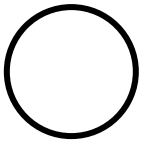
Toolchains i kompilacja

8 marca 2023

Wiktoria Kuna



Toolchain

Toolchain

- Zestaw narzędzi, który pozwala nam skompilować kod źródłowy tak, by pliki wykonywalne mogły zostać uruchomione na naszym urządzeniu docelowym.
- Musi zawierać: kompilator, konsolidator (linker) i biblioteki uruchomieniowe (runtime libraries)

Toolchain

GNU (GCC)

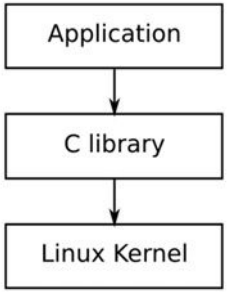
- Licencja GPL
- Kompatybilność
- Dojrzałość

LLVM (Clang)

- Licencja BSD
- Szybsza kompilacja
- Dokładniejsze i trafniejsze wiadomości diagnostyczne

GNU Toolchain - skład

- **Binutils** - narzędzia zawierające m.in. assembler i konsolidator
- **GNU Compiler Collection (GCC)** - kompilatory dla C i innych języków (zależnie od wersji)
- **Biblioteka języka C** - ustandaryzowane API, stanowiące główny interfejs między kernelem a aplikacjami



Rodzaje toolchainów (na nasze potrzeby)

Natywne

Toolchain działający na tym samym typie systemu (czasem tym samym systemie), co program, który generuje.

Skrośne

Toolchain działa na innym typie systemu niż system docelowy.

Ale skoro mam taką samą architekturę, to po co mi w ogóle osobny toolchain?

Ale skoro mam taką samą architekturę, to po co mi w ogóle osobny toolchain?

Osobne środowisko, niezmienność toolchainu podczas życia projektu.

Podczas pracy nad projektem nie zmieniaj toolchainu. Nie zmieniaj też wersji niczego, jeśli nie jesteś w 100% pewn_ .

Parametry do określenia przed zbudowaniem toolchainu

Architektura CPU

X86_64, AArch64, arm, …

Kolejność bajtów (endianness)

Big/Little-endian

Wsparcie dla liczb zmiennoprzecinkowych

Nie każdy procesor implementuje liczby zmiennoprzecinkowe na poziomie sprzętu.

ABI (Application Binary Interface)

Calling convention.

Nazwa GNU toolchainu - dekodujemy

Vendor - “dostawca” Kernel

arm — none — linux — gnueabihf

Architektura CPU, czasem kolejność bajtów jest oznaczona przez el/eb np. armeb

gnu lub musl; ABI, tu eabi = Extended Application Binary Interface a hf oznacza Hard-Float tj. zakładamy że nasz (docelowy) procesor ma jednostkę obsługującą liczby zmiennoprzecinkowe

Weryfikacja

```
$ gcc -dumpmachine
x86_64-pc-linux-gnu
```

```
$ arm-linux-gcc -dumpmachine
arm-buildroot-linux-gnueabi
```

Kompilacja skrośna z użyciem Makefile

Przy korzystaniu z makefile musimy pamiętać o wyspecyfikowaniu zmiennych środowiskowych **ARCH** i **CROSS_COMPILE.**

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
```

lub

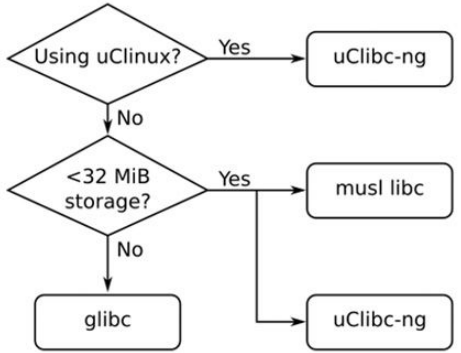
```
$ export ARCH=arm CROSS_COMPILE=arm-linux-gnueabi-
$ make
```

Biblioteka języka C

Możemy przebierać w opcjach, jedne z popularniejszych:

- **glibc** (libc6)- standardowa biblioteka C (GNU), implementuje najwięcej POSIX API. (LGPL)
- **musl libc** - mniejsza alternatywa dla libc. Dobra, gdy mamy mało pamięci. (MIT)
- **uClibc-ng** - µ jak mikrokontroler, początkowo rozwijana dla uClinux, teraz zaadaptowana dla Linuxa. (LGPL)

Propozycja z Mastering Embedded Linux Programing:



Jak używać toolchainu?

Musimy dodać go do ścieżki:

```
$ PATH=~/path/to/the/toolchain/bin:$PATH
```

Skompilujmy program helloworld:

```
$ arm-none-linux-gnueabihf-gcc hello.c -o hello
```

```
$ file hello
hello: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter
/lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, with debug_info, not stripped
```

Narzędzia, które znajdziemy w toolchainie

addr2line	Konwertuje adresy na nazwę i linię pliku źródłowego (z tabel debugowych)
ar	Narzędzie do tworzenia statycznych bibliotek
as	GNU assembler
c++filt	Rozwiązywanie symboli c++ i Javy
cpp	C preprocessor (rozwija #include, #define i inne dyrektywy)
elfedit	Aktualizowanie nagłówkóœ plików ELF
g++	GNU C++ frontend (zakłada, że pliki źródłowe zawierają kod C++)
gcc	GNU C frontend (zakłada, że pliki źródłowe zawierają kod C)

Narzędzia, które znajdziemy w toolchainie

gcov	Narzędzie pokrycia kodu
gdb	GNU Debugger
gprof	Narzędzie do profilowania kodu
ld	Konsolidator GNU
nm	Lista symboli z plików obiektowych
objcopy	Kopiowanie i translacja plików obiektowych
objdump	Wyświetlanie informacji o plikach obiektowych
ranlib	Tworzenie/Modyfikowanie indeksów w bibliotekach statycznych

Narzędzia, które znajdziemy w toolchainie

readelf	Informacja o plikach ELF
size	Informacja o wielkości sekcji i wielkości pliku
strings	Wypisywalne napisy/znaki z plików
strip	Do pozbawienia pliku symboli debugowych

To skąd wziąć ten cały toolchain?

Możemy pobrać gotowy

Zalety

- Niski wkład pracy
- Szybki proces (pobieranie + instalacja)

Wady

- Jesteśmy ograniczeni do konfiguracji zapewnionej przez dostawcę
- Musimy polegać na dostawcy w sprawie aplikowania patchy (w tym dotyczących bezpieczeństwa)

To skąd wziąć ten cały toolchain?

Możemy pobrać gotowy

Tylko z zaufanych źródeł, np.:

- Producent SoC/płytki, z którą pracujemy,
- Konsorcja zapewniające niskopoziomowe wsparcie dla danej architektury, np. Linario oferuje gotowe toolchainy dla arma (i regularnie je aktualizuje),
- Inne firmy dostarczające narzędzi do Linuxa
- Pakiety oferowane dla naszej dystrybucji,
- SDK (software development kit) oferowane przez systemy budowania dla urządzeń wbudowanych (np. Yocto)

To skąd wziąć ten cały toolchain?

Możemy zbudować z wykorzystaniem narzędzi do budowania (Yocto, Buildroot)

Zalety

- Stosunkowo niski wkład pracy
- Od razu będziemy mogli zbudować pozostałe niezbędne rzeczy dla naszego systemu (kernel, system plików i bootloader)

Wady

- Te narzędzia są potężne, budowanie toolchainu to tylko krok pośredni w całym procesie

To skąd wziąć ten cały toolchain?

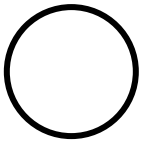
Do it yourself!

Zalety

- Możliwość największej personalizacji
- Dużo wybuchów
- Dużo błędów (tj. dużo sytuacji prowokujących naukę)

Wady

- brak



Krok w dorosłość

Budujemy własny toolchain z LFS

Instrukcje krok po kroku znajdziemy w
||| części podręcznika LFS

Chcąc zbudować toolchain własnoręcznie należy się ich dokładnie trzymać (chyba że wiemy, co robimy).
W prezentacji opowiemy sobie co nas czeka.

Etapy budowy

1. Budowanie toolchainu (cross-toolchain).
2. Użycie dopiero zbudowanego toolchainu do zbudowania kilku narzędzi.
3. Zmiana środowiska przy pomocy chroot i skonstruowanie pozostałych narzędzi potrzebnych nam do zbudowania systemu docelowego.

Kompilacja skrośna na LFS

Build - Maszyna, na której budujemy programy

Host - Maszyna/system, na której uruchamiamy zbudowane programy.

Target - System docelowy, dla którego kompilator produkuje kod.

Etap	Build	Host	Target	Działanie
1	PC	PC	LFS	Budujemy kompilator skrośny cc1 korzystając z cc-pc na PC.
2	PC	LFS	LFS	Budujemy kompilator cc-lfs korzystając z cc1 na PC.
3	LFS	LFS	LFS	Przebudowujemy i testujemy cc-lfs korzystając z cc-lfs na LFS. (opcjonalnie)

Kompilacja skrośna na LFS

Musimy zwrócić uwagę na jeszcze jeden fakt:

- Potrzebujemy skompilować bibliotekę glibc (będąc wiernymi podręcznikowi LFS) dla naszej maszyny LFS tj. korzystając z kompilatora skrośnego cc1.
- Kompilator cc1 korzysta z wewnętrznej biblioteki, która zapewnia funkcje niedostępne w zestawie instrukcji assemblera (libgcc). Libgcc musi zostać skonsolidowana z glibc, by działać poprawnie.
- Co więcej, libstdc++ też musi zostać skonsolidowana z glibc.

I co teraz?

Kompilacja skrośna na LFS

Najpierw zbudujemy starszą wersję libgcc, w której brakuje kilku istotnych funkcjonalności (jak obsługa wyjątków czy wątków), potem zbudujemy glibc i libstdc++ przestarzałym kompilatorem.

Poprzez wybrakowanie libgcc, w libstdc++ będzie brakować nam niektórych funkcjonalności.

Gdy zbudujemy już GCC (cc1 z tabelki) poinstruujemy go, żeby przebudował libgcc i libstdc++, ale linkujemy libstdc++ do nowo przebudowanej libgcc (zamiast tej starej wersji).

Kompilacja cross-toolchainu

Binutils - iteracja I	Konsolidator, assembler i inne niezbędne narzędzia. Kompilujemy jako pierwsze, GCC i Glibc korzystają z konsolidatora i assemblera.
GCC - iteracja I	GNU Compiler Collection, m.in. kompilatory C i C++.
Linux API Headers	Nagłówki, które udostępnią API kernela dla Glibc.
Glibc	Główna biblioteka języka C. Podstawowe funkcje do obsługi pamięci, obsługi plików, arytmetyki etc.
Libstdc++	Standardowa biblioteka C++

Kompilacja narzędzi tymczasowych

M4	Zawiera procesor makr (potrafi skopiować plik wraz z rozszerzeniem jego makr)
Ncurses	Obsługa terminala niezależnie od jego typu
Bash	Bourne-Again Shell
Coreutils	Podstawowe programy dla systemu operacyjnego
Diffutils	Programy pokazujące różnicę w plikach i katalogach
File	Potrzebna do rozróżniania typów plików
Findutils	Wyszukiwanie plików

Kompilacja narzędzi tymczasowych

Gawk	Manipulacja plików tekstowych
Grep	Przeszukiwanie zawartości plików
Gzip	(De)Kompresowanie plików
Make	Generowanie plików wykonywalnych (i nie tylko)
Patch	Aplikowanie “patchy”
Sed	Edytor strumieniowy
Tar	(Roz)Pakowanie archiw typu tar

Kompilacja narzędzi tymczasowych

Xz	(De)Kompresowanie plików
Binutils - iteracja II	
Gcc - iteracja II	

Brakujące puzzle

- Zmiana własności. Cała hierarchia katalogów należy do użytkownika z maszyny host. Wpp. na naszej maszynie byłoby UID nienależące do żadnego użytkownika (tzn. mogłoby takie zostać przypisane do nowego użytkownika).
- Przygotowanie wirtualnych systemów plików. Tworzymy katalogi do zamontowania, montujemy i populujemy /dev (by kernel mógł zamontować devtmpfs podczas bootowania) i montujemy pozostałe utworzone katalogi.

```
mkdir -pv $LFS/{dev,proc,sys,run}

mount -v --bind /dev $LFS/dev

mount -v --bind /dev/pts $LFS/dev/pts
mount -vt proc proc $LFS/proc
mount -vt sysfs sysfs $LFS/sys
mount -vt tmpfs tmpfs $LFS/run
```

Brakujące puzzle

- Przełączamy się w środowisko chroot-em by móc zbudować pozostałe aplikacje potrzebne w systemie.

```
chroot "$LFS" /usr/bin/env -i   \
    HOME=/root                  \
    TERM="$TERM"                \
    PS1='(lfs chroot) \u:\w\$ ' \
    PATH=/usr/bin:/usr/sbin     \
    /bin/bash --login
```

Brakujące puzzle

- Tworzymy brakujące katalogi

```
mkdir -pv /{boot,home,mnt,opt,srv}

mkdir -pv /etc/{opt,sysconfig}
mkdir -pv /lib/firmware
mkdir -pv /media/{floppy,cdrom}
mkdir -pv /usr/{,local/}{include,src}
mkdir -pv /usr/local/{bin,lib,sbin}
mkdir -pv /usr/{,local/}share/{color,dict,doc,info,locale,man}
mkdir -pv /usr/{,local/}share/{misc,terminfo,zoneinfo}
mkdir -pv /usr/{,local/}share/man/man{1..8}
mkdir -pv /var/{cache,local,log,mail,opt,spool}
mkdir -pv /var/lib/{color,misc,locate}

ln -sfv /run /var/run
ln -sfv /run/lock /var/lock

install -dv -m 0750 /root
install -dv -m 1777 /tmp /var/tmp
```

Brakujące puzzle

- Tworzymy niezbędne pliki i linki symboliczne, jak
  - /etc/hosts
  - /etc/passwd
  - /etc/group
  - /proc/self/mounts -> /etc/mtab (lista zamontowanych systemów plików)

Narzędzia ciąg dalszy

Gettext	Lokalizacja i internacjonalizacja. (Pozwala kompilować programy i wyświetlać wiadomości w języku użytkownika)
Bison	Parser
Perl	Practical Extraction and Report Language
Python	Środowisko Pythona
Texinfo	Pisanie, czytanie, konwersja stron info.
Util-linux	Różne narzędzia systemowe

Backup

Wychodzimy ze środowiska chroot za pomocą polecenia exit.

```
mountpoint -q $LFS/dev/shm && umount $LFS/dev/shm
umount $LFS/dev/pts
umount $LFS/{sys,proc,run,dev}

cd $LFS
tar -cJpf $HOME/lfs-temp-tools-11.3.tar.xz .
```

Przywracanie

Wystarczy rozpakować:

```
cd $LFS
rm -rf ./*
tar -xpf $HOME/lfs-temp-tools-11.3.tar.xz
```

Dziękuję