

# Sprawozdanie – Kompilacja Jądra

Karol Kula

Zanim przeszedłem do systemu linux ze względu na to, że mam dosyć mocny komputer najpierw zwiększyłem ilość ramu oraz rdzeni dla wirtualnej maszyny, dzięki temu oraz dzięki temu, że posiadam dysk ssd na złączu m2 kompilacja powinna przebiec bardzo szybko.

## System

RAM: 11423 MB  
Procesory: 6  
Boot Order: Dysk twardy, Napęd optyczny  
Akceleracja: VT-x/AMD-V, Zagnieżdżone stronicowanie, PAE/NX

## Pobranie kernela

Na początku oczywiście trzeba pobrać odpowiednią wersję jądra w tym celu odwiedziłem stronę [www.kernel.org](http://www.kernel.org). W momencie pisania tego sprawozdania najnowsza stabilna wersja jądra to wersja 5.18.3. Następnie uruchomiłem maszynę wirtualną jednak ze względu na to że do maszyny wirtualnej nie mogłem wkleić linka do jądra użyłem narzędzia PowerShell i za pomocą SSH załogowałem się na konto root oczywiście wcześniej sprawdzając adres ip maszyny poleceniem `ip addr show`.

```
WybierzOpenSSH SSH client
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\karol> ssh 192.168.1.37 -l root
Password:
Last login: Fri Jun 10 06:41:50 2022
Linux 5.15.27-smp.
root@slack:~#
```

Następnie przeszedłem do katalogu `/usr/src` dzięki poleceniu `cd`

```
OpenSSH SSH client
root@slack:/# cd /usr/src/
root@slack:/usr/src#
```

Oraz użyłem polecenie `wget` z wcześniej skopiowanym linkiem, dzięki czemu rozpoczęło się pobieranie jądra

```
WybierzOpenSSH SSH client
root@slack:/# cd /usr/src/
root@slack:/usr/src# wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
--2022-06-10 06:57:57-- https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.3.tar.xz
Translacja cdn.kernel.org (cdn.kernel.org)... 151.101.113.176, 2a04:4e42:1b::432
Łączenie się z cdn.kernel.org (cdn.kernel.org)[151.101.113.176]: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 47%[=====] 58,84M 988KB/s eta 67s
```

Następnie w celu rozpakowania pobranej paczki użyłem polecenia `tar -xvf linux-5.18.3.tar.xz`  
Końcówkę procesu rozpakowania widać na poniższym zrzucie ekranu.

```
OpenSSH SSH client
linux-5.18.3/virt/kvm/
linux-5.18.3/virt/kvm/Kconfig
linux-5.18.3/virt/kvm/Makefile.kvm
linux-5.18.3/virt/kvm/async_pf.c
linux-5.18.3/virt/kvm/async_pf.h
linux-5.18.3/virt/kvm/binary_stats.c
linux-5.18.3/virt/kvm/coalesced_mmio.c
linux-5.18.3/virt/kvm/coalesced_mmio.h
linux-5.18.3/virt/kvm/dirty_ring.c
linux-5.18.3/virt/kvm/eventfd.c
linux-5.18.3/virt/kvm/irqchip.c
linux-5.18.3/virt/kvm/kvm_main.c
linux-5.18.3/virt/kvm/kvm_mm.h
linux-5.18.3/virt/kvm/pfncache.c
linux-5.18.3/virt/kvm/vfio.c
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#
```

Metoda stara

Po rozpakowaniu paczki pojawił się katalog *linux-5.18.3*, do którego wszedłem.

```
OpenSSH SSH client
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
linux-5.18.3/  linux-5.18.3.tar.xz
root@slack:/usr/src# cd linux-5.18.3
root@slack:/usr/src/linux-5.18.3#
```

Następnie przekopiowałem aktualną konfigurację jądra do pliku *.config*

```
OpenSSH SSH client
root@slack:/usr/src# ls
linux-5.18.3/  linux-5.18.3.tar.xz
root@slack:/usr/src# cd linux-5.18.3
root@slack:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.3#
```

Oraz dla pewności sprawdziłem poleceniem *nano* czy plik na pewno został stworzony.

```
OpenSSH SSH client
GNU nano 6.0 .config
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86 5.15.27 Kernel Configuration
#
CONFIG_CC_VERSION_TEXT="gcc (GCC) 11.2.0"
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=110200
CONFIG_CLANG_VERSION=0
CONFIG_AS_IS_GNU=y
CONFIG_AS_VERSION=23700
CONFIG_LD_IS_BFD=y
CONFIG_LD_VERSION=23700
CONFIG_LLD_VERSION=0
CONFIG_CC_CAN_LINK=y
CONFIG_CC_CAN_LINK_STATIC=y
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_CC_HAS_ASM_GOTO_OUTPUT=y

^G Pomoc      ^O Zapisz    ^W Wyszukaj  ^K Wytnij   ^T Wykonaj   ^C Lokalizacja M-U Odwołaj   M-A Ustaw znacz
^X Wyjdź      ^R Wczyt.plik ^\ Zastąp  ^U Wklej    ^J Wyjustuj  ^/ Do linii  M-E Odtwórz  M-6 Kopiuj
```

Następnie użyłem polecenia *make localmodconfig* w celu wygenerowania pliku.

```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.3# nano .config
root@slack:/usr/src/linux-5.18.3# make localmodconfig
using config: '.config'
*
* Restart config...
*
*
* Timers subsystem
*
Timer tick handling
  1. Periodic timer ticks (constant rate, no dynticks) (HZ_PERIODIC)
> 2. Idle dynticks system (tickless idle) (NO_HZ_IDLE)
choice[1-2?]: 2
Old Idle dynticks config (NO_HZ) [Y/n/?] y
High Resolution Timer Support (HIGH_RES_TIMERS) [Y/n/?] y
Clocksource watchdog maximum allowable skew (in us) (CLOCKSOURCE_WATCHDOG_MAX_SKEW_US) [100] (NEW) _
```

Następnie zostały wyświetlone pytania o poszczególne moduły jądra, zostawiłem wszystko domyślnie tak jak było na zajęciach.

```
OpenSSH SSH client
Perform selftest on IDA functions (TEST_IDA) [N/m/y/?] n
Test module loading with 'hello world' module (TEST_LKM) [N/m/?] n
Test module for compilation of bitops operations (TEST_BITOPS) [N/m/?] n
Test module for stress/performance analysis of vmalloc allocator (TEST_VMALLOCC) [N/m/?] n
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#
```

Następnie przeszedłem już do samej kompilacji jądra dzięki użyciu polecenia *make -j6 bzImage*, użyłem parametru *-j6* ze względu na to, że moja wirtualna maszyna wykorzystuje sześć rdzeni.

```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# make -j6 bzImage
```

Tak jak przypuszczałem, dzięki udostępnieniu wirtualnej maszynie większej ilości zasobów, jądro skompilowało się całkiem szybko, a dokładnie to w 8 minut i 13 sekund.

```
OpenSSH SSH client
CC      arch/x86/boot/video-bios.o
HOSTCC  arch/x86/boot/tools/build
CC      arch/x86/boot/compressed/string.o
CC      arch/x86/boot/compressed/cmdline.o
CPUTSTR arch/x86/boot/cpustr.h
CC      arch/x86/boot/compressed/error.o
CC      arch/x86/boot/cpu.o
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
RELOCS  arch/x86/boot/compressed/vmlinux.relocs
HOSTCC  arch/x86/boot/compressed/mkpiggy
CC      arch/x86/boot/compressed/cpuflags.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#
```

Następnie przeszedłem do kompilacji modułów dzięki użyciu polecenia *make -j6 modules*

```
OpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# make -j6 modules_
```

Skompilowanie modułów było bardzo szybkie dokładnie zajęło to 53 sekundy.

```
OpenSSH SSH client
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/core/snd.ko
LD [M]  sound/pci/ac97/snd-ac97-codec.ko
LD [M]  sound/pci/snd-intel8x0.ko
LD [M]  sound/soundcore.ko
root@slack:/usr/src/linux-5.18.3#
```

Następnie przeszedłem do instalacji modułów, dzięki użyciu polecenia *make modules\_install*

```
OpenSSH SSH client
LD [M]  sound/pci/snd-intel8x0.ko
LD [M]  sound/soundcore.ko
root@slack:/usr/src/linux-5.18.3# make modules_install
```

Instalacja modułów była prawie natychmiastowa.

```
WybierzOpenSSH SSH client
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/p8022.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/psnap.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/802/stp.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/8021q/8021q.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/ipv6/ipv6.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/llc/llc.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/rfkill/rfkill.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/wireless/cfg80211.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/ac97_bus.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/core/snd-pcm.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/core/snd-timer.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/core/snd.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/pci/ac97/snd-ac97-codec.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/pci/snd-intel8x0.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/soundcore.ko
DEPMOD /lib/modules/5.18.3-smp
root@slack:/usr/src/linux-5.18.3#
```

Następnie użyłem komendy *cp* do skopiowania plików potrzebnych do uruchomienia.

```
WybierzOpenSSH SSH client
root@slack:/usr/src/linux-5.18.3# cp arch/x86/boot/bzImage /boot/vmlinuz-staraMetoda-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp System.map /boot/System.map-staraMetoda-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp .config /boot/config-staraMetoda-5.18.3-smp
root@slack:/usr/src/linux-5.18.3#
```

Następnie przeszedłem do tworzenia linku symbolicznego w systemie dla tablicy symboli kernela. W tym celu przeszedłem do katalogu */boot/* dzięki użyciu komendy *cd*, następnie usunąłem starą tablicę symboli za pomocą komendy *rm*, i za pomocą komendy *ln* utworzyłem link symboliczny.

```
OpenSSH SSH client
root@slack:/# cd /boot/
root@slack:/boot# rm System.map
root@slack:/boot# ln -s System.map-staraMetoda-5.18.3-smp System.map
root@slack:/boot#
```

Po skopiowaniu plików i stworzeniu linku symbolicznego przeszedłem do generowania komendy, która stworzy ramdisk:

```
OpenSSH SSH client
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-smp
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot#
```

Używam tak wygenerowanej komendy tylko ze zmieniając nazwą ostatniego parametru:

*mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-staraMetoda-5.18.3-smp.gz*

Jak widać na poniższym zrzucie wszystko przebiegło bez problemu.

```
OpenSSH SSH client
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-staraMetoda-5.18.3-smp.gz
49039 bloków
/boot/initrd-staraMetoda-5.18.3-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

Aby można było skorzystać z tak przygotowanego kernela dodałem wpis LILO

Najpierw otworzyłem plik konfiguracyjny LILO za pomocą polecenia *nano*

```
OpenSSH SSH client
root@slack:/boot# nano /etc/lilo.conf
```

Po przewinięciu pliku w dół, można zobaczyć miejsce gdzie dodaje się wpisy.

```
OpenSSH SSH client
GNU nano 6.0 /etc/lilo.conf
# VESA framebuffer console @ 800x600x64k
#vga=788
# VESA framebuffer console @ 800x600x32k
#vga=787
# VESA framebuffer console @ 800x600x256
#vga=771
# VESA framebuffer console @ 640x480x64k
#vga=785
# VESA framebuffer console @ 640x480x32k
#vga=784
# VESA framebuffer console @ 640x480x256
#vga=769
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
root = /dev/sda1
label = "Slackware 15.0"
read-only
# Linux bootable partition config ends
-
^G Pomoc      ^O Zapisz    ^W Wyszukaj  ^K Wytnij    ^T Wykonaj   ^C Lokalizacja M-U Odwołaj   M-A Ustaw znacz
^X Wyjdź      ^R Wczyt.plik ^\ Zastąp   ^U Wklej     ^J Wyjustuj  ^/ Do linii  M-E Odtwórz  M-6 Kopiuje
```

Następnie dodałem odpowiedni wpis dla metody starej.

```
OpenSSH SSH client
GNU nano 6.0 /etc/lilo.conf Zmieniony
# VESA framebuffer console @ 800x600x256
#vga=771
# VESA framebuffer console @ 640x480x64k
#vga=785
# VESA framebuffer console @ 640x480x32k
#vga=784
# VESA framebuffer console @ 640x480x256
#vga=769
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
root = /dev/sda1
label = "Slackware 15.0"
read-only
# Linux bootable partition config ends
image = /boot/vmlinuz-staraMetoda-5.18.3-smp
root = /dev/sda1
initrd = /boot/initrd-staraMetoda-5.18.3-smp.gz
label = "Stara Metoda"
read-only
^G Pomoc      ^O Zapisz    ^W Wyszukaj  ^K Wytnij    ^T Wykonaj   ^C Lokalizacja M-U Odwołaj   M-A Ustaw znacz
^X Wyjdź      ^R Wczyt.plik ^\ Zastąp   ^U Wklej     ^J Wyjustuj  ^/ Do linii  M-E Odtwórz  M-6 Kopiuje
```

Następnie zaktualizowałem LILO poleceniem *LILO*.

```
WybierzOpenSSH SSH client
root@slack:~# lilo
Warning: LBA32 addressing assumed
Added Slackware_15.0 *
Added Stara_Metoda +
One warning was issued.
root@slack:~#
```

Oraz komendą *reboot* zrestartowałem maszynę.

```
Windows PowerShell
root@slack:~# reboot

Broadcast message from root@slack.localhost (pts/0) (Fri Jun 10 14:36:08 2022):

The system is going down for reboot NOW!
root@slack:~# Connection to 192.168.1.37 closed by remote host.
Connection to 192.168.1.37 closed.
PS C:\Users\karol>
```

Ponieważ zrestartowałem maszynę utraciłem z nią połączenie w powershellu ale przeszedłem bezpośrednio do okna wirtualnej maszyny.

```
Slackware (gr0) (Slax) [Uruchomiona] - Oracle VM VirtualBox

OS Selection

Slackware_15.0
Stara_Metoda

slackware
linux

Select an OS to boot, or hit <Tab> for a LILO prompt:
```

Jak widać wpis został poprawnie dodany do LILO, i nowa wersja kernela jest możliwa do wyboru.

Następnie wybrałem wpis Stara\_Metoda i na szczęście cały system uruchomił się bez problemu.

```
Slackware (gr0) (Slax) [Uruchomiona] - Oracle VM VirtualBox
DUID 00:04:a9:74:57:d7:c2:cc:91:48:ad:2d:c2:58:7e:16:3d:85
eth0: waiting for carrier
eth0: carrier acquired
eth0: IAD 27:61:7a:e9
eth0: soliciting a DHCP lease
eth0: offered 10.0.2.15 from 10.0.2.2
eth0: probing address 10.0.2.15/24
eth0: leased 10.0.2.15 for 86400 seconds
eth0: adding route to 10.0.2.0/24
eth0: adding default route via 10.0.2.2
forked to background, child pid 685
eth1: polling for DHCP server
dhcpcd-9.4.1 starting
DUID 00:04:a9:74:57:d7:c2:cc:91:48:ad:2d:c2:58:7e:16:3d:85
eth1: waiting for carrier
eth1: carrier acquired
eth1: IAD 27:3d:3e:c4
eth1: soliciting a DHCP lease
eth1: offered 192.168.1.37 from 192.168.1.1
eth1: probing address 192.168.1.37/24
eth1: leased 192.168.1.37 for 86400 seconds
eth1: adding route to 192.168.1.0/24
eth1: adding default route via 192.168.1.1
forked to background, child pid 775
Starting system message bus: /usr/bin/dbus-uuidgen --ensure ; /usr/bin/dbus-daemon --system
Starting elogind: /lib/elogind/elogind --daemon
Starting OpenSSH SSH daemon: /usr/sbin/sshd
Starting ACPI daemon: /usr/sbin/acpid
Updating MIME database: /usr/bin/update-mime-database /usr/share/mime &
Updating gtk.immodules:
  /usr/bin/update-gtk-immodules &
Updating gdk-pixbuf.loaders:
  /usr/bin/update-gdk-pixbuf-loaders &
Compiling GSettings XML schema files:
  /usr/bin/glib-compile-schemas /usr/share/glib-2.0/schemas &
Starting crond: /usr/sbin/crond -l notice
Starting atd: /usr/sbin/atd -b 15 -l 1
Loading /usr/share/kbd/keymaps/i386/qwerty/pl.map.gz
Starting gpm: /usr/sbin/gpm -n /dev/mouse -t inps2

Welcome to Linux 5.18.3-smp i686 (tty1)

slack login: root
Password:
Last login: Fri Jun 10 14:42:51 on tty1
Linux 5.18.3-smp.
root@slack:~#
```

## Metoda Nowa

Ponieważ metoda nowa jest bardzo podobna do metody starej pozwolę sobie na nieco mniej szczegółowe opisy.

W folderze linux-5.18.3/scripts/kconfig znajduje się skrypt streamline\_config.pl



```
Windows PowerShell
GNU nano 6.0 streamline_config.pl
# It gives you the ability to turn off all the modules that are
# not loaded on your system.
#
# Howto:
#
# 1. Boot up the kernel that you want to stream line the config on.
# 2. Change directory to the directory holding the source of the
#    kernel that you just booted.
# 3. Copy the configuration file to this directory as .config
# 4. Have all your devices that you need modules for connected and
#    operational (make sure that their corresponding modules are loaded)
# 5. Run this script redirecting the output to some other file
#    like config_strip.
# 6. Back up your old config (if you want too).
# 7. copy the config_strip file to .config
# 8. Run "make oldconfig"
#
# Now your kernel is ready to be built with only the modules that
# are loaded.
#
# Here's what I did with my Debian distribution.
#
# cd /usr/src/linux-2.6.10
# cp /boot/config-2.6.10-1-686-smp .config
# ~/bin/streamline_config > config_strip
# mv .config config_sav
^G Pomoc      ^O Zapisz     ^W Wyszukaj   ^K Wytnij     ^T Wykonaj    ^C Lokalizacja M-U Odwołaj      M-A Ustaw znacz
^X Wyjdź      ^R Wczyt.plik ^_ Zastąp     ^U Wklej      ^J Wyjustuj   ^_ Do linii   M-E Odtwórz    M-6 Kopiuj
```

To właśnie tego skryptu używa się do kompilacji jądra nową metodą, w skrypcie jest opisane co należy robić krok po kroku.

Najpierw podobnie jak w metodzie starej stworzyłem plik konfiguracyjny:

```
Windows PowerShell
root@slack:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.3#
```

A potem przeszedłem już do użycia skryptu.

```
Windows PowerShell
root@slack:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.3# ./scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'
root@slack:/usr/src/linux-5.18.3#
```

Po konfiguracji przeszedłem do wykonania komendy *make oldconfig*. I podobnie jak wcześniej wszystkie opcję zatwierdziłem przyciskiem enter.

```
Windows PowerShell
root@slack:/usr/src/linux-5.18.3# make oldconfig
.config:440:warning: symbol value 'm' invalid for I8K
.config:8107:warning: symbol value 'm' invalid for VIDEO_ZORAN_DC30
.config:8108:warning: symbol value 'm' invalid for VIDEO_ZORAN_ZR36060
.config:8109:warning: symbol value 'm' invalid for VIDEO_ZORAN_BUZ
.config:8110:warning: symbol value 'm' invalid for VIDEO_ZORAN_DC10
.config:8111:warning: symbol value 'm' invalid for VIDEO_ZORAN_LML33
.config:8112:warning: symbol value 'm' invalid for VIDEO_ZORAN_LML33R10
.config:8113:warning: symbol value 'm' invalid for VIDEO_ZORAN_AVS6EYES
.config:9703:warning: symbol value 'm' invalid for CRYPTO_LIB_BLAKE2S_GENERIC
*
* Restart config...
*
*
* Timers subsystem
*
Timer tick handling
  1. Periodic timer ticks (constant rate, no dynticks) (HZ_PERIODIC)
  > 2. Idle dynticks system (tickless idle) (NO_HZ_IDLE)
choice[1-2]: 2
Old Idle dynticks config (NO_HZ) [Y/n/?] y
High Resolution Timer Support (HIGH_RES_TIMERS) [Y/n/?] y
Clocksource watchdog maximum allowable skew (in us) (CLOCKSOURCE_WATCHDOG_MAX_SKEW_US) [100] (NEW) _
```

Po wykonaniu tej komendy przeszedłem do kompilacji jądra.

```
Windows PowerShell
root@slack:/usr/src/linux-5.18.3# make -j6 bzImage_
```

W przypadku nowej metody jądro skompilowało się w 11 minut i 15 sekund.

```
Windows PowerShell
CC      arch/x86/boot/version.o
AS      arch/x86/boot/compressed/head_32.o
VOFFSET arch/x86/boot/compressed/./voffset.h
CC      arch/x86/boot/compressed/cmdline.o
OBJCOPY arch/x86/boot/compressed/vmlinux.bin
CC      arch/x86/boot/compressed/error.o
RELOCS  arch/x86/boot/compressed/vmlinux.relocs
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 (#2)
root@slack:/usr/src/linux-5.18.3#
```

Tak jak poprzednio po kompilacji jądra trzeba skompilować moduły.

```
Windows PowerShell
root@slack:/usr/src/linux-5.18.3# make -j6 modules_
```

Ku mojemu zdziwieniu moduły kompilowały się 38 minut i 13 sekund

```
WybierzWindows PowerShell
LD [M]  sound/soc/sof/xtensa/snd-sof-xtensa-dsp.ko
LD [M]  sound/synth/emux/snd-emux-synth.ko
LD [M]  sound/synth/snd-util-mem.ko
LD [M]  sound/usb/bcd2000/snd-bcd2000.ko
LD [M]  sound/usb/6fire/snd-usb-6fire.ko
LD [M]  sound/usb/caiaq/snd-usb-caiaq.ko
LD [M]  sound/usb/line6/snd-usb-line6.ko
LD [M]  sound/usb/hiface/snd-usb-hiface.ko
LD [M]  sound/usb/line6/snd-usb-pod.ko
LD [M]  sound/usb/line6/snd-usb-podhd.ko
LD [M]  sound/usb/line6/snd-usb-toneport.ko
LD [M]  sound/usb/line6/snd-usb-variax.ko
LD [M]  sound/usb/misc/snd-ua101.ko
LD [M]  sound/usb/snd-usb-audio.ko
LD [M]  sound/usb/usx2y/snd-usb-usx2y.ko
LD [M]  sound/usb/snd-usbmidi-lib.ko
LD [M]  sound/usb/usx2y/snd-usb-us1221.ko
LD [M]  sound/virtio/virtio_snd.ko
LD [M]  sound/x86/snd-hdmi-lpe-audio.ko
LD [M]  virt/lib/irqbypass.ko
root@slack:/usr/src/linux-5.18.3#
```

Następnie zainstalowałem moduły komendą *make modules\_install* (instalowały się znacznie dłużej niż w metodzie starej)

Następnie użyłem komendy *cp* do skopiowania plików potrzebnych do uruchomienia.

```
Windows PowerShell
root@slack:/usr/src/linux-5.18.3# cp arch/x86/boot/bzImage /boot/vmlinuz-nowaMetoda-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp System.map /boot/System.map-nowaMetoda-5.18.3-smp
root@slack:/usr/src/linux-5.18.3# cp .config /boot/config-nowaMetoda-5.18.3-smp
root@slack:/usr/src/linux-5.18.3#
```

Następnie przeszedłem do tworzenia linku symbolicznego w systemie dla tablicy symboli kernela. W tym celu przeszedłem do katalogu `/boot/` dzięki użyciu komendy `cd`, następnie usunąłem starą tablicę symboli za pomocą komendy `rm`, i za pomocą komendy `ln` utworzyłem link symboliczny.

```
Windows PowerShell
root@slack:/boot# System.map
-bash: System.map: nie znaleziono polecenia
root@slack:/boot# rm System.map
root@slack:/boot# ln -s System.map-nowaMetoda-5.18.3-smp System.map
root@slack:/boot#
```

Po skopiowaniu plików i stworzeniu linku symbolicznego przeszedłem do generowania komendy, która stworzy ramdisk:

```
Windows PowerShell
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.3-smp
#
# mkinitrd_command_generator.sh revision 1.45
#
# This script will now make a recommendation about the command to use
# in case you require an initrd image to boot a kernel that does not
# have support for your storage or root filesystem built in
# (such as the Slackware 'generic' kernels').
# A suitable 'mkinitrd' command will be:
mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot#
```

I użyłem tej komendy tylko modyfikując nazwę

`mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-nowaMetoda-5.18.3-smp.gz`

```
Windows PowerShell
root@slack:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-nowaMetoda-5.18.3-smp.gz
49399 bloków
/boot/initrd-nowaMetoda-5.18.3-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

Ostatnim krokiem było oczywiście dodanie wpisu do lilo z tak przygotowanym kernelem

```
Windows PowerShell
GNU nano 6.0 /etc/lilo.conf Zmieniony
# VESA framebuffer console @ 640x480x64k
#vga=785
# VESA framebuffer console @ 640x480x32k
#vga=784
# VESA framebuffer console @ 640x480x256
#vga=769
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only
image = /boot/vmlinuz-staraMetoda-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-staraMetoda-5.18.3-smp.gz
  label = "Stara Metoda"
  read-only
image = /boot/vmlinuz-nowaMetoda-5.18.3-smp
  root = /dev/sda1
  initrd = /boot/initrd-nowaMetoda-5.18.3-smp.gz
  label = "Nowa Metoda"
```

Oraz zaktualizowanie lilo poleceniem `lilo`

```
Windows PowerShell
Added Stara_Metoda +
Added Nowa_Metoda +
One warning was issued.
root@slack:~#
```

Po zresetowaniu maszyny wirtualnej widzimy że wszystko się w porządku i mamy do wyboru nową oraz starą metodę



A nowa metoda bez problemu się uruchamia:

```
Slackware (gr0) (Slax) [Uruchomiona] - Oracle VM VirtualBox
Starting haveged entropy daemon: /sbin/haveged
Starting the network interfaces...
eth0: polling for DHCP server
dhcpcd-9.4.1 starting
DUID 00:04:a9:74:57:d7:c2:cc:91:48:ad:2d:c2:58:7e:16:3d:85
eth0: waiting for carrier
eth0: carrier acquired
eth0: IAID 27:61:7a:e9
eth0: soliciting a DHCP lease
eth0: offered 10.0.2.15 from 10.0.2.2
eth0: probing address 10.0.2.15/24
eth0: leased 10.0.2.15 for 86400 seconds
eth0: adding route to 10.0.2.0/24
eth0: adding default route via 10.0.2.2
forked to background, child pid 976
eth1: polling for DHCP server
dhcpcd-9.4.1 starting
DUID 00:04:a9:74:57:d7:c2:cc:91:48:ad:2d:c2:58:7e:16:3d:85
eth1: waiting for carrier
eth1: carrier acquired
eth1: IAID 27:3d:3e:e4
eth1: soliciting a DHCP lease
eth1: offered 192.168.1.37 from 192.168.1.1
eth1: probing address 192.168.1.37/24
eth1: leased 192.168.1.37 for 86400 seconds
eth1: adding route to 192.168.1.0/24
eth1: adding default route via 192.168.1.1
forked to background, child pid 1068
Starting system message bus: /usr/bin/dbus-uuidgen --ensure ; /usr/bin/dbus-daemon --system
Starting elogind: /lib/elogind/elogind --daemon
Starting OpenSSH SSH daemon: /usr/sbin/sshd
Starting ACPI daemon: /usr/sbin/acpid
Updating MIME database: /usr/bin/update-mime-database /usr/share/mime &
Updating gtk.immodules:
/usr/bin/update-gtk-immodules &
Updating gdk-pixbuf.loaders:
/usr/bin/update-gdk-pixbuf-loaders &
Compiling GSettings XML schema files:
/usr/bin/glib-compile-schemas /usr/share/glib-2.0/schemas &
Starting crond: /usr/sbin/crond -l notice
Starting atd: /usr/sbin/atd -b 15 -l 1
Loading /usr/share/kbd/keymaps/i386/qwerty/pl.map.gz
Starting gpm: /usr/sbin/gpm -m /dev/mouse -t imps2

Welcome to Linux 5.18.3-smp i686 (tty1)

slack login:
```

## Podsumowanie

Podczas kompilacji jądra dwiema metodami nie trafiłem na żadne problemy. Obie metody są tak naprawdę bardzo podobne jednak metoda nowa wydaje się lekko łatwiejsza ponieważ w skrypcie jest opisane co i jak należy zrobić. Mimo tego, że metoda nowa wydaje się prostsza to kompilacją modułów i samego jądra trwała tą metodą prawie 50 minut dłużej co może być nie pożądane gdy komuś się śpieszy. Co ciekawe zauważyłem, że w zasadzie za każdym razem system z kernelem skompilowanym nową metodą odpala się szybciej niż ten z kernelem skompilowanym metodą starą. Więc osoba stawiająca na wydajność systemu powinna wybrać metodę nową.