Toolchain

dr inż. Krzysztof Konopko
e-mail: k.konopko@pb.edu.pl

Środowisko kompilacji skrośnej

Program wykładu:

- Wprowadzenie do kompilacji skrośnej.
- GCC
- Binutils.
- Biblioteka standardowa C.
- Pliki nagłówkowe jądra.
- Tworzenie toolchaina.

Wprowadzenie

- Zwykle narzędzia programistyczne dostępne dla GNU / Linux przeznaczone są do tworzenia systemu i aplikacji natywnych.
- W przypadku systemu wbudowanego jest to zwykle niemożliwe:
  - system wbudowany zazwyczaj ma zbyt małe zasoby (procesor, pamięć, urządzenia IO),
  - system wbudowany jest zazwyczaj dużo wolniejszy od stacji roboczej,
  - nie zawsze istnieje możliwość (potrzeba) instalacji całego środowiska deweloperskiego na komputerze docelowym.
- Dlatego najczęściej wykorzystuje się cross-kompilator, który umożliwia kompilowanie na stacji roboczej (architektura x86) oprogramowania dla systemu o architekturze ARM.

Podstawowe komponenty

GCC

GCC (ang. GNU Compiler Collection) - zestaw kompilatorów do różnych języków programowania rozwijany w ramach projektu GNU. W skład GCC wchodzą kompilatory wielu języków programowania w tym:

- C - gcc
- C++ - g++
- Objective-C - gobjc
- Fortran - g77 oraz nowa implementacja Fortranu 95 o nazwie GFortran
- Java - gcj
- Ada – gnat

GCC

Kompilatory wchodzące w skład GCC mogą być uruchamiane na wielu różnych platformach sprzętowych i systemowych. Za ich pomocą można generować kod wynikowy przeznaczony dla różnych procesorów w tym:

- Alpha
- ARM
- AVR
- IA-64
- MIPS
- Motorola M68000 i wiele innych układów tej firmy
- PowerPC
- SPARC/SPARC64
- x86/x86-64

Binutils

addr2line	Tłumaczy adres w komunikacie błędu (core dump) na odpowiedni plik źródłowy i numer linii. Używany przy debugowaniu.
ar	Archiwizator. Kilka skompilowanych plików . o (zazwyczaj biblioteki przygotowane do statycznego linkowania) może zostać zapakowanych do pojedynczego pliku .a.
as	Asembler GNU - używany przez GCC.
c++filt	Używany przez linker przy konsolidacji kodu napisanego w C++ lub Javie.
gprof	Kolejne narzędzie do debugowania, wyświetla statystyki użycia poszczególnych funkcji (Profiler).
ld	Linker (konsolidator), używany przez GCC.
nm	Wyświetla identyfikatory pojawiające się w danym pliku .o.
objdump	Wyświetla informacje o plikach skompilowanych.
ranlib	Pomocniczy do tworzenia paczek z wersjami statycznymi bibliotek. Generuje indeks identyfikatorów zawartych we wszystkich plikach w archiwum i przechowuje go w tym archiwum w specjalnym formacie.
readelf	Informacje o plikach wykonywalnych w formacie ELF. Pozwala na przykład stwierdzić, jakie biblioteki są wymagane przez dany plik wykonywalny.
size	Rozmiary poszczególnych sekcji pliku wykonywalnego.
strings	Program analizuje dowolny plik binarny, znajduje i wyświetla wszystkie zawarte w nim drukowalne ciągi znaków.
strip	Usuwa dodatkowe informacje z plików wykonywalnych i bibliotek. Program staje się mniejszy (i jego ładowanie trwa krócej), ale tracimy możliwość debugowania go.

Biblioteka standardowa C

glibc zawiera m.in.:

- interfejs do funkcji jądra systemu - tworzący pomost między wywołaniami systememowymi i prostymi fukcjami write() i read(),
- formatowane wejście i wyjście (słynne funkcje z grupy printf),
- kod zapewniający kompatybilność ze standardem interfejsu POSIX, ANSI C i innymi standardami,
- procedury alokacji pamięci,
- buforowane I/O,
- procedury matematyczne (w GNU/Linuksie ta część biblioteki znajduje się w osobnym pliku - libm.so),
- procedury operowania na łańcuchach tekstowych,
- obsługa lokalizacji,
- funkcje daty i czasu,
- funkcje pseudolosowe.

Pliki nagłówkowe jądra

Biblioteka standardowa i kompilowane programy muszą znać sposób komunikacji z jądrem w tym:

- dostępne wywołania systemowe `<asm-generic/unistd.h>`,
- definicje stałych `<asm-generic/fcntl.h>, <asm/fcntl.h>, <linux/fcntl.h>`,
- struktury danych `<asm/stat.h>` itd.

Dlatego biblioteka standardowa C wymaga plików nagłówkowych jądra.

Tworzenie toolchaina

Nazwa narzędzi tworzących toolchain wskazuje kompilatorowi i narzędziom Binutils dla jakiej architektury bajtów w słowie oraz ABI odbywa się kompilacja.

1) architektura: arm, armeb;
2) wariant: unknown, none; czasem wersja rdzenia: cortex_a8, nazwa producenta itp.; może określać również kolejność bajtów w słowie;
3) system operacyjny: linux, symbian, android - wskazuje, że generowany kod jest łączony dynamicznie z odpowiednią biblioteką standardową dla danego systemu; ten element nazwy może w ogóle nie występować, co oznacza, że toolchain służy do generowania kodu działającego bezpośrednio na procesorze;
4) wersja biblioteki standardowej/ABI: gnu, gnueabi, uclibcgnueabi,

np.: arm-unknown-linux-uclibcgnueabi-*

(E)ABI

Kompilator GCC dla architektury ARM pozwala wybrać ABI (ang. *Application Binary Interface*, ABI) czyli zestaw reguł i ustawień kompilacji i linkowania programów, które decydują o współpracy pomiędzy poszczególnymi programami, bibliotekami i systemem operacyjnym.

W przeciwieństwie do API (ang. *Application Programming Interface*), które określa interfejs dla programisty (dostępne funkcje i ich parametry), ABI dotyczy programów w wersji skompilowanej.

Domyślne ABI, którego instrukcje generuje nasz kompilator, definiujemy na etapie kompilacji GCC parametrem:

```
--with-mabi=<abi>
```

Na etapie korzystania z toolchaina, wybór ABI, dla którego jest kompilowany program, dokonywany jest parametrem:

```
-mabi=<abi>
```

Wersja procesora i architektury

Można określić rodzaj kodu generowanego przez kompilator w zależności od architektury danego procesora oraz sposób optymalizowania kodu typowego dla danego układu poprzez następujące opcje:

- `-mcpu=` - nazwa docelowego procesora. Na podstawie nazwy procesora określana jest wersja architektury, generowany kod asemblera zawierający specyficzne dla niej instrukcje,
- `-march=` - nazwa docelwej architektury. Określa architekturę dla której generowany jest kod asemblera.
- `-mtune=` - używany do optymalizacji kodu dla konkretnego procesora. Przyjmuje te same parametry, co `-mcpu=`. O ile `-mcpu=` generuje kod działający tylko na konkretnym procsorze/wersji architektury, o tyle `-mtune=` optymalizuje kod asemblera dla konkretnego modelu procesora (ale kod działa na każdym procesorze zgodnym z architekturą). Na przykład kod może być generowany dla procesora zgodnego z arm9, ale najszybciej działać na arm926ej-s: `-mcpu=arm9 -mtune=arm926ej-s`.

Kolejność bajtów w słowie

Istnieją zasadniczo dwa sposoby uporządkowania danych w pamięci:

- Big endian (spotykane także grubokońcowość) to forma zapisu danych, w której najbardziej znaczący bajt (zwany też grubym bajtem, z ang. high-order byte) umieszczony jest jako pierwszy.
- Little endian (spotykane także cienkokońcowość) to forma zapisu danych, w której mniej znaczący bajt (zwany też dolnym bajtem, z ang. low-order byte) umieszczony jest jako pierwszy.

Wyboru dokonujemy, ustalając odpowiednią nazwę narzędzi, oraz przy konfiguracji biblioteki standardowej.

Operacje zmiennoprzecinkowe

W przypadku braku możliwości sprzętowej realizacji, operacje zmiennoprzecinkowe mogą być emulowane na jeden z następujących sposobów:

- Kompilator GCC zastępuje, na etapie kompilacji, instrukcje zmiennoprzecinkowe, (opcja: -msoft-float).
- Programy korzystają z instrukcji zmiennoprzecinkowych, które są emulowane przez jądro systemu.

Decyzje o rodzaju emulacji podejmowane są na etapie konfiguracji toolchaina.

„Ręczne” tworzenie toolchaina

Budowa systemu od zera (z dostępnych źródeł na otwartych licencjach) pozwala lepiej zrozumieć, jak działa Linux, skąd się biorą poszczególne programy. Umożliwia dostosowanie ich do własnych potrzeb.

Wady to przede wszystkim dłuższy niż przy zastosowaniu automatycznych narzędzi czas poświęcony na przygotowanie i kompilację oraz konieczność samodzielnego dbania o ewentualne aktualizacje i reagowania na rozpoznane błędy bezpieczeństwa.

Gotowe toolchainy

Wady gotowych toolchainów:

- służą do ogólnych zastosowań i mają działać na jak największej ilości systemów, co w zasadzie wyklucza wykorzystanie optymalizacji;
- z reguły są dostarczane przez producentów sprzętu, mogą więc zawierać tylko optymalizacje na konkretny procesor/platformę, nie ma gwarancji, że skompilowany nimi kod w ogóle da się uruchomić na innym zestawie;
- dostarczają stabilnych (czytaj: starych) wersji kompilatora i narzędzi, często nieobsługujących najnowszych procesorów z ich nowymi funkcjami.

Zalety gotowych toolchainów:

- gotowe do użytku od razu po zainstalowaniu/rozpakowaniu;
- sprawdzone (przez to, że są powszechnie używane, jest szansa, że ktoś już natknął się na błąd, który właśnie się nam przytrafił);

Przykładowe gotowe toolchainy:

- Sourcery CodeBench toolchain,
- Linaro toolchain,
- gotowe pakiet toolchaina zawarty w dystrybucji np. w Ubuntu:

```
sudo apt-get install gcc-arm-linux-gnueabi
```

Gotowe toolchainy

Sourcery CodeBench:

CodeSourcery jest firmą, która sprzedaje gotowe toolchainy zapewniając wsparcie udostępnia jednak też i darmową wersję „lite”

http://mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/

Dla architektury ARM dostępne są między innymi następujące wersje:

- EABI - biblioteka standardowa newlibc - do kompilacji programów działających bezpośrednio na sprzęcie (bez systemu operacyjnego);
- uClinux - biblioteka standardowa uClibc;
- GNU/Linux - biblioteka GNU/libc;

Wystarczy pobrać instalator lub archiwum (skorzystamy z tego drugiego), rozpakować i ustawić zmienne środowiskowe.

Linaro:

Jest projektem rozpoczętym przez firmy: ARM, Freescale, IBM, Samsung, ST-Ericsson i Texas Instruments 3 czerwca 2010 roku. Ma na celu dostarczanie, zgodnie z zapowiedziami, co pół roku, „stabilnego, zoptymalizowanego zestawu bazowych narzędzi dla dystrybucji i deweloperów systemów wbudowanych". Zgodnie z założeniami, nacisk ma zostać położony na jądro oraz system bazowy.

`https://wiki.linaro.org/WorkingGroups/ToolChain`

Toolchainy generowane automatycznie

Innym rozwiązaniem jest użycie narzędzi, które automatyzują proces budowania toolchaina. Najbardziej popularne to:

- Crosstool-ng http://crosstool-ng.org/,
- Buildroot http://www.buildroot.net,
- Yocto https://www.yoctoproject.org/.

Inne toolchainy można znaleźć na:

http://elinux.org/Toolchains

Tworzenie toolchaina z zastosowaniem crosstool-ng

Kompilacja i instalacja ct-ng na podstawie wydanej wersji

```
wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-VERSION.tar.bz2
tar -xf crosstool-ng-VERSION.tar.bz2
cd crosstool-ng-VERSION
./configure --prefix=/some/place
make
make install
```

Kompilacja i instalacja ct-ng na podstawie wersji rozwojowej

```
git clone https://github.com/crosstool-ng/crosstool-ng
cd crosstool-ng
./bootstrap
./configure --enable-local
make
make install
```

Tworzenie toolchaina

```
./ct-ng help
./ct-ng armv7-rpi2-linux-gnueabihf
./ct-ng menuconfig
./ct-ng build
```