Kompilacja jądra Linux - Sebastian Sacharczuk

System:

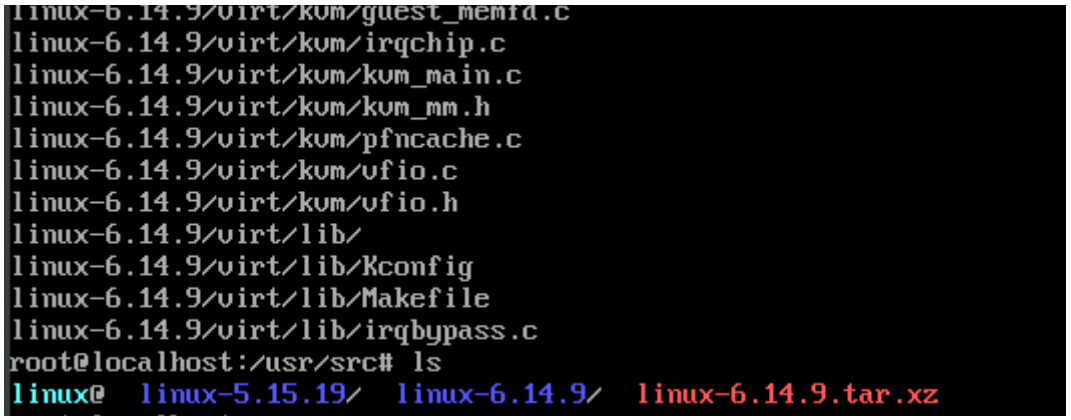
- Rdzenie CPU: 4
- RAM: 8192 MB
- Kernel bazowy: 5.15.4

1. Pobranie najnowszego stabilnego jądra ze strony https://cdn.kernel.org/
(stan na dzień 29-05-2025 - stablina wersja 6.14.9)

```
cd /usr/src
wget https://cdn.kernel.org/pub/linux/kernel/v6.x/linux-6.14.9.tar.xz
```

2. Rozpakowanie jądra

```
tar -xvpf linux-6.14.9.tar.xz
```



3. Konfiguracja jądra

3.1. Przejście do katalogu z rozpakowanym jądrem

```
cd linux-6.14.9
```

3.2. Skopiowanie aktualnie używanej konfiguracji jądra

```
zcat /proc/config.gz > .config
```

3.3. Tworzenie pliku konfiguracyjnego (dwie metody)

3.3.1. Metoda stara

- Utworzenie pliku konfiguracyjnego jądra tylko z tymi modułami, które używamy (stara metoda)

```
make localmodconfig
```

(na każde pytanie odnośnie dołączenia modułów, wybierałem opcję domyślne <ENTER>)

```
Test module for compilation of bitops operations (TEST_BITOPS) [N/m/y/?] n
Test module for stress/performance analysis of vmalloc allocator (TEST_VMALLOC) [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
module kallsyms find_symbol() test (TEST_KALLSYMS) [N/m/?] (NEW)
Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] n
Test heap/page initialization (TEST_MEMINIT) [N/m/y/?] n
Test HMM (Heterogeneous Memory Management) (TEST_HMM) [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
Test module for correctness and stress of objpool (TEST_OBJPOOL) [N/m/?] (NEW)
#
# configuration written to .config
#
```

3.3.2. Metoda nowa

- Sprawdzenie skryptu do tworzenia pliku .config tylko z tymi modułami, które używamy (nowa metoda)
  nano scripts/kconfig/streamline_config.pl

```
#!/usr/bin/env perl
# SPDX-License-Identifier: GPL-2.0
#
# Copyright 2005-2009 - Steven Rostedt
#
#  It's simple enough to figure out how this works.
#  If not, then you can ask me at stripconfig@goodmis.org
#
# What it does?
#
#   If you have installed a Linux kernel from a distribution
#   that turns on way too many modules than you need, and
#   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 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
#    mv config_strip .config
#    make oldconfig
#
```

- Tworzenie konfiguracji z tymi modułami, które są rzeczywiście używane w systemie.
  scripts/kconfig/streamline_config.pl > config_strip

```
root@localhost:/usr/src/linux-6.14.9# scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'
crct10dif_pclmul config not found!!
crc32_pclmul config not found!!
module fb_sys_fops did not have configs CONFIG_FB_SYSMEM_FOPS
```

- Kopia zapasowa
  mv .config config_save
- Ustawienie nowej konfiguracji do właściwego pliku
  mv config_strip .config
- Uzupełnienie konfiguracji
  make oldconfig

```
  Test the XArray code at runtime (TEST_XARRAY) [N/m/y/?] n
  Test the Maple Tree code at runtime or module load (TEST_MAPLE_TREE) [N/m/y/?] (NEW)
  Perform selftest on resizable hash table (TEST_RHASHTABLE) [N/m/y/?] n
  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/y/?] n
  Test module for stress/performance analysis of vmalloc allocator (TEST_VMALLOC) [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
  module kallsyms find_symbol() test (TEST_KALLSYMS) [N/m/?] (NEW)
  Test memcat_p() helper function (TEST_MEMCAT_P) [N/m/y/?] n
  Test heap/page initialization (TEST_MEMINIT) [N/m/y/?] n
  Test HMM (Heterogeneous Memory Management) (TEST_HMM) [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
  Test module for correctness and stress of objpool (TEST_OBJPOOL) [N/m/?] (NEW)
#
# configuration written to .config
#
```

4. Sprawdzenie aktualnie załadowanych modułów
   lsmod

```
Module                  Size  Used by
vboxguest              49152  6
cfg80211             1347584  0
8021q                  40960  0
garp                   16384  1 8021q
stp                    12288  1 garp
mrp                    20480  1 8021q
llc                    16384  2 stp,garp
rfkill                 40960  3 cfg80211
efivarfs               28672  1
ipv6                  716800  36
vmwgfx                458752  2
snd_intel8x0           45056  2
drm_client_lib         12288  1 vmwgfx
drm_ttm_helper         16384  2 vmwgfx
snd_ac97_codec        196608  1 snd_intel8x0
ttm                   102400  2 vmwgfx,drm_ttm_helper
ac97_bus               12288  1 snd_ac97_codec
snd_pcm               176128  2 snd_intel8x0,snd_ac97_codec
drm_kms_helper        225280  3 vmwgfx,drm_ttm_helper,drm_client_lib
snd_timer              45056  1 snd_pcm
intel_rapl_msr         16384  0
joydev                 20480  0
psmouse               192512  0
drm                   716800  8 vmwgfx,drm_kms_helper,drm_ttm_helper,drm_client_lib,ttm
i2c_piix4              24576  0
pcnet32                53248  0
ohci_pci               12288  0
snd                   131072  8 snd_intel8x0,snd_timer,snd_ac97_codec,snd_pcm
i2c_smbus              12288  1 i2c_piix4
intel_agp              24576  0
video                  69632  0
intel_rapl_common      45056  1 intel_rapl_msr
ohci_hcd               45056  1 ohci_pci
intel_gtt              24576  1 intel_agp
ehci_pci               12288  0
e1000                 151552  0
agpgart                45056  3 intel_agp,intel_gtt,ttm
ehci_hcd               69632  1 ehci_pci
ghash_clmulni_intel    16384  0
```

4. Kompilacja obrazu jądra (parametr j == ilość CPUs)

time make -j4 bzImage

```
  CC      arch/x86/boot/compressed/efi.o
  AS      arch/x86/boot/compressed/efi_mixed.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)

real    12m29,032s
user    31m59,451s
sys     10m54,646s
```

5. Zbudowanie modułów jądra (parametr j == ilość CPUs)

time make -j5 modules

```
  LD [M]  sound/core/snd-pcm.ko
  LD [M]  sound/pci/snd-intel8x0.ko
  LD [M]  sound/pci/ac97/snd-ac97-codec.ko
  LD [M]  sound/ac97_bus.ko
  LD [M]  net/802/psnap.ko
  LD [M]  net/802/p8022.ko
  LD [M]  net/802/stp.ko
  LD [M]  net/802/garp.ko
  LD [M]  net/802/mrp.ko
  LD [M]  net/ipv6/ipv6.ko
  LD [M]  net/8021q/8021q.ko
  LD [M]  net/wireless/cfg80211.ko
  LD [M]  net/llc/llc.ko
  LD [M]  net/rfkill/rfkill.ko

real    1m8,476s
user    3m3,963s
sys     1m2,238s
```

6. Instalacja modułów

time make -j5 modules_install

```
  INSTALL /lib/modules/6.14.9/kernel/sound/core/snd-pcm.ko
  INSTALL /lib/modules/6.14.9/kernel/sound/pci/snd-intel8x0.ko
  INSTALL /lib/modules/6.14.9/kernel/sound/pci/ac97/snd-ac97-codec.ko
  INSTALL /lib/modules/6.14.9/kernel/sound/ac97_bus.ko
  INSTALL /lib/modules/6.14.9/kernel/net/802/p8022.ko
  INSTALL /lib/modules/6.14.9/kernel/net/802/psnap.ko
  INSTALL /lib/modules/6.14.9/kernel/net/802/stp.ko
  INSTALL /lib/modules/6.14.9/kernel/net/802/garp.ko
  INSTALL /lib/modules/6.14.9/kernel/net/802/mrp.ko
  INSTALL /lib/modules/6.14.9/kernel/net/ipv6/ipv6.ko
  INSTALL /lib/modules/6.14.9/kernel/net/8021q/8021q.ko
  INSTALL /lib/modules/6.14.9/kernel/net/wireless/cfg80211.ko
  INSTALL /lib/modules/6.14.9/kernel/net/llc/llc.ko
  INSTALL /lib/modules/6.14.9/kernel/net/rfkill/rfkill.ko
  DEPMOD  /lib/modules/6.14.9

real    0m1,282s
user    0m0,960s
sys     0m1,005s
```

7. Sprawdzenie zainstalowanych modułów

ls /lib/modules/6.14.9/

```
root@localhost:/usr/src/linux-6.14.9# ls /lib/modules/6.14.9/
build@         modules.alias.bin          modules.builtin.bin      modules.dep.bin  modules.softdep
kernel/        modules.builtin            modules.builtin.modinfo  modules.devname  modules.symbols
modules.alias  modules.builtin.alias.bin  modules.dep              modules.order    modules.symbols.bin
```

8. Przekopiowanie plików kernela do systemu

8.1. Przekopiowanie obrazu jądra do katalogu /boot

cp arch/x86_64/boot/bzImage /boot/vmlinuz-custom-6.14.9

8.2. Przekopiowanie tablicy symboli używanych przez kernel do katalogu /boot

cp System.map /boot/System.map-custom-6.14.9

8.3. Przekopiowanie pliku konfiguracyjnego kernela

cp .config /boot/config-custom-6.14.9

9. Utworzenie linku symbolicznego w systemie dla tablicy symboli kernela

9.1. Przejście do katalogu /boot

cd /boot

9.2. Usunięcie starej tablicy symboli

rm System.map

9.3. Utworzenie linku symbolicznego

ln -s System.map-custom-6.14.9 System.map

```
root@localhost:/boot# ln -s System.map-custom-6.14.9 System.map
root@localhost:/boot# ls
README.initrd@              config-custom-6.14.9       grub/        onlyblue.dat          vmlinuz-generic@
System.map@                 config-generic-5.15.19.x64 initrd-tree/ slack.bmp             vmlinuz-generic-5.15.19
System.map-custom-6.14.9    config-huge-5.15.19.x64    initrd.gz    tuxlogo.bmp           vmlinuz-huge@
System.map-generic-5.15.19  efi/                       inside.bmp   tuxlogo.dat           vmlinuz-huge-5.15.19
System.map-huge-5.15.19     elilo-ia32.efi*            inside.dat   vmlinuz@
config@                     elilo-x86_64.efi*          onlyblue.bmp vmlinuz-custom-6.14.9
```

10. Utworzenie dysku RAM

10.1 Wywołanie skryptu generującego komendę do wykonania

/usr/share/mkinitrd/mkinitrd_command_generator.sh -k 6.14.9

```
root@localhost:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 6.14.9
#
# 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 6.14.9 -f ext4 -r /dev/sda4 -m ext4 -u -o /boot/initrd.gz
```

10.2. Wywołanie komendy tworzącej dysk RAM

mkinitrd -c -k 6.14.9 -f ext4 -r /dev/sda4 -m ext4 -u -o /boot/initrd-custom-6.14.9.gz

```
root@localhost:/boot# mkinitrd -c -k 6.14.9 -f ext4 -r /dev/sda4 -m ext4 -u -o /boot/initrd-custom-6.14.9.gz
51373 bloki
/boot/initrd-custom-6.14.9.gz created.
Be sure to run lilo again if you use it.
```

11. Konfiguracja GRUB

11.1 Generowanie pliku konfiguracyjnego GRUB

grub-mkconfig -o /boot/grub/grub.cfg

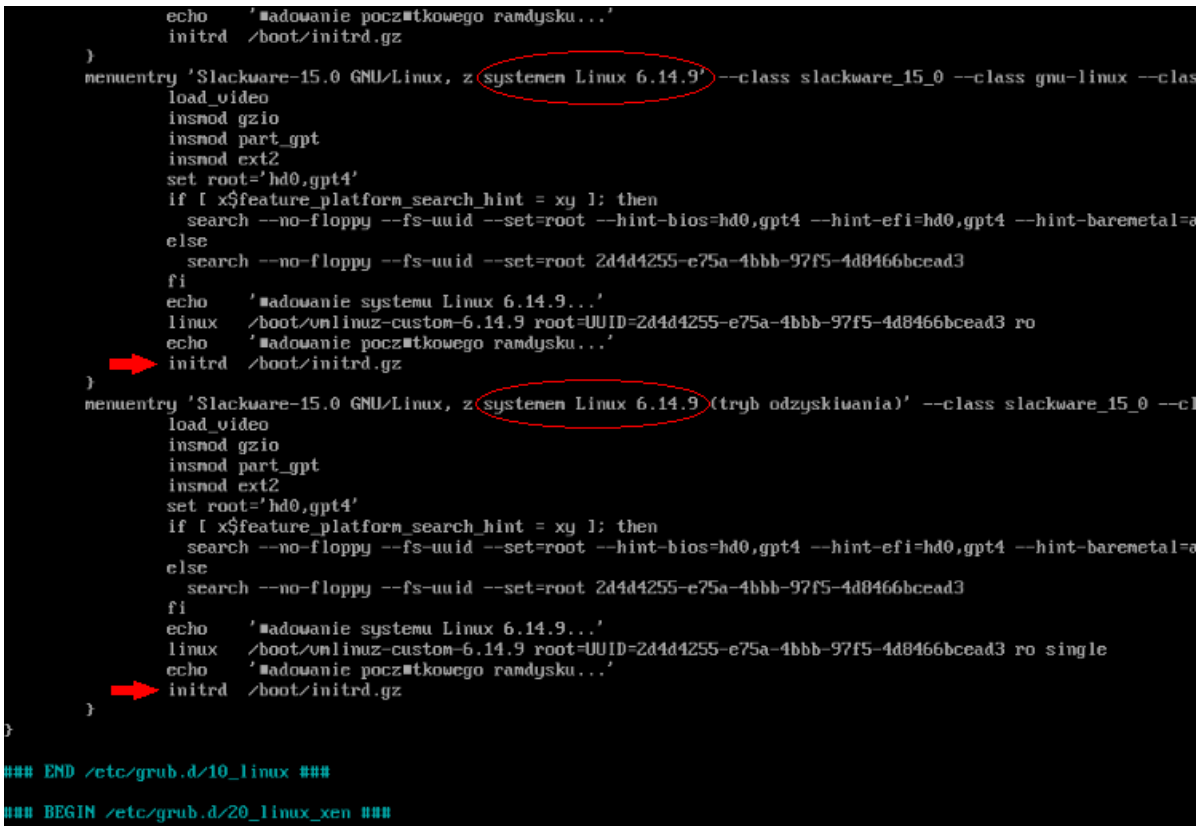
```
root@localhost:/boot# grub-mkconfig -o /boot/grub/grub.cfg
Generowanie pliku konfiguracyjnego gruba...
Znaleziono obraz Linuksa: /boot/vmlinuz-huge-5.15.19
Znaleziono obraz initrd: /boot/initrd.gz
Znaleziono obraz Linuksa: /boot/vmlinuz-huge
Znaleziono obraz initrd: /boot/initrd.gz
Znaleziono obraz Linuksa: /boot/vmlinuz-generic-5.15.19
Znaleziono obraz initrd: /boot/initrd.gz
Znaleziono obraz Linuksa: /boot/vmlinuz-generic
Znaleziono obraz initrd: /boot/initrd.gz
Znaleziono obraz Linuksa: /boot/vmlinuz-custom-6.14.9
Znaleziono obraz initrd: /boot/initrd.gz
Uwaga: os-prober nie zostanie uruchomiony w celu wykrycia innych uruchamialnych partycji.
Systemy na nich nie zostan■ dodane do konfiguracji rozruchowej GRUB-a.
Prosz■ sprawdzi■ dokumentacj■ dotycz■c■ GRUB_DISABLE_OS_PROBER.
Dodawanie wpisu menu rozruchowego dla ustawie■ firmware'u UEFI...
gotowe
```

11.2. Należy wprowadzić zmiany w konfiguracji GRUB

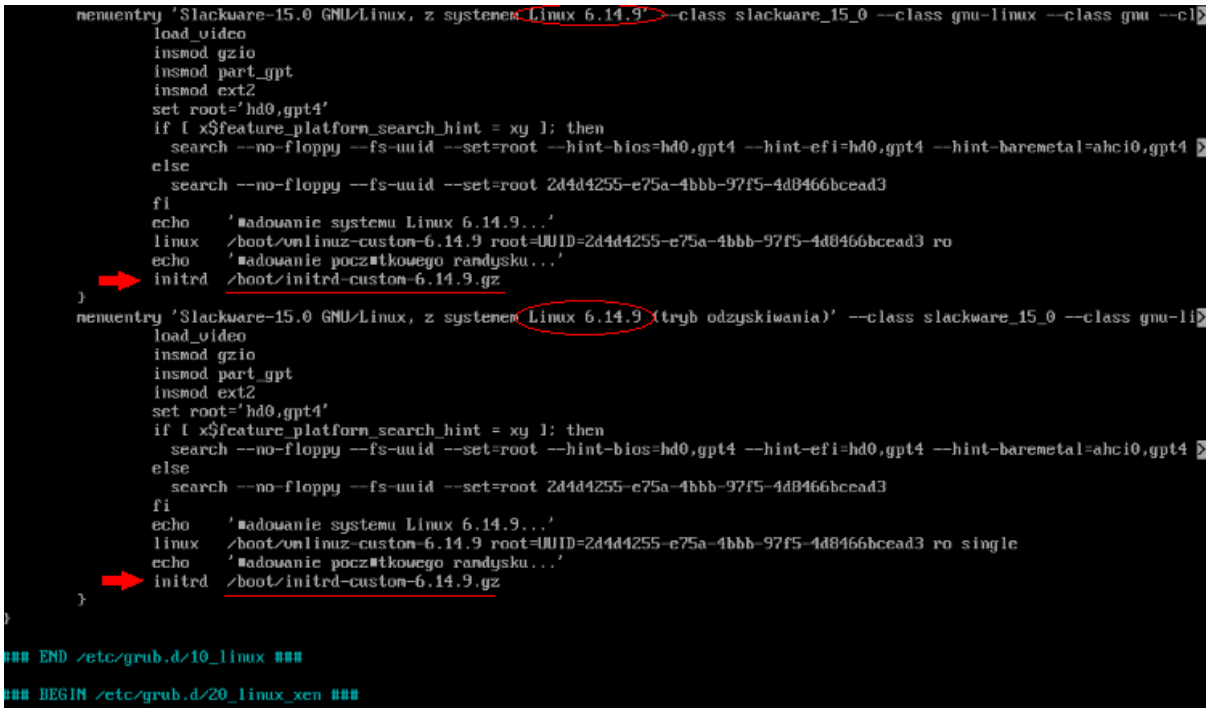
nano /boot/grub/grub.cfg

W polach **menuentry** z naszym nowym kernelem (Linux 6.14.9) należy zmienić wartość dla **initrd** na “/boot/initrd-custom-6.14.9.gz”**.** [Prawdopodobnie 2 wystąpienia menuentry]

Przed zmianami:



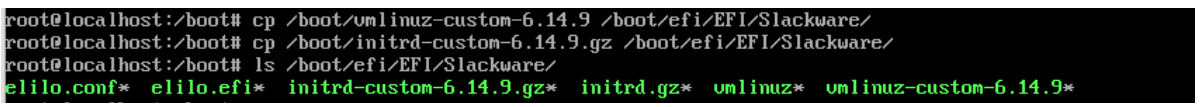
Po zmianach:



12. Skopiowanie obrazu jądra oraz ramdisku na partycję EFI

cp /boot/vmlinuz-custom-6.14.9 /boot/efi/EFI/Slackware/

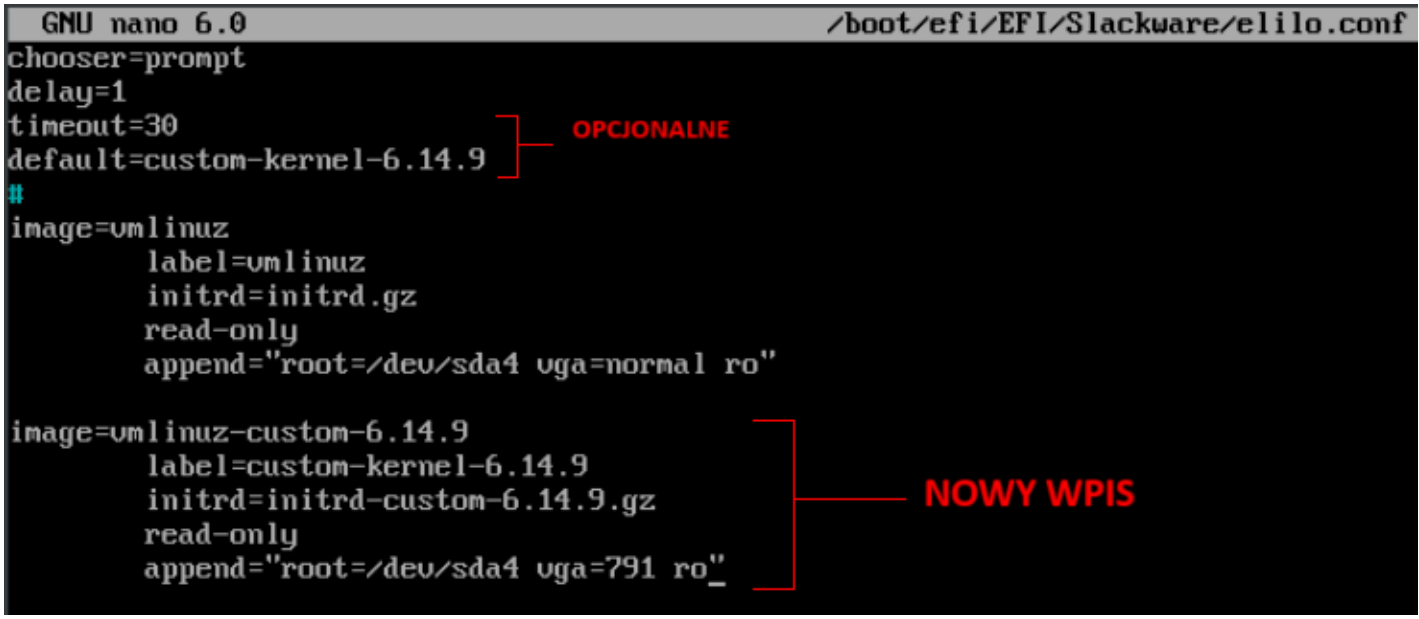
cp /boot/initrd-custom-6.14.9.gz /boot/efi/EFI/Slackware/



13. Dodanie nowego wpisu do konfiguracji bootloadera ELILO

nano /boot/efi/EFI/Slackware/elilo.conf

Po zmianach:



14. MIGAWKA, REBOOT I TESTUJEMY

WNIOSKI

W przypadku obu metod:

- myszka przewodowa – działa
- internet – dziala
- dźwięk – nie działa (pewnie wina ustawień wirtualnej maszyny)

Z mojego researchu wynika że “stara metoda” tworzenia konfiguracji jądra (make localmodconfig) załącza do pliku konfiguracyjnego moduły które są aktualnie załadowane. Istnieje ryzyko że nie będą załadowane moduły odpowiadające za np. WI-FI, Bluetooth, drukarki.

Natomiat “nowa metoda” wykorzystująca skrypt streamline_config.pl jest bardziej zaawansowana. Skrypt przeszukuje cały system nie omijając sterowników wbudowanych w jądro.

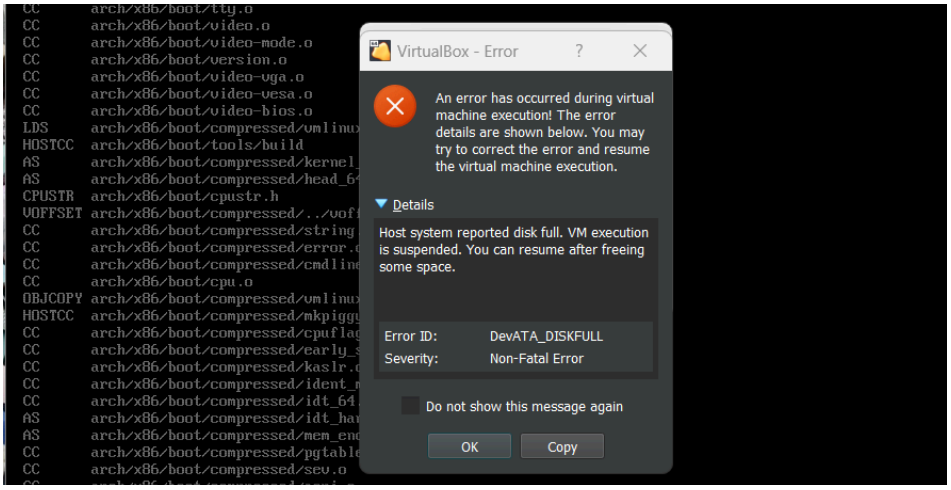
Module	Size	Used by
vboxguest	49152	6
cfg80211	1347584	0
8021q	40960	0
garp	16384	1 8021q
stp	12288	1 garp
mrp	20480	1 8021q
llc	16384	2 stp,garp
rfkill	40960	3 cfg80211
efivarfs	28672	1
ipv6	716800	36
vmwgfx	458752	2
snd_intel8x0	45056	2
drm_client_lib	12288	1 vmwgfx
drm_ttm_helper	16384	2 vmwgfx
snd_ac97_codec	196608	1 snd_intel8x0
ttm	102400	2 vmwgfx,drm_ttm_helper
ac97_bus	12288	1 snd_ac97_codec
intel_rapl_msr	16384	0
joydev	20480	0
drm_kms_helper	225280	3 vmwgfx,drm_ttm_helper,drm_client_lib
snd_pcm	176128	2 snd_intel8x0,snd_ac97_codec
snd_timer	45056	1 snd_pcm
ohci_pci	12288	0
i2c_piix4	24576	0
intel_agp	24576	0
psmouse	192512	0
drm	716800	8 vmwgfx,drm_kms_helper,drm_ttm_helper,drm_client_lib,ttm
snd	131072	8 snd_intel8x0,snd_timer,snd_ac97_codec,snd_pcm
ohci_hcd	45056	1 ohci_pci
i2c_smbus	12288	1 i2c_piix4
intel_gtt	24576	1 intel_agp
ehci_pci	12288	0
pcnet32	53248	0
intel_rapl_common	45056	1 intel_rapl_msr
ehci_hcd	69632	1 ehci_pci
e1000	151552	0
soundcore	16384	1 snd
ghash_clmulni_intel	16384	0
agpgart	45056	3 intel_agp,intel_gtt,ttm
mii	20480	1 pcnet32
evdev	28672	11
i2c_core	118784	5 drm_kms_helper,i2c_smbus,psmouse,i2c_piix4,drm
serio_raw	12288	0
button	20480	0
battery	24576	0
ac	12288	0
loop	36864	0

Module	Size	Used by
vboxguest	49152	6
cfg80211	1347584	0
8021q	40960	0
garp	16384	1 8021q
stp	12288	1 garp
mrp	20480	1 8021q
llc	16384	2 stp,garp
rfkill	40960	3 cfg80211
efivarfs	28672	1
ipv6	716800	36
vmwgfx	458752	2
drm_client_lib	12288	1 vmwgfx
snd_intel8x0	45056	2
drm_ttm_helper	16384	2 vmwgfx
snd_ac97_codec	196608	1 snd_intel8x0
ttm	102400	2 vmwgfx,drm_ttm_helper
ac97_bus	12288	1 snd_ac97_codec
snd_pcm	176128	2 snd_intel8x0,snd_ac97_codec
drm_kms_helper	225280	3 vmwgfx,drm_ttm_helper,drm_client_lib
snd_timer	45056	1 snd_pcm
intel_rapl_msr	16384	0
joydev	20480	0
i2c_piix4	24576	0
intel_agp	24576	0
drm	716800	8 vmwgfx,drm_kms_helper,drm_ttm_helper,drm_client_lib,ttm
psmouse	192512	0
snd	131072	8 snd_intel8x0,snd_timer,snd_ac97_codec,snd_pcm
i2c_smbus	12288	1 i2c_piix4
ohci_pci	12288	0
intel_gtt	24576	1 intel_agp
pcnet32	53248	0
ohci_hcd	45056	1 ohci_pci
ehci_pci	12288	0
intel_rapl_common	45056	1 intel_rapl_msr
video	69632	0
e1000	151552	0
agpgart	45056	3 intel_agp,intel_gtt,ttm
soundcore	16384	1 snd
ghash_clmulni_intel	16384	0
mii	20480	1 pcnet32
ehci_hcd	69632	1 ehci_pci
i2c_core	118784	5 drm_kms_helper,i2c_smbus,psmouse,i2c_piix4,drm
evdev	28672	13
wmi	28672	1 video
button	20480	0
serio_raw	12288	0
ac	12288	0
battery	24576	0
loop	36864	0

W praktyce nie dostrzegam różnic w załadowanych modułach (po lewej streamline po prawej loadmodconfig) z wyjątkiem tego że moduł evdev jest użyty 11 razy w nowej metodzie a 13 w starej.

Jeśli chodzi o różnice w procesie konfigurowania to wiadome jest że “stara metoda” jest prostsza w użyciu z racji że ogranicza się do napisania jednej komendy. Skrypt streamline ma opis działania i instrukcje, które warto przeczytać, więc skomplikowanie użycia również jest bardzo małe.

Problemów jako takich nie napotkałem w trakcie konfiguracji, ponieważ pilnowałem się by nie popełnić literówek co myślę byłoby uciążliwym do sprawdzania błędem. Jedynie podczas pierwszej próby konfiguracji maszyna wirtualna odmówiła dalszej pracy, bo nie zadbałem o miejsce na dysku.



Po ukończeniu zadania postanowiłem sprawdzić wielowątkowość procesu kompilacji jądra. W komendzie make -j<> bzImage, mimo że miałem ustawione w wirtualnej maszynie 4 rdzenie to postanowiłem sprawdzić szybkość dla innych podanych liczb.

Wirtualna maszyna: 4 rdzenie i 8 GB RAM

j == 5

```
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    14m15,624s
user    37m24,120s
sys     13m23,161s
```

j == 4

```
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    12m29,032s
user    31m59,451s
sys     10m54,646s
```

j == 3

```
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    14m14,625s
user    30m30,738s
sys     9m27,206s
```

Wniosek? Dziwne. Zwiększe RAM do 11 GB.

j == 5

```
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    12m42,453s
user    32m55,593s
sys     11m57,063s
```

Szczerze to oczekiwałem że wynik się nie poprawi, a tak to powinienem sprawdzić znowu dla j == 4

j == 4

```
  BUILD   arch/x86/boot/bzImage
Kernel: arch/x86/boot/bzImage is ready  (#1)

real    12m57,697s
user    33m10,669s
sys     11m24,644s
```

Wyniki:

1. 12m29,032s – j == 4 i 8 GB
2. 12m42,453s – j == 5 i 11 GB
3. 12m57,697s – j == 4 i 11 GB
4. 14m14,625s – j == 3 i 8 GB
5. 14m15,624s – j == 5 i 8 GB

Nie mam pojęcia jakim cudem mniejsza ilość ram poradziła sobie lepiej przy tej samej liczbie wątków. Myślałem, że może jeden dodatkowy wątek wypełniłby jakieś luki, w których procesor nie pracuje. Widocznie środowisko badań nie było jednakowe dla każdej próby i kończę test z wnioskiem, że lepiej nie kombinować.