

# Spis treści

1. Przygotowania wstępne .....	4
1.1. Maszyna wirtualna .....	4
1.1.1. Procesor .....	4
1.1.2. Pamięć RAM .....	5
1.2. Pliki źródłowe jądra .....	5
2. Metoda stara .....	7
2.1. Kopiowanie konfiguracji .....	7
2.2. Make localmodconfig – plik konfiguracyjny .....	7
2.3. Kompilacja obrazu jądra .....	8
3. Metoda nowa .....	12
3.1. Instrukcja .....	12
3.2. Config_strip .....	13
3.3. Kompilacja jądra .....	14
4. Hardinfo – Benchmark.....	18
Podsumowanie.....	21

# **Patryk Cwalina**

## **Kompilacja jądra**



**296480**  
**2022**

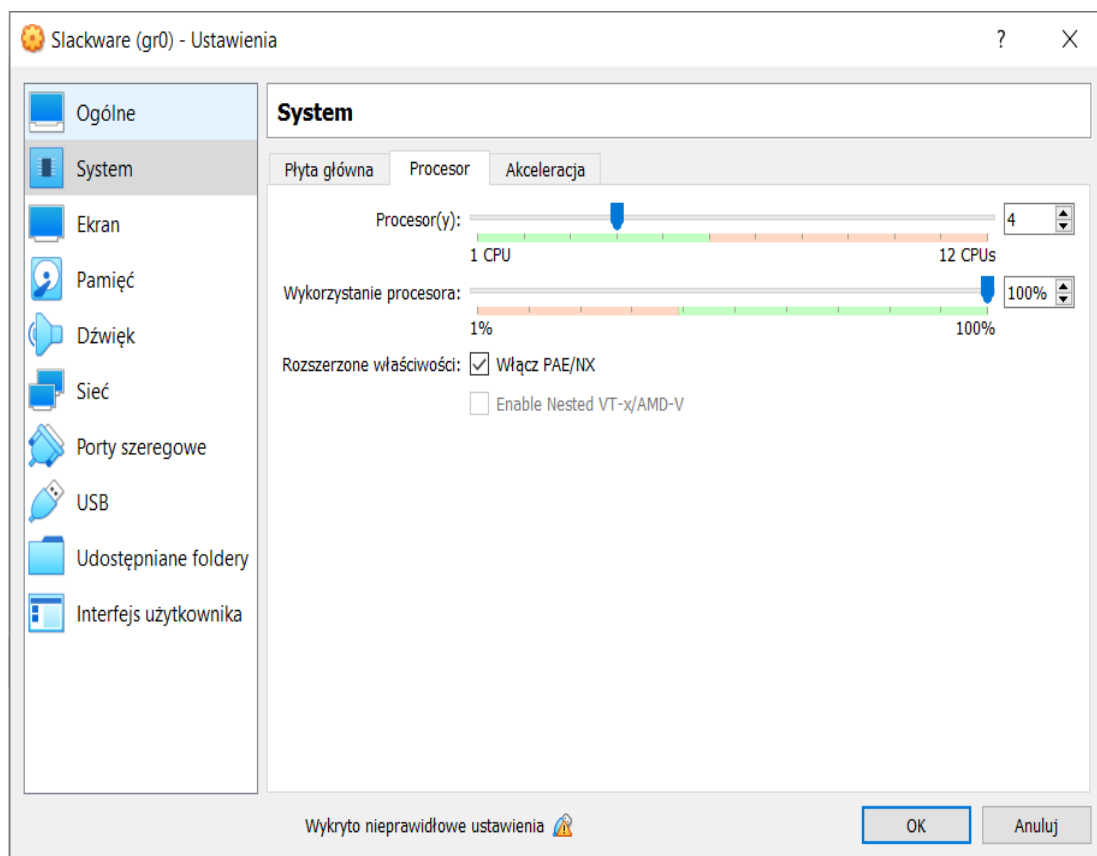


# 1. Przygotowania wstępne

## 1.1. Maszyna wirtualna

### 1.1.1. Procesor

Przed rozpoczęciem jakichkolwiek działań upewniam się, że moja maszyna wirtualna posiada wystarczającą ilość przydzielonych rdzeni procesora – przyspieszy to cały proces kompilacji

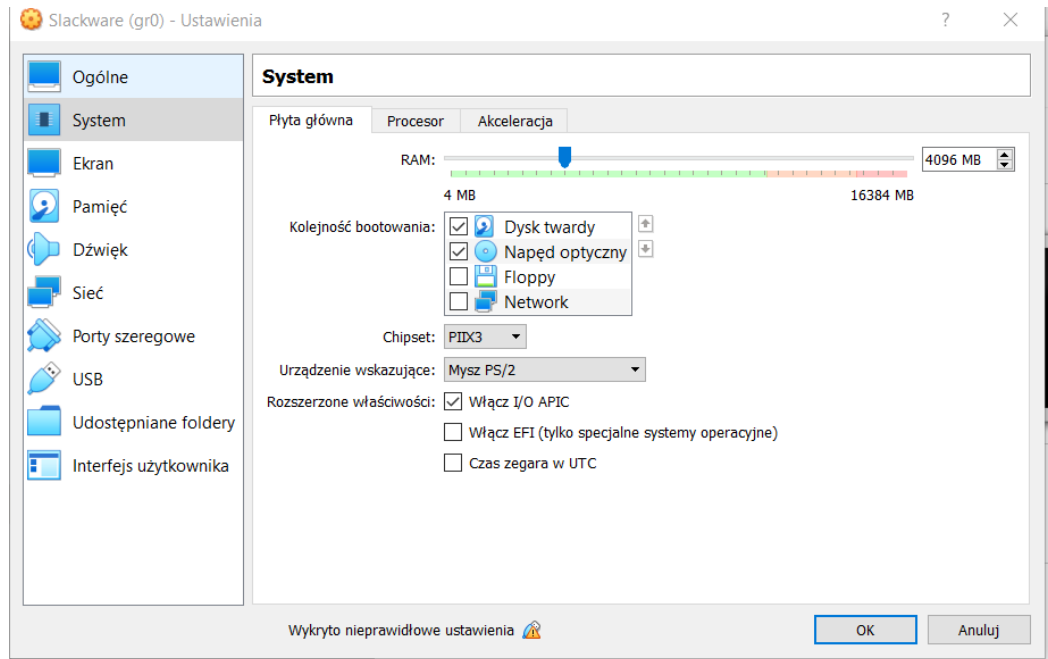


Rysunek 1:Ustawienia maszyny [CPU]

Jak widać na powyższym rysunku maszyna posiada 4 rdzenie procesora na 12 możliwych, nie chciałem ustawiać więcej, aby zbytnio nie obciążać prywatnego komputera .

### 1.1.2. Pamięć RAM

Początkowo chciałem ustawić pamięć ram na 8GB, jednak po wpisaniu frazy **‘Does RAM affect compiling kernel linux’** nie znalazłem źródeł, które wskazywałyby na to, że większa ilość pamięć RAM znacznie przyspieszy cały proces, więc uznałem ze dalsze obciążanie mojego komputera nie przyniesie oczekiwanych korzyści.



Rysunek 2: Ustawienia maszyny [RAM]

## 1.2. Pliki źródłowe jądra

Wersja jądra na zainstalowanej maszynie to **5.15.27-smp**, zatem możliwa jest aktualizacja do najnowszej i stabilnej wersji jaką na ten moment jest **5.18.3**. Pliki do tej wersji można pobrać ze strony [www.kernel.org](http://www.kernel.org) za pomocą komendy `wget` (jeżeli posiadamy połączenie z Internetem).

```
root@slack.localhost
-----
OS: Slackware 15.0 i586 i686
Host: Oracle Corporation VirtualBox
Kernel: 5.15.27-smp
Uptime: 48 mins
Packages: 1497 (pkgtool)
Shell: bash 5.1.16
Resolution: 800x600
Terminal: /dev/tty1
CPU: Intel i7-8750H (2) @ 2.208GHz
GPU: VMware SVGA II Adapter
Memory: 289MiB / 2001MiB
```

Rysunek 3: Wersja jądra na maszynie

Po pobraniu jądra za pomocą komendy *wget* należy je rozpakować za pomocą polecenia w tym przypadku użyłem komendy *tar*.

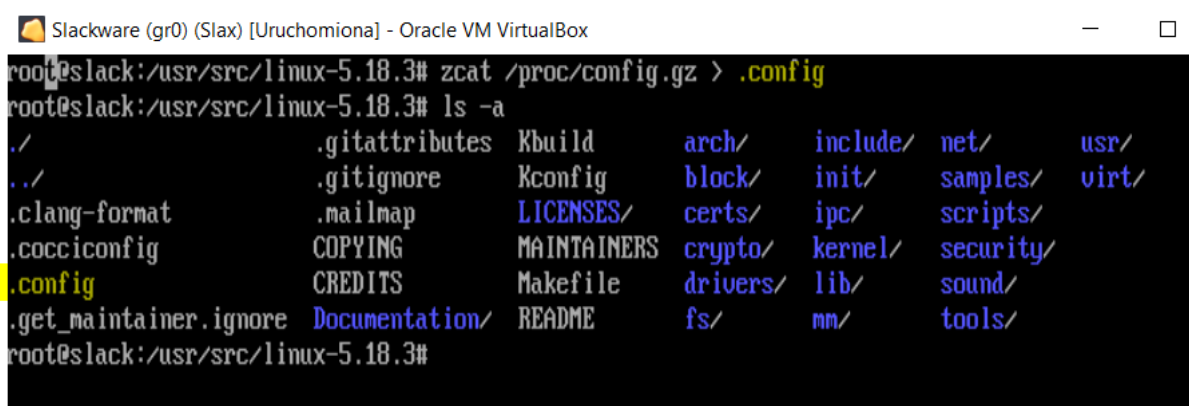
```
root@slack:/usr/src# ls
linux-5.18.3/ linux-5.18.3.tar.xz
root@slack:/usr/src# _
```

*Rysunek 4 Rozpakowane archiwum z jądrem*

## 2. Metoda stara

### 2.1. Kopiowanie konfiguracji

Kolejnym krokiem jest przekopiowanie konfiguracji jądra do pliku `.config`, w tym celu używam polecenia `cd` i wchodzę do katalogu z rozpakowanymi plikami nowego jądra, a następnie używam polecenia *zcat* *dodatkowa komenda* `ls` z *flagą* `-a` *pokaże ukryte pliki*.



```
Slackware (gr0) (Slax) [Uruchomiona] - Oracle VM VirtualBox
root@slack:/usr/src/linux-5.18.3# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.18.3# ls -a
./          .gitattributes  Kbuild      arch/       include/    net/        usr/
../         .gitignore     Kconfig     block/      init/       samples/    virt/
.clang-format .mailmap       LICENSES/   certs/      ipc/        scripts/
.cocciconfig  COPYING        MAINTAINERS crypto/      kernel/     security/
.config       CREDITS        Makefile    drivers/    lib/        sound/
.get_maintainer.ignore Documentation/  README     fs/         mm/         tools/
root@slack:/usr/src/linux-5.18.3#
```

Rysunek 5:Kopiowanie konfiguracji do pliku `.config`

### 2.2. Make localmodconfig – plik konfiguracyjny

Za pomocą komendy `make localmodconfig` przygotuje plik konfiguracyjny, należy pamiętać o tym, aby wszystkie moduły, których potrzebujemy były załadowane przed użyciem tej komendy, narzędzie, które może być do tego pomocne jest **Modprobed-db**, jednak nie chciałem zbyt komplikować sobie całego procesu kompilacji i uczyć się obsługiwanania tego narzędzia.

```

AMD ACPI2Platform devices support (X86_AMD_PLATFORM_DEVICE) [Y/n/?] y
Intel SoC IOSF Sideband support for SoC platforms (IOSF_MBI) [Y/?] y
Enable IOSF sideband access through debugfs (IOSF_MBI_DEBUG) [N/y/?] n
Eurobraille/Iris poweroff module (X86_32_IRIS) [N/m/y/?] n
Single-depth WCHAN output (SCHED_OMIT_FRAME_POINTER) [Y/n/?] y
Processor family
1. 486SX (M486SX)
2. 486DX (M486)
3. 586/K5/5x86/6x86/6x86MX (M586)
4. Pentium-Classic (M586TSC)
5. Pentium-MMX (M586MMX)
6. Pentium-Pro (M686)
7. Pentium-II/Celeron(pre-Coppermine) (MPENTIUMII)
> 8. Pentium-III/Celeron(Coppermine)/Pentium-III Xeon (MPENTIUMIII)
9. Pentium M (MPENTIUMM)
10. Pentium-4/Celeron(P4-based)/Pentium-4 M/older Xeon (MPENTIUM4)
11. K6/K6-II/K6-III (MK6)
12. Athlon/Duron/K7 (MK7)
13. Opteron/Athlon64/Hammer/K8 (MK8)
14. Crusoe (MCRUSOE)
15. Efficeon (MEFFICEON)
16. Winchip-C6 (MWINCHIPC6)
17. Winchip-2/Winchip-2A/Winchip-3 (MWINCHIP3D)
18. AMD Elan (MELAN)
19. GeodeGX1 (MGEODEGX1)
20. Geode GX/LX (MGEODE_LX)
21. CyrixIII/VIA-C3 (MCYRIXIII)
22. VIA C3-2 (Nehemiah) (MVIAC3_2)
23. VIA C7 (MVIAC7)
24. Core 2/newer Xeon (MCORE2)
25. Intel Atom (MATOM)
choice[1-25?]: 8
Generic x86 support (X86_GENERIC) [Y/n/?] y
HPET Timer Support (HPET_TIMER) [Y/n/?] y
Enable DMI scanning (DMI) [Y/n/?] y
Maximum number of CPUs (NR_CPUS) [32] 32
Cluster scheduler support (SCHED_CLUSTER) [Y/n/?] (NEW) _

```

Rysunek 6:Konsola po wpisaniu komendy `make localmodconfig`

Po przeklikaniu zaawansowanych ustawień konfiguracja jest zapisana do pliku `.config`, nie chciałem nic zmieniać w tych ustawieniach, ponieważ obawiałem się, że coś w późniejszych etapach pójdzie nie tak jak powinno.

## 2.3. Kompilacja obrazu jądra

Przydzieliłem mojej maszynie 4 z 12 dostępnych rdzeni, dlatego używam komendy `make bzImage` z parametrem `-j4` który przyspieszy cały proces.

```

Kernel: arch/x86/boot/bzImage is ready (#1)
root@slack:/usr/src/linux-5.18.3#

```

Rysunek 7 Wynik po ukończeniu kompilacji jądra

Po kompilacji obrazu jądra czas przystąpić do kompilacji modułów za pomocą komendy `make -j4 modules`.

```

LD [M] sound/pci/snd-intel8x0.ko
LD [M] sound/soundcore.ko
root@slack:/usr/src/linux-5.18.3#

```

Rysunek 8:Kompilacja modułów jądra



Gdy kompilacja modułów dokonała się pomyślnie przystępuje do instalacji modułów za pomocą komendy `make -j4 modules_install`

```
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#
```

Rysunek 9: Wynik instalacji modułów

Kolejnym krokiem jest przekopiowanie plików nowego jądra do katalogu skorzystam z komendy `cp`, katalog nazywa się **cwalina-kernel-old-method**

```
root@slack:/usr/src/linux-5.18.3# cp arch/x86/boot/bzImage /boot/vmlinuz-cwalina-kernel-old-method
root@slack:/usr/src/linux-5.18.3# cp System.map /boot/System.map-cwalina-kernel-old-method
root@slack:/usr/src/linux-5.18.3# cp .config /boot/config-cwalina-kernel-old-method
root@slack:/usr/src/linux-5.18.3# w _
```

Rysunek 10: Proces kopiowania plików jądra do nowego katalogu

W katalogu boot usuwamy starą tablicę symboli i zapisujemy ją linkiem symbolicznym do uprzednio skopiowanej tablicy w tym celu użyłem komend `rm` oraz `ln`

```
root@slack:/usr/src/linux-5.18.3# cd /boot/
root@slack:/boot# rm System.map
root@slack:/boot# ln -s System.map-cwalina-kernel-old-method System.map
root@slack:/boot# _
```

Rysunek 11: Usunięcie starej tablicy symboli i linkowanie

Po pomyślnym wykonaniu poprzedniego kroku czas na wygenerowanie komendy służącej do wygenerowania ramdisku, dodatkowo **należy pamiętać o flagie -k** która wskazuje na wersję jądra

```

root@slack:/usr/src/linux-5.18.3# /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:/usr/src/linux-5.18.3# _

```

Rysunek 12: Wygenerowana komenda za pomocą generatora komend

Po skopiowaniu wygenerowanej komendy i wywołaniu jej w terminalu pokazuje się następujący widok

```

root@slack:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-cwalina-
kernel-old.smp.gz
49039 bloków
/boot/initrd-cwalina-kernel-old.smp.gz created.
Be sure to run lilo again if you use it.

```

Rysunek 13: Wynik po wpisaniu wcześniej wspomnianej komendy

Jest to informacja o tym, że należy skonfigurować bootloader lilo, w tym celu należy odpalić plik lilo.conf edytorem, w moim przypadku jest to nano i dodać wpis który będzie odpalał system z nowym jądrem, wpis ten powinien znajdować się między zaznaczonymi obszarami.

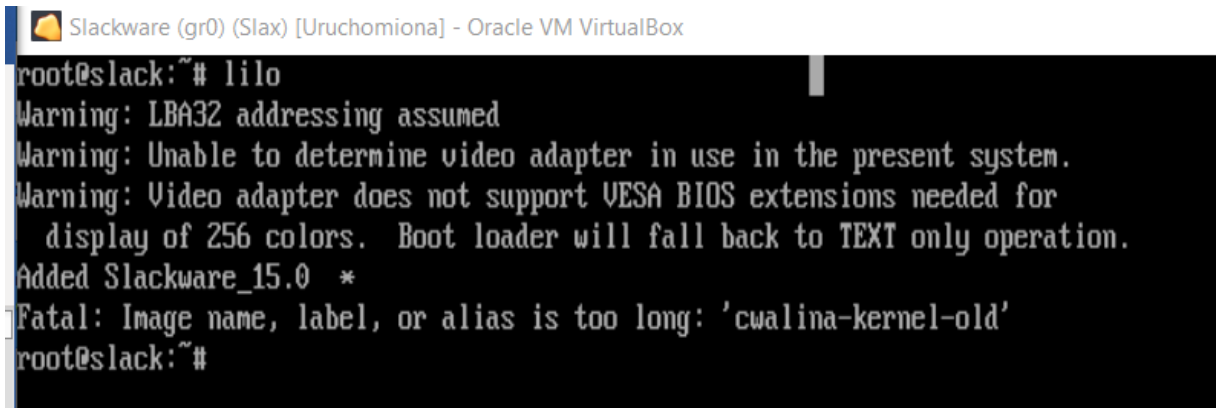
```

# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only
image = /boot/vmlinuz-cwalina-kernel-old-method
  root = /dev/sda1
  initrd = /boot/initrd-cwalina-kernel-old.smp.gz
  label = "cwalina-kernel-old"
  read-only_
# Linux bootable partition config ends

```

Rysunek 14: Zmiany w pliku lilo.conf

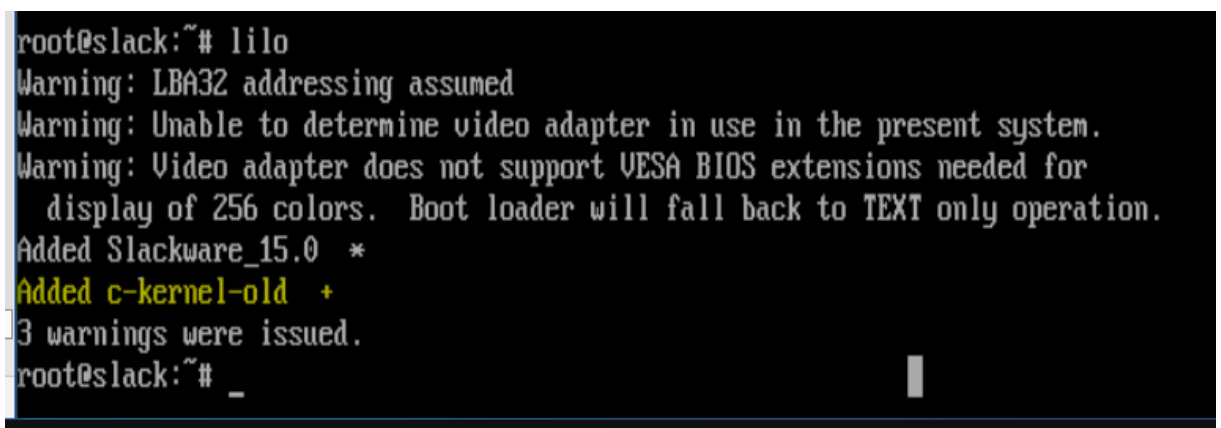
Po wykonaniu komendy lilo wystąpił następujący błąd



```
root@slack:~# lilo
Warning: LBA32 addressing assumed
Warning: Unable to determine video adapter in use in the present system.
Warning: Video adapter does not support VESA BIOS extensions needed for
display of 256 colors. Boot loader will fall back to TEXT only operation.
Added Slackware_15.0 *
Fatal: Image name, label, or alias is too long: 'cwalina-kernel-old'
root@slack:~#
```

Rysunek 15: Błąd podczas użycia komendy lilo

Nazwa obrazu, etykiety lub aliasu jest zbyt długa więc muszę ją zmienić, na moje szczęście okazało się, że nazwa etykiety jest za długa więc zmieniłem ją na **c-kernel-old**




```
root@slack:~# lilo
Warning: LBA32 addressing assumed
Warning: Unable to determine video adapter in use in the present system.
Warning: Video adapter does not support VESA BIOS extensions needed for
display of 256 colors. Boot loader will fall back to TEXT only operation.
Added Slackware_15.0 *
Added c-kernel-old +
3 warnings were issued.
root@slack:~# _
```

Rysunek 16: Pomyślna naprawa błędu, działanie komendy lilo

Po tych zmianach mogę zresetować wirtualną maszynę i sprawdzić czy w lilo mam dodatkową opcję wyboru.

Po resecie wirtualnej maszyny widzę nowy wpis, po odpaleniu systemu sprawdzam wersje jądra za pomocą komendy neofetch, jak widać kompilacja kernela starą metodą przebiegła pomyślnie, **aktualna wersja to 5.18.3-smp**

```
root@slack.localhost
-----
OS: Slackware 15.0 i586 i686
Host: Oracle Corporation VirtualBox
Kernel: 5.18.3-smp
Uptime: 40 secs
Packages: 1497 (pkgtool)
Shell: bash 5.1.16
Resolution: 1024x768
Terminal: /dev/tty1
CPU: Intel i7-8750H (2) @ 2.207GHz
GPU: VMware SVGA II Adapter
Memory: 287MiB / 2000MiB
```



Rysunek 17: Nowe jądro systemu

## 3. Metoda nowa

### 3.1. Instrukcja

Za pomocą komendy nano scripts/kconfig/strea,line\_config.pl otwieram plik, w którym znajduje się instrukcja jak przeprowadzić konfigurację dla nowej metody.

```

# 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 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
#

```

Rysunek 18: Instrukcja do metody nowej

## 3.2. Config\_strip

Kopiuje plik /boot/config oraz uruchamiam skrypt, a jego wyjście przekierowuje do pliku **config\_strip**

```

root@slack:/usr/src/linux-5.18.3# cp /boot/config .config
root@slack:/usr/src/linux-5.18.3# ./scripts/kconfig/streamline_config.pl > config_strip
using config: '.config'

```

Rysunek 19:Przekierowanie do config\_strip

Po tym kroku używam komendy make oldconfig

```

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 µs) (CLOCKSOURCE_WATCHDOG_MAX_SKEW_US) [100] (NEW)

```

Rysunek 20: Ostrzeżenia o symbolach

Pojawiają mi się ostrzeżenia o symbolach, jednak to zostało wygenerowane automatycznie więc się tym nie przejmuję, po przeklikaniu ogromnej ilości dodatkowych opcji których nie chciałem zmieniać moja konfiguracja została zapisana do pliku .config

### 3.3. Kompilacja jądra

Podobnie jak w metodzie starej kompiluje jądro tymi samymi komendami.

```

root@slack:/usr/src/linux-5.18.3# make -j4 bzImage
SYNC    include/config/auto.conf.cmd
CALL    scripts/atomic/check-atomics.sh
CC      arch/x86/kernel/asm-offsets.s
UPD     include/generated/asm-offsets.h
CALL    scripts/checksyscalls.sh
CC      init/main.o
AS      arch/x86/entry/entry_32.o
CC      arch/x86/events/core.o
CC      arch/x86/entry/syscall_32.o
CC      kernel/fork.o

```

Rysunek 21: Kompilacja jądra komendą make -j4 bzImage

Po kompilacji czas na moduły analogicznie używam komendy make -j4 modules

```
LD [M] sound/soc/sof/snd-sof-acpi.ko
LD [M] sound/soc/sof/snd-sof-pci.ko
LD [M] sound/soc/sof/snd-sof-probes.ko
LD [M] sound/soc/sof/snd-sof-utils.ko
LD [M] sound/soc/sof/snd-sof.ko
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/6fire/snd-usb-6fire.ko
LD [M] sound/usb/bcd2000/snd-bcd2000.ko
LD [M] sound/usb/caiaq/snd-usb-caiaq.ko
LD [M] sound/usb/hiface/snd-usb-hiface.ko
LD [M] sound/usb/line6/snd-usb-line6.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-pod.ko
LD [M] sound/usb/line6/snd-usb-variiax.ko
LD [M] sound/usb/misc/snd-ua101.ko
LD [M] sound/usb/snd-usb-audio.ko
LD [M] sound/usb/snd-usbmidi-lib.ko
LD [M] sound/usb/usx2y/snd-usb-us122l.ko
LD [M] sound/usb/usx2y/snd-usb-usx2y.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# wy_
```

Rysunek 22: Wynik kompilacji jądra

Po tym kroku przechodzę do instalacji modułów za pomocą tej samej komendy jak w przypadku metody starej `make modules_install`

```
INSTALL /lib/modules/5.18.3-smp/kernel/sound/virtio/virtio_snd.ko
INSTALL /lib/modules/5.18.3-smp/kernel/sound/x86/snd-hdmi-lpe-audio.ko
INSTALL /lib/modules/5.18.3-smp/kernel/virt/lib/irqbypass.ko
DEPMOD /lib/modules/5.18.3-smp
root@slack:/usr/src/linux-5.18.3# _
```

Rysunek 23: Wynik instalowania modułów

Po zainstalowaniu modułów tworzę przekopiuje pliki do folderu `/boot` i tworzę odpowiednie dowiązania

```

root@slack:/usr/src/linux-5.18.3# cp arch/x86/boot/bzImage /boot/vmlinuz-c-kernel-new
root@slack:/usr/src/linux-5.18.3# cp System.map /boot/System.map-c-kernel-new
root@slack:/usr/src/linux-5.18.3# mv /b
bin/ boot/
root@slack:/usr/src/linux-5.18.3# cp .c
.clang-format .coconfig .config .config.old
root@slack:/usr/src/linux-5.18.3# cp .config /b
bin/ boot/
root@slack:/usr/src/linux-5.18.3# cp .config /boot/config-c-kernel-new
root@slack:/usr/src/linux-5.18.3# cd /boot/
root@slack:/boot# rm System.map-
rm: nie można usunąć 'System.map-': Nie ma takiego pliku ani katalogu
root@slack:/boot# rm System.map
rm: nie można usunąć 'System.map': Nie ma takiego pliku ani katalogu
root@slack:/boot# ln -s System.map-c-kernel-new System.map
root@slack:/boot# _

```

Rysunek 24: Proces kopiowania plików do /boot/

Po tych czynnościach generuje ramdisk

```

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# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/in
initrd-cwalina-kernel-old.smp.gz initrd.gz inside.dat
initrd-tree/ inside.bmp
root@slack:/boot# mkinitrd -c -k 5.18.3-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-c-kernel-new.smp.gz
49399 bloków
/boot/initrd-c-kernel-new.smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot# _

```

Rysunek 25: Generowanie ramdisku za pomocą wygenerowanej komendy

Po wygenerowaniu ramdisku dodaje wpis do lilo.conf



```
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 15.0"
  read-only
image = /boot/vmlinuz-cwalina-kernel-old-method
  root = /dev/sda1
  initrd = /boot/initrd-cwalina-kernel-old.smp.gz
  label = "c-kernel-old"
  read-only
image = /boot/vmlinuz-c-kernel-new
  root = /dev/sda1
  initrd = /boot/initrd-c-kernel-new.smp.gz
  lable = "c-kernel-new"
  read-only
```

Rysunek 26: Wpis w pliku lilo.conf

```
Added c-kernel-old +
Unrecognized token "lable" at or above line 77 in file '/etc/lilo.conf'
root@slack:/boot#
```

Rysunek 27: Literówka w pliku lilo.conf


Po poprawieniu literówki, komenda lilo zadziałała resetuje maszynę i patrzę, czy istnieje nowy wpis



Rysunek 28: Nowy wpis

Jak widać nowy wpis jest widoczny, teraz przechodzę do uruchomienia

```
root@slack.localhost
-----
OS: Slackware 15.0 i586 i686
Host: Oracle Corporation VirtualBox
Kernel: 5.18.3-smp
Uptime: 36 secs
Packages: 1497 (pkgtool)
Shell: bash 5.1.16
Resolution: 1024x768
: Terminal: /dev/tty1
: CPU: Intel i7-8750H (2) @ 2.207GHz
: GPU: VMware SVGA II Adapter
Memory: 294MiB / 2000MiB
```



Rysunek 29:Zaktualizowana wersja jądra

Po uruchomieniu widzę, że jądro zostało zaktualizowane

## 4. Hardinfo – Benchmark

Za pomocą programu Hardinfo przetestuje oba jądra, aby zainstalować ten program użyłem komendy git clone i pobrałem go ze zdalnego repozytorium oraz użyłem komendy

*cmake make Install*

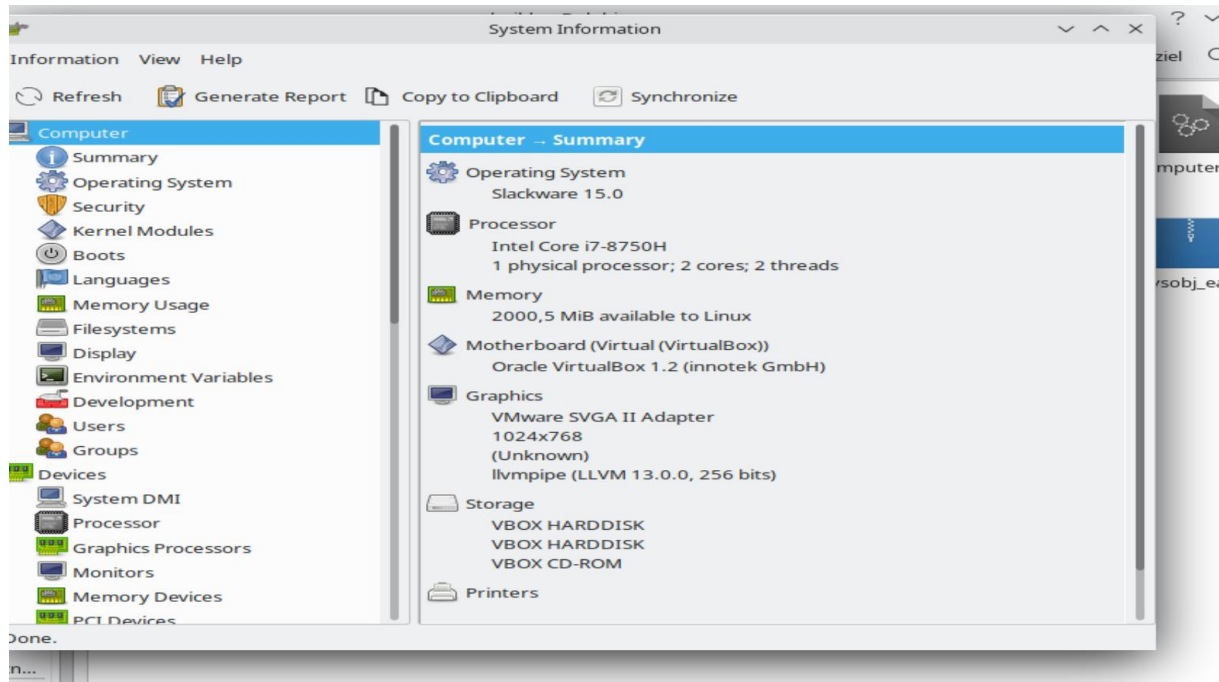
```

[ 65%] Building C object CMakeFiles/benchmark.dir/modules/benchmark.c.o
In file included from /root/hardinfo/modules/benchmark.c:35:
/root/hardinfo/modules/benchmark/bench_results.c: In function 'append_cpu_config':
/root/hardinfo/modules/benchmark/bench_results.c:350:39: warning: format '%ld' expects argument of type 'long int', but argume
3 has type 'gint64' {aka 'long long int'} [-Wformat=]
 350 |     g_string_append_printf(output, "%ldx %.2f %s", json_node_get_int(member_node),
      |                                     ^~^
      |                                     |
      |                                     long int      gint64 {aka long long int}
      |                                     %lld
[ 66%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/bench_util.c.o
[ 67%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/blowfish.c.o
[ 68%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/blowfish2.c.o
[ 69%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/cryptohash.c.o
[ 70%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/fbench.c.o
[ 71%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/fftbench.c.o
[ 72%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/fft.c.o
[ 73%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/fib.c.o
[ 74%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/nd5.c.o
[ 75%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/nqueens.c.o
[ 76%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/raytrace.c.o
[ 77%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/sha1.c.o
[ 78%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/zlib.c.o
[ 79%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/sysbench.c.o
[ 80%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/drawing.c.o
[ 81%] Building C object CMakeFiles/benchmark.dir/modules/benchmark/guibench.c.o
[ 82%] Linking C shared module benchmark.so
[ 82%] Built target benchmark
[ 83%] Building C object CMakeFiles/computer.dir/modules/computer.c.o
[ 84%] Building C object CMakeFiles/computer.dir/modules/computer/alsa.c.o
[ 85%] Building C object CMakeFiles/computer.dir/modules/computer/boots.c.o
[ 86%] Building C object CMakeFiles/computer.dir/modules/computer/display.c.o
[ 87%] Building C object CMakeFiles/computer.dir/modules/computer/environment.c.o
[ 88%] Building C object CMakeFiles/computer.dir/modules/computer/filesystem.c.o
[ 89%] Building C object CMakeFiles/computer.dir/modules/computer/languages.c.o
[ 90%] Building C object CMakeFiles/computer.dir/modules/computer/loadavg.c.o
[ 91%] Building C object CMakeFiles/computer.dir/modules/computer/memory.c.o
[ 92%] Building C object CMakeFiles/computer.dir/modules/computer/memory_usage.c.o
[ 93%] Building C object CMakeFiles/computer.dir/modules/computer/modules.c.o
[ 94%] Building C object CMakeFiles/computer.dir/modules/computer/os.c.o
[ 95%] Building C object CMakeFiles/computer.dir/modules/computer/ubuntu_flavors.c.o
[ 96%] Building C object CMakeFiles/computer.dir/modules/computer/uptime.c.o
[ 97%] Building C object CMakeFiles/computer.dir/modules/computer/users.c.o
[ 98%] Building C object CMakeFiles/computer.dir/modules/computer/groups.c.o
[100%] Linking C shared module computer.so
[100%] Built target computer
[100%] Built target i18n
root@slack:~/hardinfo/build#

```

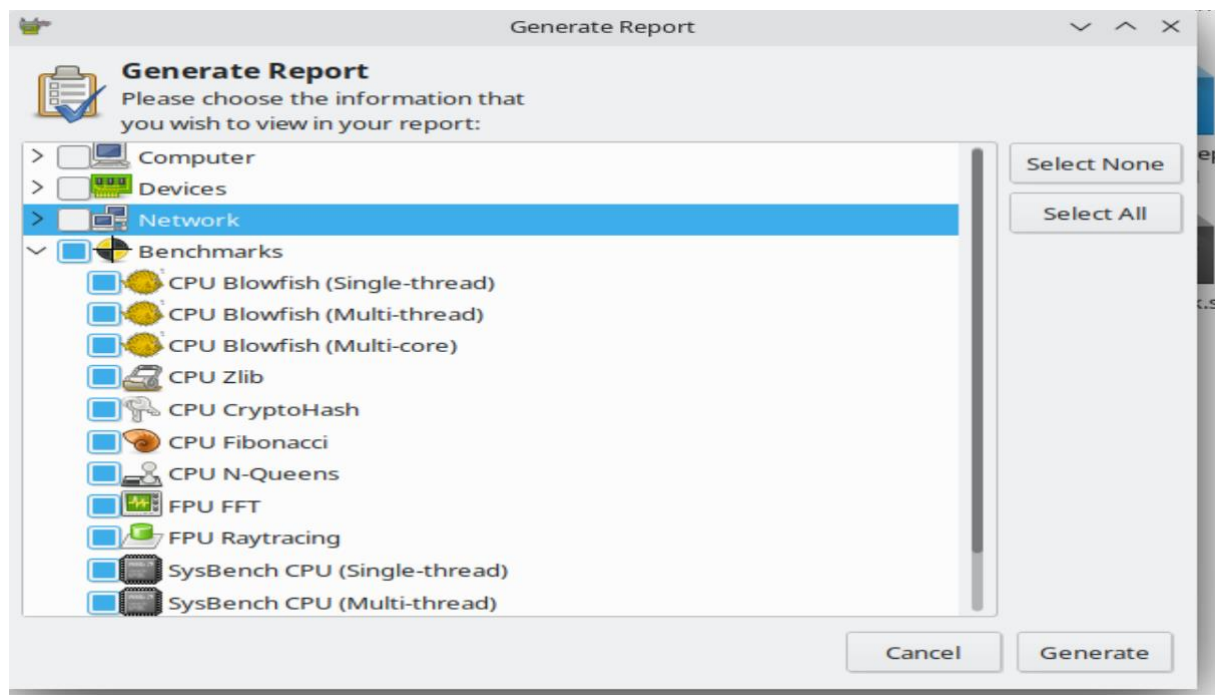
Rysunek 30: Instalacja Hardinfo

Przeszedłem do trybu graficznego i odpaliłem program Hardinfo



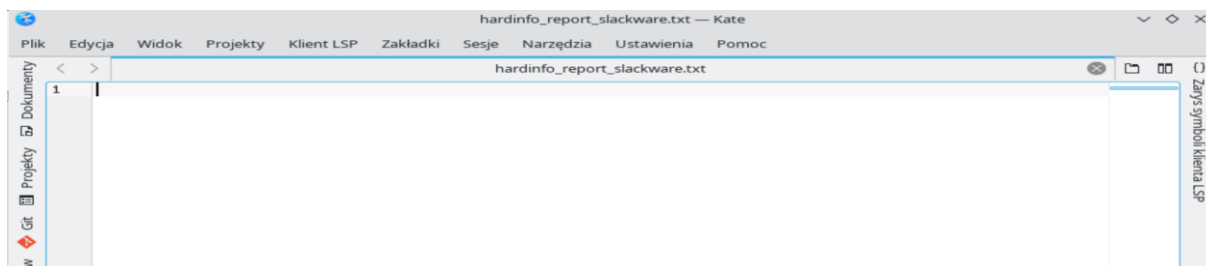
Rysunek 31:GUI Hardinfo

Pierwsze co zwróciło moją uwagę to dostępna pamięć oraz procesor, moja maszyna wirtualna ma inne ustawienia niż te które są widoczne na zrzucie ekranu



Rysunek 32:Ustawienia testów

Zaznaczam interesujące mnie opcje i generuje raport.



Rysunek 33: Błędy podczas generowania raportów

Niestety podczas wielu prób zrobienia benchmarka, za każdym razem miałem błąd, wyciek pamięci i raporty były zawsze puste, może to przez maszynę wirtualną, jej parametry były złe odczytywane od samego początku, uznałem że dalsza walka z Hardinfo nie ma sensu i zaprzestałem prób benchmarkowania.

## Podsumowanie

W przypadku obu metod kompilacja jądra zakończyła się pomyślnie poza małymi błędami takimi jak literówka w pliku lilo.conf bądź zbyt długa nazwa etykiety (brak jasnych wymagań co do zawartości pliku lilo.conf) nie miałem większych problemów jednak odniosłem wrażenie że kompilacja metoda nową trwała znacznie dłużej. W moim odczuciu żadna z metod nie jest przyjazna użytkownikowi, używam systemu Linux na co dzień w pracy, jednak miałem problemy z obiema metodami, ponieważ nie są one intuicyjne. Ilość zaawansowanej konfiguracji podczas tworzenia pliku konfiguracyjnego jest przytłaczająca, nawet gdybym chciał coś w nim zmienić to obawiałbym się że coś pójdzie nie tak. Instrukcja do nowej metody jest bardzo skromna, jednak ułatwia cały proces kompilacji.

Benchmark jądra niestety nie poszedł po mojej myśli, mam wrażenie że oprogramowanie Hardinfo nie działa wcale, bądź nie działa na maszynie wirtualnej.