# 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.



#### Pobranie kernela

Na początku oczywiście trzeba pobrać odpowiednią wersję jądra w tym celu odwiedziłem stronę www.kernel.org. W momencie pisania tego sprawozdania najnowsza stabilna wersją 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 zalogowałem się na konto root oczywiście wcześniej sprawdzając adres ip maszyny poleceniem *ip addr show*.

Następnie przeszedłem do katalogu /usr/src dzięki poleceniu cd



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



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

#### 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/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

```
Proof@slack:/usr/src# 1s
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
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
                              ^O Zapisz
^R Wczyt.
                                                                                                                        ^T Wykonaj
^J Wyjustu
                                                            ^W Wyszukaj
^\ Zastąp
                                                                                                                                                           Lokalizacja <mark>M-U</mark> Odwołaj
Do linii M-E Odtwórz
                                                                                               Wytnij
Wklej
                                                                                                                                                                                                                   M-A Ustaw znacz
     Pomoc
                                   Wczyt.plik
                                                                 Zastąp
                                                                                                                             Wyjustuj
```

Następnie użyłem polecenia make localmodconfig w celu wygenerowania pliku.

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

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.



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

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

Skompilowanie modułów było bardzo szybkie dokładnie zajęło to 53 sekundy.

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

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

Instalacja modułów była prawie natychmiastowa.

```
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/gsnap.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/8021q.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/pv6/ipv6.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/lic/lic.ko
INSTALL /lib/modules/5.18.3-smp/kernel/net/lic/lic.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-ac97-codec.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/kernel/sound/soundcore.ko
DEPMOD /lib/modules/5.18.3-smp/kernel/sound/soundcore.ko
DEPMOD /lib/modules/5.18.3-smp/kernel/sound/soundcore.ko
```

Następnie użyłem komendy cp do skopiowania plików potrzebnych do uruchomienia.

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

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

```
DopenSSH 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 zmienią 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.

```
# 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.

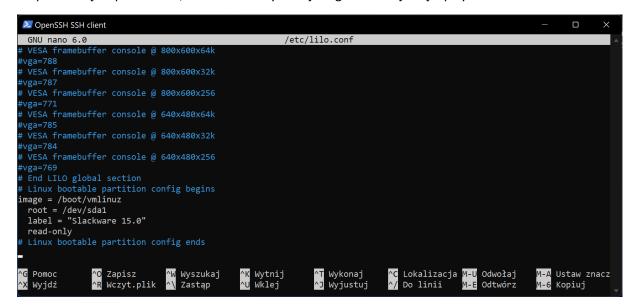
root@slack:/boot# _
```

Aby można było skorzystać z tak przygotowanego kernela dodałem wpis LILO

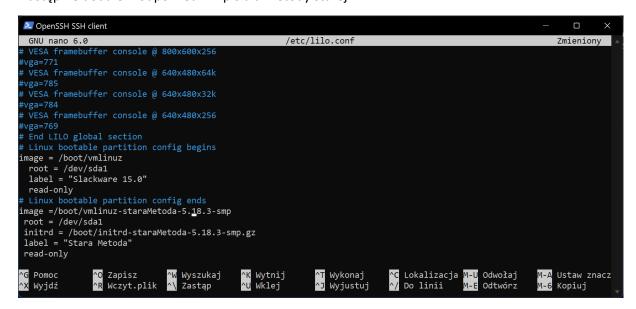
Najpierw otworzyłem plik konfiguracyjny LILO za pomocą polecenia nano

```
PopenSSH SSH client − □ ×
root@slack:/boot# nano /etc/lilo.conf
```

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



Następnie dodałem odpowiedni wpis dla metody starej.



Następnie zaktualizowałem LILO poleceniem LILO.

```
WybierzOpenSSH SSH client

— □ ×

root@slack:~# lilo

Warning: LBA32 addressing assumed

Added Slackware_15.0 *

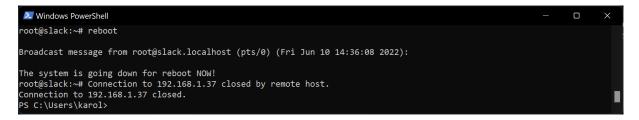
Added Stara_Metoda +

One warning was issued.

root@slack:~# _

■
```

Oraz komendą reboot zrestartowałem maszynę.



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



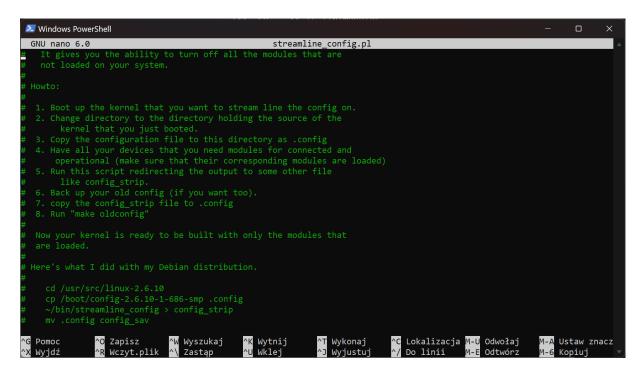
Jak widać wpis został poprawnie dodany do LILO, i nowa wersja kernela jest możliwa do wyboru.

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

### 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



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:

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.

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
          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
         arch/x86/boot/compressed/error.o
 RELOCS arch/x86/boot/compressed/vmlinux.relocs
         arch/x86/boot/compressed/early_serial_console.o
         arch/x86/boot/compressed/kaslr.o
         arch/x86/boot/compressed/acpi.o
         arch/x86/boot/compressed/misc.o
         arch/x86/boot/compressed/vmlinux.bin.lzma
 MKPIGGY arch/x86/boot/compressed/piggy.S
         arch/x86/boot/compressed/piggy.o
LD arch/x86/boot/compressed/vmlinux ZOFFSET arch/x86/boot/zoffset.h
 OBJCOPY arch/x86/boot/vmlinux.bin
         arch/x86/boot/header.o
         arch/x86/boot/setup.elf
 OBJCOPY arch/x86/boot/setup.bin
 BUILD arch/x86/boot/bzImage
(ernel: arch/x86/boot/bzImage is ready (#2)
oot@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
          sound/soc/sof/xtensa/snd-sof-xtensa-dsp.ko
          sound/synth/emux/snd-emux-synth.ko
          sound/synth/snd-util-mem.ko
          sound/usb/bcd2000/snd-bcd2000.ko
          sound/usb/6fire/snd-usb-6fire.ko
          sound/usb/caiaq/snd-usb-caiaq.ko
sound/usb/line6/snd-usb-line6.ko
LD
LD
          sound/usb/hiface/snd-usb-hiface.ko
          sound/usb/line6/snd-usb-pod.ko
          sound/usb/line6/snd-usb-podhd.ko
sound/usb/line6/snd-usb-toneport.ko
sound/usb/line6/snd-usb-variax.ko
          sound/usb/misc/snd-ua101.ko
LD
          sound/usb/snd-usb-audio.ko
LD
          sound/usb/usx2y/snd-usb-usx2y.ko
          sound/usb/snd-usbmidi-lib.ko
          sound/usb/usx2y/snd-usb-us122l.ko
          sound/virtio/virtio_snd.ko
          sound/x86/snd-hdmi-lpe-audio.ko
ΙD
LD [M] virt/lib/irqbypass.ko
oot@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/ dzięku użyciu komendy cd, następnie usunąłem starą tablice symboli za pomocą komendy rm, i za pomocą komendy ln utworzyłem link symboliczny.

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

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.qz

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

```
Windows PowerShell
                                                                      /etc/lilo.conf
                                                                                                                                              Zmieniony
  VESA framebuffer console @ 640x480x64k
 VESA framebuffer console @ 640x480x256
  End LILO global section
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"
                                                                                                     Lokalizacja <mark>M-U</mark> Odwołaj
Do linii M-E Odtwórz
                                                             Wytnij
Wklej
                                                                              ^T Wykonaj
^J Wyjustuj
                                                                                                                                         M-A Ustaw znacz
M-6 Kopiuj
                                          Wyszukaj
   Pomoc
                       Zapisz
   Wyjdź
                       Wczyt.plik
                                                                                                                                              Kopiuj
```

Oraz zaktualizowanie lilo poleceniem lilo

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:

```
Starting haveged entropy daenon: /sbin/haveged
Starting the network interfaces...

Starting the network interfaces...

Starting the network interfaces...

Subject of the starting of the starting the starting the starting for carrier of the subject of the starting for carrier of the subject of the subject of the starting for carrier of the subject of the s
```

#### 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 robić. Mimo tego, że metoda nowa wydaję się prostsza to kompilacją modułów i samego jądra trwała tą metoda 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.