tar.bz2
# cd tar-1.14
# ./configure --host=powerpc-➥
linux --target=powerpc-linux ➥
--prefix=$HOME/ppc
# make
# make install
```

Para confirmar, novamente usamos o comando file. O resultado deve ser:

```
ELF 32-bit MSB executable, ➥
PowerPC or Cisco 4500, version ➥
1 (SYSV), dynamically linked➥
(uses shared libs), not stripped
```

Missão cumprida. Agora divirta-se compilando seus softwares favoritos para rodar numa torradeira, lavadora ou no microondas. ■

## INFORMAÇÕES

[1] *http://www.openslack.org/~piterpk/cross/gcc-3.3.1-crossppc.diff*