

Kompilacja kernela

Mikołaj Cekut

16 czerwca 2022

1 Ogólna konfiguracja

Na początku, niezależnie od metody komplikacji jądra należy pobrać pliki źródłowe wersji jądra, którą chcemy zainstalować. Możemy skorzystać ze strony: <https://www.kernel.org/>. Wykorzystamy wersję: **5.18.2**. Link do pobrania tej wersji możemy znaleźć pod adresem:

- <https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.2.tar.xz>

Aby pobrać plik za pomocą konsoli Linuxa korzystamy z polecenia:

- `cd /usr/src && wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.2.tar.xz`

Dzięki temu poleceniu przeniesiemy się do folderu **/usr/src** i pobierzemy plik. Plik jest skompresowany, więc musimy go rozpakować za pomocą polecenia **tar -xvf**

```
Starting the network interfaces...  
eth0: polling for DHCP server  
dhcpcd-9.4.1 starting  
DUID 00:04:a9:85:88:51:1d:cc:84:49:bd:40:41:eb:26:ef:8b:c7  
eth0: waiting for carrier  
eth0: carrier acquired  
eth0: IAID 27:47:47:01  
eth0: rebinding lease of 10.0.2.15  
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 842  
Starting system message bus: /usr/bin/dbus-uuidgen --ensure ; /usr/bin/dbus-daemon --system  
Starting elogind: /lib/systemd/system/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 5.15.19-smp i686 (tty1)  
slack login: root  
Password:  
Last login: Tue Jun  7 17:38:30 on tty1  
Linux 5.15.19-smp.  
root@slack:~# cd /usr/src && wget https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.18.2.tar.xz
```

Rysunek 1: Pobieranie paczki

```
linux-5.18.2.tar.xz      100%[=====] 123.78M  41.3MB/s   in 3.0s  
2022-06-07 17:46:08 (41.3 MB/s) - linux-5.18.2.tar.xz - saved [129796020/129796020]  
root@slack:/usr/src# tar -xvf linux-5.18.2.tar.xz
```

Rysunek 2: Rozpakowanie paczki

2 Metoda stara

Aktualną konfigurację kernela trzeba przekopiować do pliku **.config** w folderze **/usr/src/linux-5.18.2**

```
linux-5.18.2/ linux-5.18.2.tar.xz
root@slack:/usr/src# cd linux-5.18.2
root@slack:/usr/src/linux-5.18.2# zcat /proc/config.gz > .config
```

Rysunek 3: Kopiowanie starego configa do nowego

Następnie należy użyć polecenia **make localmodconfig** aby wygenerować plik. Polecenia wyświetliło nam, że korzysta z pliku **.config** oraz pozwala nam na konfigurację poszczególnych modułów jądra. Używam domyślnej konfiguracji modułów - aby to zrobić, gdy otrzymamy zapytanie o moduł wciskamy **ENTER**.

```
root@slack:/usr/src# cd linux-5.18.2
root@slack:/usr/src/linux-5.18.2# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.2# ls -l .config
-rw-r--r-- 1 root root 237825 Jun  7 17:50 .config
root@slack:/usr/src/linux-5.18.2# make localmodconfig
```

Rysunek 4: Make localmodconfig

```
HOSTCC scripts/basic/fixed
HOSTCC scripts/kconfig/conf.o
HOSTCC scripts/kconfig/confdata.o
HOSTCC scripts/kconfig/expr.o
LEX scripts/kconfig/lexer.lex.c
YACC scripts/kconfig/parser.tab.[ch]
HOSTCC scripts/kconfig/lexer.lex.o
HOSTCC scripts/kconfig/menu.o
HOSTCC scripts/kconfig/parser.tab.o
HOSTCC scripts/kconfig/preprocess.o
HOSTCC scripts/kconfig/symbol.o
HOSTCC scripts/kconfig/util.o
HOSTLD scripts/kconfig/conf
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)
```

Rysunek 5: Output make localmodconfig

Kolejnym krokiem jest komplikacja bazowego jądra za pomocą komendy **make -j4 bzImage** oraz modułów jądra za pomocą komendy **make -j4 modules**.

```
[root@slack:~# cd /usr/src/linux-5.18.2
root@slack:/usr/src/linux-5.18.2# make -j4 bzImage
```

Rysunek 6: Make -j4 bzImage

```
CC      arch/x86/boot/video-bios.o
HOSTCC arch/x86/boot/tools/build
CPUSTR arch/x86/boot/cpustr.h
CC      arch/x86/boot/compressed/error.o
CC      arch/x86/boot/cpu.o
OBJCOPY arch/x86/boot/compressed/vmlinuz.bin
RELOCS arch/x86/boot/compressed/vmlinuz.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/vmlinuz.bin.lzma
MRPIGGY arch/x86/boot/compressed/piggy.S
AS     arch/x86/boot/compressed/piggy.o
LD     arch/x86/boot/compressed/vmlinuz
ZOFFSET arch/x86/boot/zoffset.h
OBJCOPY arch/x86/boot/vmlinuz.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-5.18.2# _
```

Rysunek 7: Wynik kompilacji bazowej

```
[root@slack:/usr/src/linux-5.18.2# make -j4 modules
```

Rysunek 8: Make -j4 modules

```

LD [M] drivers/powercap/intel_rapl_common.ko
LD [M] drivers/powercap/intel_rapl_msr.ko
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/l1c/l1c.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-intel18x0.ko
LD [M] sound/soundcore.ko
root@slack:/usr/src/linux-5.18.2# _

```

Rysunek 9: Wynik komplikacji modułów

Po skompilowaniu modułów należy je zainstalować komendą **make modules_install**.

```

INSTALL /lib/modules/5.18.2-smp/kernel/drivers/powercap/intel_rapl_msr.ko
INSTALL /lib/modules/5.18.2-smp/kernel/drivers/video/fbdev/core/fb_sys_fops.ko
INSTALL /lib/modules/5.18.2-smp/kernel/drivers/video/fbdev/core/syscopyarea.ko
INSTALL /lib/modules/5.18.2-smp/kernel/drivers/video/fbdev/core/sysfillrect.ko
INSTALL /lib/modules/5.18.2-smp/kernel/drivers/video/fbdev/core/sysimgblt.ko
INSTALL /lib/modules/5.18.2-smp/kernel/drivers/virt/vboxguest/vboxguest.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/802/garp.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/802/mrp.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/802/p8022.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/802/psnap.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/802/stp.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/8021q/8021q.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/ipv6/ipv6.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/l1c/l1c.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/rfkill/rfkill.ko
INSTALL /lib/modules/5.18.2-smp/kernel/net/wireless/cfg80211.ko
INSTALL /lib/modules/5.18.2-smp/kernel/sound/ac97_bus.ko
INSTALL /lib/modules/5.18.2-smp/kernel/sound/core/snd-pcm.ko
INSTALL /lib/modules/5.18.2-smp/kernel/sound/core/snd-timer.ko
INSTALL /lib/modules/5.18.2-smp/kernel/sound/core/snd.ko
INSTALL /lib/modules/5.18.2-smp/kernel/sound/pci/ac97/snd-ac97-codec.ko
INSTALL /lib/modules/5.18.2-smp/kernel/sound/pci/snd-intel18x0.ko
INSTAL /lib/modules/5.18.2-smp/kernel/sound/soundcore.ko
DEPMOD /lib/modules/5.18.2-smp/
root@slack:/usr/src/linux-5.18.2# _

```

Rysunek 10: Wynik instalacji modułów

Kolejnym krokiem jest skopiowanie pozostałych plików potrzebnych do uruchomienia jądra.

```
root@slack:/usr/src/linux-5.18.2# cp arch/x86/boot/bzImage /boot/vmlinuz-5.18.2-smp
root@slack:/usr/src/linux-5.18.2# cp System.map /boot/System.map-5.18.2-smp
root@slack:/usr/src/linux-5.18.2# cp .config /boot/config-5.18.2-smp
root@slack:/usr/src/linux-5.18.2#
```

Rysunek 11: Kopiowanie potrzebnych plików do katalogu /boot

Po zainstalowaniu modułów i skopiowania plików nastepnym krokiem jest wygenerowaniu ramdisku. Za pomocą komendy **mkinitrd_command_generator.sh -k 5.18.2-smp** otrzymamy polecenie, które musimy wykorzystać do wygenerowania ramdisku.

```
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.18.2-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 5.18.2-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot#
```

Rysunek 12: mkinitrd_command_generator.sh -k 5.18.2-smp

```
root@slack:/boot# mkinitrd -c 5.18.2-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
OK: /lib/modules/5.15.19-smp/kernel/fs/jbd2/jbd2.ko added.
OK: /lib/modules/5.15.19-smp/kernel/fs/mbcache.ko added.
OK: /lib/modules/5.15.19-smp/kernel/fs/ext4/ext4.ko added.
51232 blocks
/boot/initrd.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

Rysunek 13: Użycie wygenerowanej komendy

Ostatnim krokiem jaki został przed możliwością uruchomienia nowego kernela jest dodanie wpisu o nim do lilo.conf oraz użycie komendy **lilo**

```
# VESA framebuffer console @ 1024x768x32k
#vga=790
# VESA framebuffer console @ 1024x768x256
#vga=773
# 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-5.18.2-smp
root = /dev/sda1
initrd = /boot/initrd-5.18.2-smp.gz
label = "Chad Linux #1"
read-only
# Linux bootable partition config ends
-- INSERT --
```

68,26 Bot

Rysunek 14: Konfiguracja w pliku lilo.conf

```
"/etc/lilo.conf" 70L, 2077B written
root@slack:/boot# lilo
Warning: LBA32 addressing assumed
Added Chad_Linux_#1 + *
One warning was issued.
root@slack:/boot# _
```

Rysunek 15: Wywołanie komendy lilo

Teraz możemy uruchomić maszynę ponownie. Jak widać nadpisałem po przednią konfigurację kernela swoją własną, nową, tak więc widać tylko jeden wpis - nowo wgraneego kernela.



Rysunek 16: Ekran startowy

Uruchamiamy więc nasz kernel. Jak widać udało nam się zalogować do systemu, tak więc komplikacja okazała się sukcesem, jednak pragnę zauważyc, iż system uruchamiał się bardzo długo (około 3-4 minut).

```
Welcome to Linux 5.18.2-smp i686 (tty1)
slack login: root
Password:
Last login: Tue Jun  7 17:56:34 on tty1
Linux 5.18.2-smp.
root@slack:~# _
```

Rysunek 17: Logowanie udane

3 Metoda nowa

Korzystając z metody nowej wykorzystujemy skrypt `streamline_config.pl` zlokalizowany w katalogu `./scripts/kconfig/`.

```

# you only want the modules you use, then this program
# is perfect for you.

# 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 streamline 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
# mv config_strip .config
# make oldconfig

#
use warnings;
use strict;
use Getopt::Long;

-- INSERT --

```

27,57 1%

Rysunek 18: Skrypt streamline_config.pl otwarty za pomocą vim zawiera w sobie tutorial

```

root@slack:/usr/src/linux-5.18.2# cp /boot/config .config
root@slack:/usr/src/linux-5.18.2# ./scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'
root@slack:/usr/src/linux-5.18.2# _

```

Rysunek 19: Wykorzystanie skryptu

Następnie używamy komendy **make oldconfig** aby stworzyć konfigurację.

```

Self test for hardware accelerated raid6 recovery (ASYNC_RAID6_TEST) [M/n/y/?] m
Test functions located in the hexdump module at runtime (TEST_HEXDUMP) [M/n/y/?] m
Test string functions at runtime (STRING_SELFTEST) [M/n/y/?] n
Test functions located in the string_helpers module at runtime (TEST_STRING_HELPERS) [N/m/y/?] n
Test strscpy*() family of functions at runtime (TEST_STRSCPY) [N/m/y/?] n
Test kstrto*() family of functions at runtime (TEST_KSTRTOX) [N/m/y/?] n
Test printf() family of functions at runtime (TEST_PRINTF) [N/m/y/?] n
Test scanf() family of functions at runtime (TEST_SCANF) [N/m/y/?] n
Test bitmap_*() family of functions at runtime (TEST_BITMAP) [N/m/y/?] n
Test functions located in the uid module at runtime (TEST_UUID) [N/m/y/?] n
Test the XArray code at runtime (TEST_XARRAY) [N/m/y/?] n
Perform selftest on resizable hash table (TEST_RHASHTABLE) [N/m/y/?] n
Perform selftest on siphash functions (TEST_SIPHASH) [N/m/y/?] (NEW)
Perform selftest on IDA functions (TEST_IDA) [N/m/y/?] n
Perform selftest on priority array manager (TEST_PARMAN) [N/m/?] n
Test module loading with 'hello world' module (TEST_LKM) [M/n/?] m
Test module for compilation of bitops operations (TEST_BITOPS) [N/m/?] n
Test module for stress/performance analysis of umalloc allocator (TEST_UMALLOC) [N/m/?] n
Test user/kernel boundary protections (TEST_USER_COPY) [M/n/?] m
Test BPF filter functionality (TEST_BPF) [M/n/?] m
Test blackhole netdev functionality (TEST_BLACKHOLE_DEV) [M/n/?] n
Test find_bit functions (FIND_BIT_BENCHMARK) [N/m/y/?] n
Test firmware loading via userspace interface (TEST_FIRMWARE) [M/n/y/?] m
sysctl test driver (TEST_SYSCTL) [N/m/y/?] n
udelay test driver (TEST_UDELAY) [M/n/y/?] m
Test static keys (TEST_STATIC_KEYS) [N/m/?] n
kmod stress tester (TEST_KMOD) [N/m/?] n
Test nemcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] n
Perform selftest on object aggregation manager (TEST_OBJAGG) [N/m/?] 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-5.18.2# _

```

Rysunek 20: make oldconfig

Następne etapy (make bzImage, make modules itd.) zostały opisane w poprzedniej metodzie, cały ten proces jest dokładnie taki sam jak poprzednio, więc przejdziemy do wyniku komplikacji. Jak widać na ekranie startowym pojawił się nowo skompilowany kernel. Niestety system nie chce się uruchomić, pojawia się błąd, po czym maszyna się "zamraża". Po próbie uruchomienia ponownie maszyny problem się powtarza.



Rysunek 21: Ekran startowy po wykonaniu drugiej metody

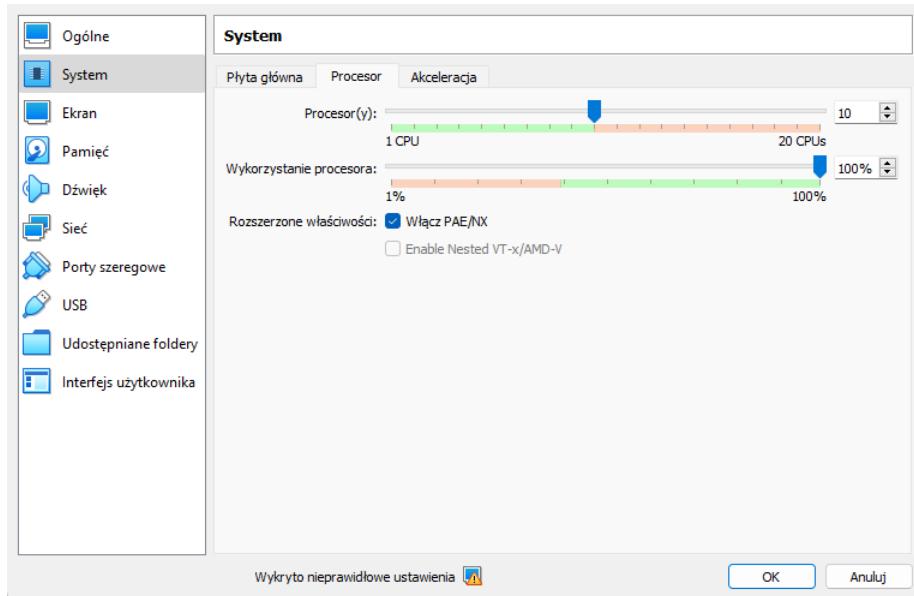
```
[  GiB)
[ 279.156634] sd 0:0:0:0: [sdal] Write Protect is off
[ 279.193922] sd 0:0:0:0: [sdal] Write cache: enabled, read cache: enabled, does
n't support DPO or FUA
[ 280.400761] sda: sda1 sda2 sda3
[ 281.162305] sd 0:0:0:0: [sdal] Attached SCSI disk
[ 281.546770] Freeing unused kernel image (initmem) memory: 1048K
[ 281.954707] Write protecting kernel text and read-only data: 22136k
[ 282.032606] NX-protecting the kernel data: 10148k
[ 282.129952] rodata_test: all tests were successful
[ 282.167115] Run /init as init process
[ 308.285178] clocksource: timekeeping watchdog on CPU6: Marking clocksource 't
sc-early' as unstable because the skew is too large:
[ 308.373090] clocksource: 'acpi_pm' wd_nsec: 2075442269
wd_now: 78c476 wd_last: 76863 mask: ffffff
[ 308.460197] clocksource: 'tsc-early' cs_nsec: 168179036
074 cs_now: cfe59b74c cs_last: 9a86f4010 mask: fffffffffff
[ 308.570698] clocksource: 'tsc-early' is current clockso
urce.
[ 308.646888] tsc: Marking TSC unstable due to clocksource watchdog
[ 308.697957] TSC found unstable after boot, most likely due to broken BIOS. Us
e 'tsc=unstable'.
[ 308.778063] sched_clock: Marking unstable (308623927644, 74024465)<-(31688471
0808, -8186758354)
```

Rysunek 22: Błąd uruchomienia drugiej metody

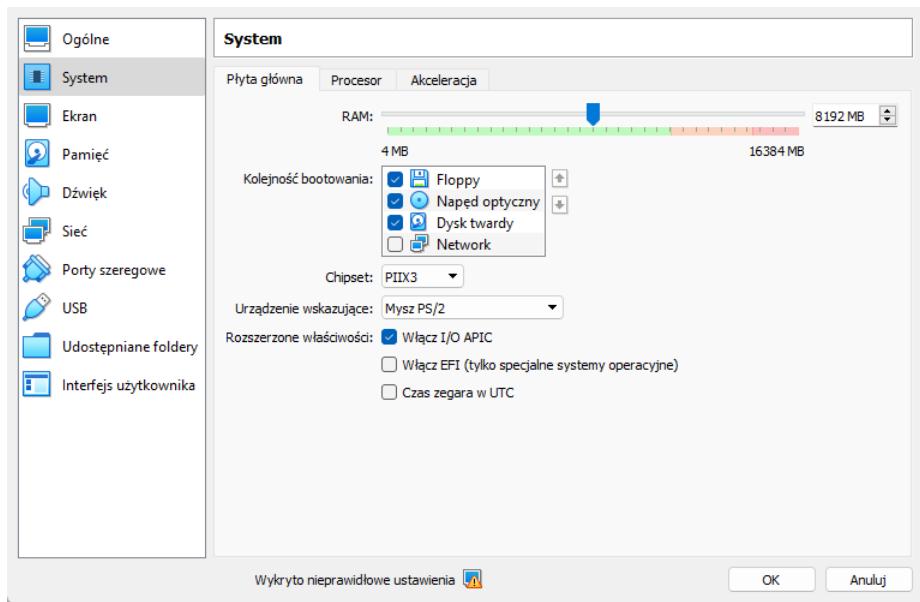
4 Podsumowanie

Porównując obie metody, co do samego procesu tworzenia nie ma większych różnic, może oprócz tego, że nową metodą moduły komplikują się wolniej. Jednak co do wyniku komplikacji nie miałem możliwości ich porównać, ponieważ po komplikacji metodą starą linux niestety nie uruchomił się.

Dodam jeszcze, że obie konfiguracje wykonywałem korzystając z tej konfiguracji maszyny wirtualnej.



Rysunek 23: Konfiguracja maszyny #1



Rysunek 24: Konfiguracja maszyny #2