Požadavky a pravidla pro tvorbu přenositelného kódu. Organizace projektů a operačních systémů pro splnění přenositelnosti mezi různými architekturami CPU. Konverze vnitřních a vnějších/síťových formátů dat.(A4M35OSP)

Požadavky a pravidla pro tvorbu přenositelného kódu

Při realizaci softwarových systémů často nemůžeme zanedbat cílovou hardwarovou architekturu stroje a musíme psát kód tak, aby jej bylo možné snadno upravit (ideálně pouze rekompilovat) pro běh na jiných platformách. V okamžiku kdy je nutné přímo interagovat s některou hardwarovou komponentou počítače (řídící jednotky, SCADA systémy, …), se dostáváme až na úroveň, kdy musíme řešit věci, jako uspořádání bytů v proměnné apod. K základním požadavkům na přenositelnost zdrojového kódu patří:

- psát čistě a používat jen to, co je jazykem deklarováno,
- pokud je to možné používat pouze standardizovaná API - POSIX, IEEE Std. 1003.1,
- nepředpokládat pořadí byte/char ve slovech - little versus big endian,
- nepředpokládat počet bitů v adresační jednotce (používat stdint.h - int32_t, …),
- psát kód proti knihovnám a nikoliv přímo proti konkrétním voláním služeb operačního systému (konkrétním makrům, číslům volání, atd.).

Organizace projektů a operačních systémů pro splnění přenositelnosti mezi různými architekturami CPU

Aby bylo možné snadno přenášet projekty mezi různými operačními systémy a jejich verzemi a hardwarovými architekturami, bylo definováno několik standardů a vyvinuto několik nástrojů, které mají tuto přenositelnost usnadnit:

- **POSIX** - **Portable Operating System Interface** - standardy **IEEE 1003** a **ISO/IEC 9945** - definuje, co by měl poskytovat operační systém, jaké příkazy, jaké vlastnosti - na základě tohoto standardu je potom poměrně snadné převádět projekty mezi různými distribucemi Linuxu, UNIXY, atd. (uvádí se, že to je mnohdy jednodušší než převod projektů mezi různými verzemi Windows)
- **Filesystem hierarchy standard (FHS)** - dokument popisující, co má být ve kterém adresáři v linuxové distribuci, mezi typické adresáře patří
  - **/usr/bin** - binární programy (platformně závislé)
  - **/usr/local** - prefix do kterého je instalován software lokální pro tento stroj
  - **/usr/share** - sdílené (na hw platformě nezávislé) zdroje - často může být tento systém sdílen mezi různými stroji (např. pomocí NFS)
  - **/usr/lib** - knihovny (platformně závislé)
  - FHS popisuje, co by měl/neměl který adresář správně obsahovat

Kompilace GNU balíků

Nejrozšířenějším nástrojem pro kompilaci GNU balíků je systém Autotools. Tento systém se skládá z více nástrojů. Mezi hlavní a nejdůležitější patří:

- m4, aclocal, autoheader
- autoheader
- autoconf
- automake

Tyto nástroje zpracovávají vstupní soubory, které obsahují informace o projektu, závislostech, požadavcích na cílovou platformu atd. Jedná se zejména o soubory:

- Makefile.am
- configure.ac
- config.h.in

Proces sestavování (build) softwarového balíku se dělí na část, která probíhá u vývojáře (vyžaduje ke svému běhu větší množinu nástrojů) a část, kterou spouští klient.

Při přípravě distribuovatelného balíku (zpravidla nějaký *.tar.gz nebo *.zip archiv) je nutné spustit následující příkazy, které zpracovávají zmíněné soubory:

- **aclocal**: configure.ac → aclocal.m4

- **autoheader**: configure.ac → config.h.in
- **autoconf**: configure.ac → configure
- **automake**: Makefile.am → Makefile.in

Celý build systém je postaven na makro procesoru `m4` (zpracovává vstupní soubory a přepisuje makra na další řetězce, v tomto případě přepisuje příkazy/řetězce na výstupní shell skript, který potom běží na cílové instalační platformě). První příkaz (aclocal) zajistí přítomnost potřebných definic pro makroprocesor `m4` v souboru `aclocal.m4`.

Autoheader umožňuje vygenerování platformně závislého hlavičkového souboru, který potom může být includován do projektu a mohou z něj být získávány nejrůznější informace, které jsou známé až při instalaci na cílovém systému (např. cesty k souborům, atd).

Programy `autoconf` a `automake` jsou nejdůležitější - `autoconf` generuje soubor `configure`, který používá klient k vygenerování souboru `Makefile`, který je následně možné použít pro zkompilování projektu. Nástroj `automake` připravuje pro soubor `Makefile` šablony, které jsou následně zpracovávány skriptem configure pro vytvoření finálního `Makefile` souboru.

Balík, který je připraven pomocí výše zmíněných nástrojů již ke své kompilaci nástroje `autotool` a `automake` **nepotřebuje**.

Konfigurace před kompilací na cílovém nebo build systému potom vypadá následovně:

```
<srcdir>./configure –host=armlinuxgnueabi enablefeature withpackagex=/opt/x
```

Při této fázi jsou vygenerovány následující soubory:

- z `Makefile.in` je vygenerován `Makefile`
- z `config.h.in` je vygenerován `config.h`

Pomocí skriptu `configure` je možné ovlivňit i `CFLAGS=x, LDFLAGS=x` v prostředí.

Finální kompilace potom probíhá následovně:

```
make all
```

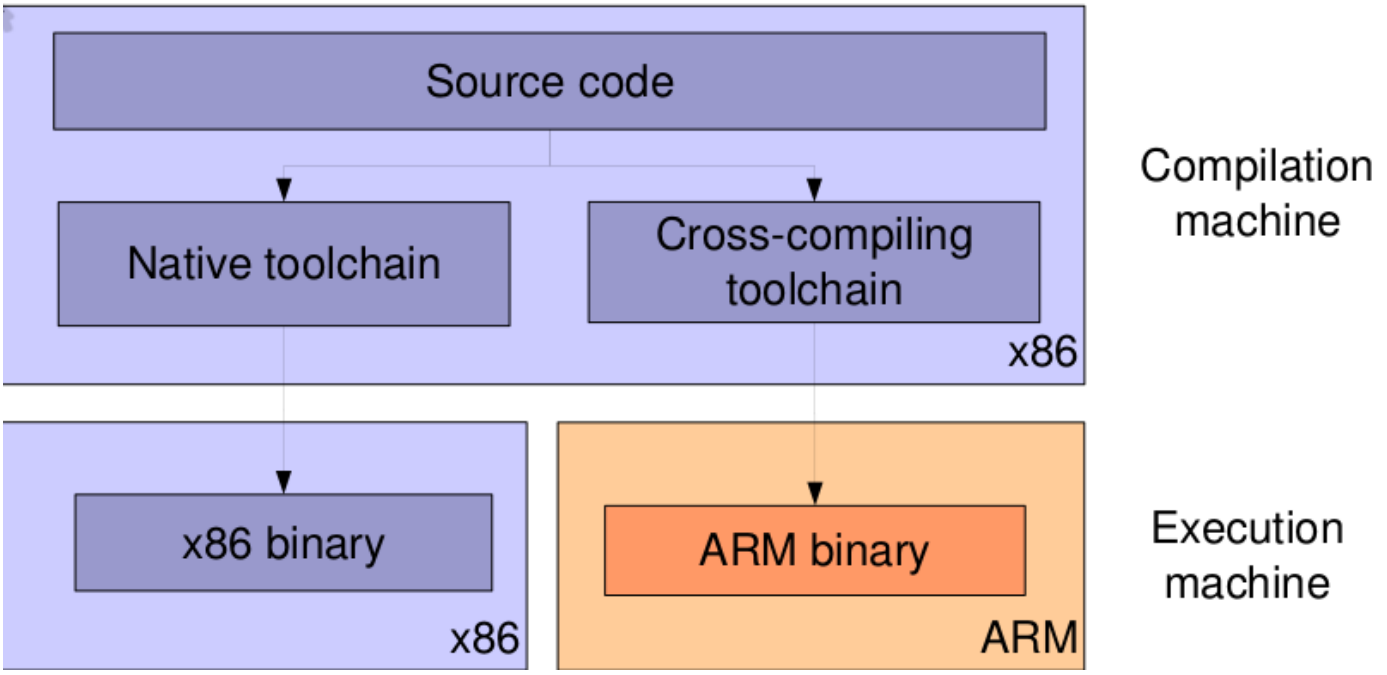
Instalace zkompilovaného projektu do cílového adresáře:

```
make DESTDIR=/packaging/root install
```

Portace kódu a křížový překlad

Většinou není nutné, aby každý uživatel kompiloval software ze zdrojových kódů. Je možné dodávat konkrétní verzi zkompilovanou pro cílovou platformu. Pro sestavení programu se používá build toolchain (v případě použití GCC se označuje jako **GNU Tool Chain**). Ten zahrnuje jednak vlastní kompiler (např. GCC - GNU Compiler Collection) a dále nástroje pro vytváření knihoven, linker atd.

Tento toolchain může kompilovat kód pro různé platformy. Na základě toho rozeznáváme **native toolchain** a **cross-compiling toolchain** (výsledné binární soubory běží na platformě toolchainu nebo na jiné platformě).



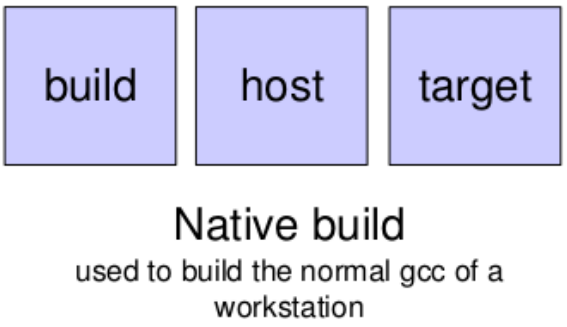
Nativní a křížové vývojové řetězce

Na základě toho, kde sestavujeme vlastní toolchain a kde jej používáme a jaké generuje cílové binární soubory rozlišujeme několik případů. V následujících sekcích jsou použity tři stroje (ve smyslu hw architektury):

- build - stroj na němž je **sestavován build toolchain** (např. gcc)
- host - stroj na němž **běží build toolchain** (zkompilovaný na stroji **build**)
- target - stroj na němž běží binární soubory generované překladačem běžícím na stroji **host**

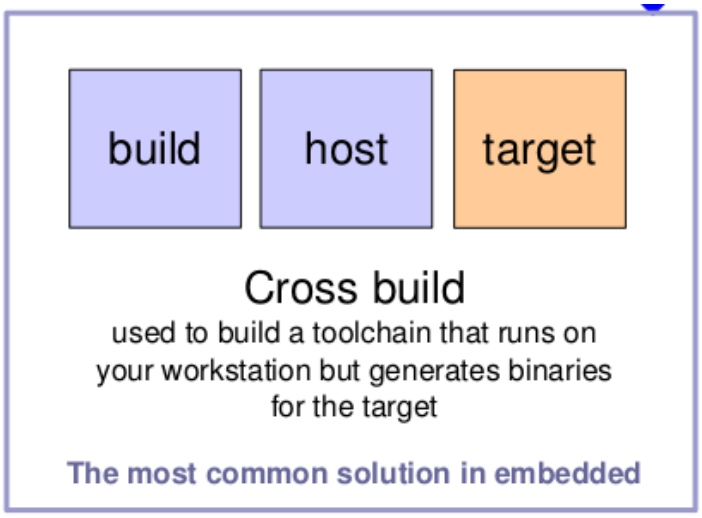
Native build

Nejběžnější je nativní build. Příkladem může být zkompilování gcc překladače v některé linuxové distribuci (e. g. Gentoo). Tento překladač je potom používán pro vygenerování binárek, které běží na téže platformě.



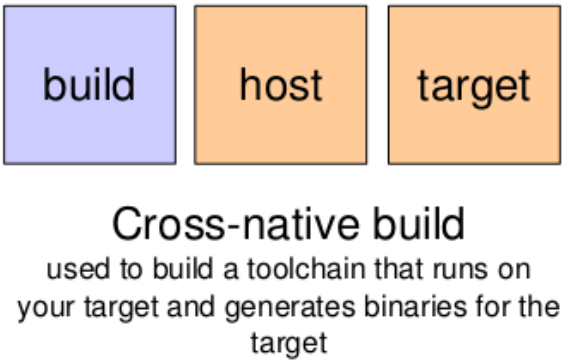
Cross build

Typický scénář použití pro běžný křížový překlad je u embedded systémů. Tyto systémy nejsou zpravidla dostatečně výkonné, abychom pro ně mohli zkompilovat překladač, který by na nich běžel a generoval binární soubory pro tuto platformu. Proto pracujeme např. v nějaké běžné linuxové distribuci (např. na architektuře core2, i686, …), kde máme překladač (host) a generujeme např. binární kód pro ARM procesor, který běží na cílovém embedded zařízení.



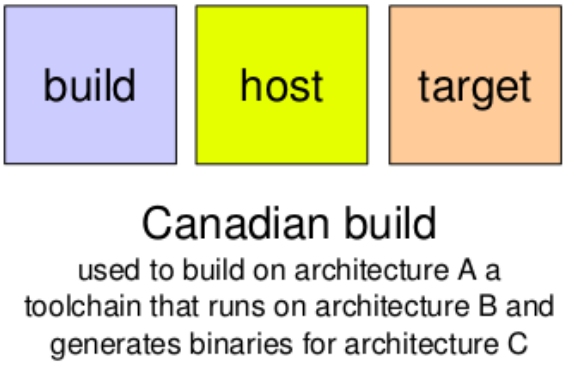
Cross-native build

Zde můžeme uvažovat např. rozdíl mezi 32bit a 64bit linuxovými distribucemi. Může se stát, že jsou servery, na kterých jsou kompilovány binární balíčky 64bitové. V tom případě je na 64 bitové architektuře zkompilován 32bitový překladač, který je možné následně používat pro generování binárních souborů pro cílovou platformu.



Canadian build

Kanadský build může být použit např. pokud vygenerujeme překladač běžící na 32bitové architektuře na 64bitovém stroji, ale tento překladač generuje třeba binární soubory pro embedded zařízení.



Konverze vnitřních a vnějších/síťových formátů dat

- little endian/big endian

- textové formáty: XML, HTML, SOAP, JSON
- external data representation (XDR)
- RPC, CORBA
- padding - zarovnávání packetů

Zdroje

- POSIX - Wikipedie [http://cs.wikipedia.org/wiki/POSIX]
- slides z přednášky o přenositelnosti sw, nová verze [http://rtime.felk.cvut.cz/osp/prednasky/osp-hw-and-porting.pdf]

statnice/oi_si_7.txt · Poslední úprava: 2011/06/05 11:14 autor: marty