

Kompilacja Kernela Linux

Metoda stara i nowa

Rafał Hrabia

nr indeksu 296583

Informatyka I st.

Uniwersytet Marii Curie-Skłodowskiej

14 czerwca 2022

Spis treści

1	Przygotowanie	2
2	Metoda stara	4
3	Metoda nowa	6
4	Wnioski	7

Rozdział 1

Przygotowanie

W pierwszej kolejności po zalogowaniu się i przejściu do katalogu `/usr/src` musimy zdobyć link do paczki z plikami źródłowymi kernela. Z racji tego, że korzystam z wirtualnej maszyny bez interfejsu graficznego posłużę się komputerem, który hostuje maszynę wirtualną. Wchodzimy na stronę i przy użyciu menu kontekstowego kopiujemy do schowka bezpośredni link do pliku(rys. 1.1).



Rysunek 1.1: Kopiowanie linku

Do pobrania pliku jedynym słusznym wyborem będzie komenda `wget`(rys. 1.2).

```
root@slack:/usr/src# wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
--2022-06-14 17:24:06-- https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
Translacja cdn.kernel.org (cdn.kernel.org)... 151.101.1.176, 151.101.65.176, 151.101.129.176, ...
Łączenie się z cdn.kernel.org (cdn.kernel.org)[151.101.1.176]:443... połączono.
Żądanie HTTP wysłano, oczekiwanie na odpowiedź... 200 OK
Długość: 129859840 (124M) [application/x-xz]
Zapis do: 'linux-5.18.3.tar.xz'

linux-5.18.3.tar.xz      100%[=====>] 123,84M  1,78MB/s   w 69s
2022-06-14 17:25:15 (1,78 MB/s) - zapisano 'linux-5.18.3.tar.xz' [129859840/129859840]

root@slack:/usr/src# ls -l
razem 126824
-rw-r--r-- 1 root root 129859840 cze  9 10:42 linux-5.18.3.tar.xz
root@slack:/usr/src#
```

Rysunek 1.2: Pobieranie kernela

Jako, że pobrany plik jest to plik archiwum używamy na nim komendy *tar -xvpf* (rys 1.3).

Dla przypomnienia (flagi polecenia *tar*):

- x – wyodrębnia wymienione pliki
- v – wypisuje nazwy wszystkich plików
- f – określa nazwę pliku archiwum tar
- p - zachowuje ustawienia dostępu do plików

```
linux-5.18.3/virt/kvm/vfio.h
linux-5.18.3/virt/lib/
linux-5.18.3/virt/lib/Kconfig
linux-5.18.3/virt/lib/Makefile
linux-5.18.3/virt/lib/irqbypass.c
root@slack:/usr/src# ls -l
razem 126828
drwxrwxr-x 24 root root    4096 cze  9 10:30 linux-5.18.3/
-rw-r--r--  1 root root 129859840 cze  9 10:42 linux-5.18.3.tar.xz
root@slack:/usr/src#
```

Rysunek 1.3: Rozpakowywanie archiwum

Jak można zauważyć został stworzony nowy folder zawierający pliki z archiwum.

Teraz musimy przekopiować *config* znajdujący się w katalogu */proc* (rys 1.4). Ponieważ plik konfiguracji ma rozszerzenie wskazujące na to, że to archiwum gzip używamy polecenia *zcat*.

```
root@slack:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.3# ls -la | grep .config
-rw-rw-r-- 1 root root    59 cze  9 10:30 .configconfig
-rw-rw-r-- 1 root root 237785 cze 14 17:53 .config
-rw-rw-r-- 1 root root   555 cze  9 10:30 Kconfig
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 1.4: Kopiowanie konfiguracji

Przyda się zrobić kopię zapasową oryginalnego configu. W tym celu kopiujemy plik *.config* do przykładowo pliku *.config.bak* (rys. 1.5).

```
root@slack:/usr/src/linux-5.18.3# cp .config .config.bak
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 1.5: Kopia zapasowa

Rozdział 2

Metoda stara

Przyszła pora na wywołanie polecenia *make localmodconfig*. Po pomyślnym wykonaniu jesteśmy informowani o nadpisaniu konfiguracji do pliku *.config* (rys. 2.1).

```
Test user/kernel boundary protections (TEST_USER_COPY) [N/m/?] n
Test BPF filter functionality (TEST_BPF) [N/m/?] n
Test blackhole netdev functionality (TEST_BLACKHOLE_DEV) [N/m/?] n
Test find_bit functions (FIND_BIT_BENCHMARK) [N/m/y/?] n
Test firmware loading via userspace interface (TEST_FIRMWARE) [N/m/y/?] n
sysctl test driver (TEST_SYSCTL) [N/m/y/?] n
udelay test driver (TEST_UDELAY) [N/m/y/?] n
Test static keys (TEST_STATIC_KEYS) [N/m/?] n
kmod stress tester (TEST_KMOD) [N/m/?] n
Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] n
Test heap/page initialization (TEST_MEMINIT) [N/m/y/?] n
Test freeing pages (TEST_FREE_PAGES) [N/m/y/?] n
Test floating point operations in kernel space (TEST_FPU) [N/m/y/?] n
Test clocksource watchdog in kernel space (TEST_CLOCKSOURCE_WATCHDOG) [N/m/y/?] n
#
# configuration written to .config
#
root@slack:/usr/src/linux-5.18.3#
root@slack:/usr/src/linux-5.18.3#
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 2.1: Wynik polecenia *make localmodconfig*

Nastąpiła wielka chwila - kompilacja jądra. W tym celu trzeba znowu użyć polecenia *make*. Tym razem użyjemy parametru *-j< liczba >* co pozwoli nam uruchomić kompilację kernela na wielu rdzeniach(lub wątkach?). W moim przypadku użyję liczby 8.

```
root@slack:/usr/src/linux-5.18.3# make -j8 bzImage
```

Rysunek 2.2: Kompilowanie kernela

Kompilacja kernela przebiegła w zawrotnym czasie 4 minut i 45 sekund (mniej więcej).

```
CC      arch/x86/boot/compressed/cmdline.o
CC      arch/x86/boot/video-bios.o
HOSTCC  arch/x86/boot/tools/build
CC      arch/x86/boot/compressed/error.o
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CPUSTR  arch/x86/boot/cpustr.h
RELOCS  arch/x86/boot/compressed/vmlinux.relocs
HOSTCC  arch/x86/boot/compressed/mkpiggy
CC      arch/x86/boot/compressed/cpuflags.o
CC      arch/x86/boot/cpu.o
CC      arch/x86/boot/compressed/early_serial_console.o
CC      arch/x86/boot/compressed/kaslr.o
CC      arch/x86/boot/compressed/acpi.o
CC      arch/x86/boot/compressed/misc.o
LZMA     arch/x86/boot/compressed/vmlinux.bin.lzma
MKPIGGY arch/x86/boot/compressed/piggy.S
AS       arch/x86/boot/compressed/piggy.o
LD       arch/x86/boot/compressed/vmlinux
ZOFFSET  arch/x86/boot/zoffset.h
OBJCOPY  arch/x86/boot/vmlinux.bin
AS       arch/x86/boot/header.o
LD       arch/x86/boot/setup.elf
OBJCOPY  arch/x86/boot/setup.bin
BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready (#1)
root@slack: /usr/src/linux-5.18.3#
```

Rysunek 2.3: Kompilowanie kernela - wynik

Kompilację modułów zleca się również komendą `make`, ale zamieniamy parametr pozycyjny `bzImage` na `modules`.

```
root@slack: /usr/src/linux-5.18.3# make -j8 modules
```

Rysunek 2.4: Kompilowanie modułów

Budowanie modułów zakończyło się zaskakująco szybko - w niecałe 36 sekund (tak, stałem ze stoperem w ręce). Pytanie czy tak powinno być? Czy to może zasługa Intel Core i9? Podejrzewam, że "stara metoda" ograniczyła ilość modułów do niezbędnego minimum. Przyjmijmy więc, że wszystko jest w porządku i kontynuujmy.

```
LD [M]  drivers/video/fbdev/core/fb_sys_fops.ko
LD [M]  drivers/video/fbdev/core/syscopyarea.ko
LD [M]  drivers/video/fbdev/core/sysfillrect.ko
LD [M]  drivers/video/fbdev/core/sysimgblt.ko
LD [M]  drivers/virt/vboxguest/vboxguest.ko
LD [M]  net/802/garp.ko
LD [M]  net/802/mrp.ko
LD [M]  net/802/p8022.ko
LD [M]  net/802/psnap.ko
LD [M]  net/802/stp.ko
LD [M]  net/8021q/8021q.ko
LD [M]  net/ipv6/ipv6.ko
LD [M]  net/llc/llc.ko
LD [M]  net/rfkill/rfkill.ko
LD [M]  net/wireless/cfg80211.ko
LD [M]  sound/ac97_bus.ko
LD [M]  sound/core/snd-pcm.ko
LD [M]  sound/core/snd-timer.ko
LD [M]  sound/pci/ac97/snd-ac97-codec.ko
LD [M]  sound/core/snd.ko
LD [M]  sound/soundcore.ko
LD [M]  sound/pci/snd-intel8x0.ko
root@slack: /usr/src/linux-5.18.3#
```

Rysunek 2.5: Kompilowanie modułów - wynik

Rozdział 3

Metoda nowa

Rozdział 4

Wnioski