Kacper Derlatka,

III rok informatyki, I stopień

Raport z przebiegu kompilacji jądra Linux przy użyciu dwóch metod generacji plików konfiguracyjnych

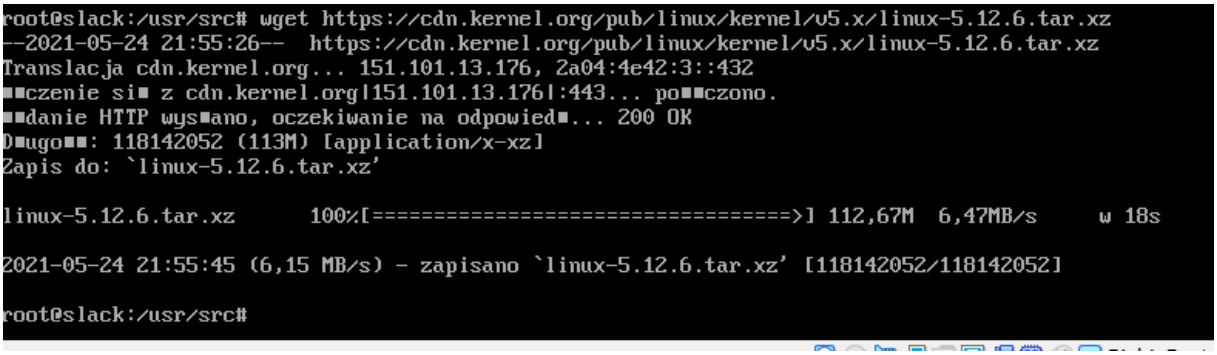
Lublin, 26 maja 2021

1. Przebieg procesu kompilacji dla metody nowej (streamline_config.pl)

W pierwszej kolejności przechodzę do katalogu /usr/src oraz pobieram archiwum z kodem źródłowym jądra za pomocą polecenia *wget*. Wersja jądra to 5.12.6. Prezentuje to Rysunek 1.

Rysunek 1. Pobieranie kodu źródłowego

Rysunek 2 przedstawia wynik wykonania polecenia. Kod źródłowy został poprawnie pobrany.

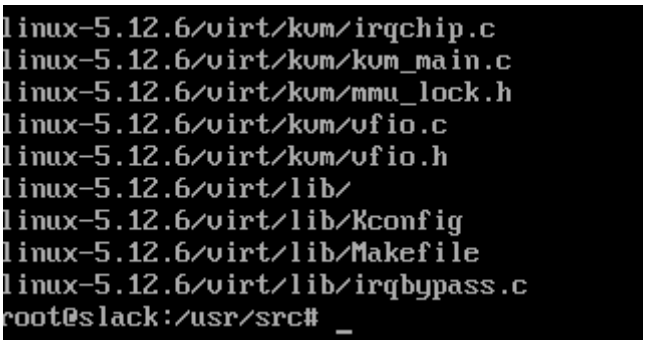


Rysunek 2. Wynik wykonania polecenia wget.

Kolejnym krokiem jest rozpakowanie plików znajdujących się w archiwum. Do tego posłużyło polecenie *tar* wraz z odpowiednimi opcjami (Rysunek 3).

Rysunek 3. Polecenie rozpakowujące archiwum

Udało się pomyślnie rozpakować pliki, co prezentuje Rysunek 4.



Rysunek 4. Terminal po rozpakowaniu plików

Następnie przechodzę do katalogu z plikami źródłowymi i ładuję konfigurację jądra na którym aktualnie pracuję do pliku .config. Proces przebiegł pomyślnie co pokazuje Rysunek 5.

```
root@slack:/usr/src# cd linux-5.12.6
root@slack:/usr/src/linux-5.12.6# zcat /proc/config.gz > .config
root@slack:/usr/src/linux-5.12.6#
```

Rysunek 5. Przejście do katalogu z plikami źródłowymi oraz załadowanie konfiguracji do pliku .config

By się upewnić że konfiguracja się tam znajduję włączam edycję pliku za pomocą *vim* (Rysunek 6).

```
root@slack:/usr/src/linux-5.12.6# vim .config
```

Rysunek 6. Edycja pliku .config

Rysunek 7 pokazuje że rzeczywiście konfiguracja dla jądra 4.4.261 na którym aktualnie pracuję została umieszczona w pliku .config.

```
#
# Automatically generated file; DO NOT EDIT.
# Linux/x86 4.4.261 Kernel Configuration
#
# CONFIG_64BIT is not set
CONFIG_X86_32=y
CONFIG_X86=y
CONFIG_INSTRUCTION_DECODER=y
CONFIG_PERF_EVENTS_INTEL_UNCORE=y
CONFIG_OUTPUT_FORMAT="elf32-i386"
CONFIG_ARCH_DEFCONFIG="arch/x86/configs/i386_defconfig"
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_HAVE_LATENCYTOP_SUPPORT=y
CONFIG_MMU=y
CONFIG_NEED_DMA_MAP_STATE=y
CONFIG_NEED_SG_DMA_LENGTH=y
CONFIG_GENERIC_ISA_DMA=y
CONFIG_GENERIC_BUG=y
CONFIG_GENERIC_HWEIGHT=y
CONFIG_ARCH_MAY_HAVE_PC_FDC=y
CONFIG_RWSEM_XCHGADD_ALGORITHM=y
CONFIG_GENERIC_CALIBRATE_DELAY=y
CONFIG_ARCH_HAS_CPU_RELAX=y
CONFIG_ARCH_HAS_CACHE_LINE_SIZE=y
CONFIG_HAVE_SETUP_PER_CPU_AREA=y
CONFIG_NEED_PER_CPU_EMBED_FIRST_CHUNK=y
CONFIG_NEED_PER_CPU_PAGE_FIRST_CHUNK=y
CONFIG_ARCH_HIBERNATION_POSSIBLE=y
CONFIG_ARCH_SUSPEND_POSSIBLE=y
CONFIG_ARCH_WANT_HUGE_PMD_SHARE=y
CONFIG_ARCH_WANT_GENERAL_HUGETLB=y
CONFIG_ARCH_SUPPORTS_OPTIMIZED_INLINING=y
CONFIG_ARCH_SUPPORTS_DEBUG_PAGEALLOC=y
CONFIG_HAVE_INTEL_TXT=y
CONFIG_X86_32_SMP=y
".config" 7400L, 163669C
```

Rysunek 7. Zawartość pliku .config

W kolejnym kroku rozpoczynam proces generowania pliku konfiguracjnego przy użyciu *streamline_config.pl*. Włączam edycję tego pliku (Rysunek 8).

```
root@slack:/usr/src/linux-5.12.6# vim scripts/kconfig/streamline_config.pl _
```

Rysunek 8. Edycja pliku streamline_config.pl

Rysunek 9 przedstawia wynik wykonania polecenia z Rysunku 8. Plik ten to skrypt w języku perl, dzięki któremu ograniczamy liczbę modułów w jądrze poprzez wykorzystanie tylko tych z których aktualnie korzystamy.

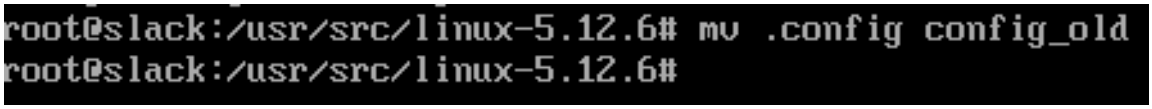
```
#!/usr/bin/env perl
# SPDX-License-Identifier: GPL-2.0
#
# Copyright 2005-2009 - Steven Rostedt
#
#  It's simple enough to figure out how this works.
#  If not, then you can ask me at stripconfig@goodmis.org
#
# What it does?
#
#   If you have installed a Linux kernel from a distribution
#   that 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 configuraton 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.
"scripts/kconfig/streamline_config.pl" 704L, 17084C
```

Rysunek 9. Zawartość pliku streamline_config.pl

Tak więc w kolejnym kroku, zgodnie z instrukcją ze skryptu, generowany jest nowy plik konfiguracyjny i zapisany pod nazwą *config_strip*. Generowanie przebiegło bez problemów Rysunek 10).

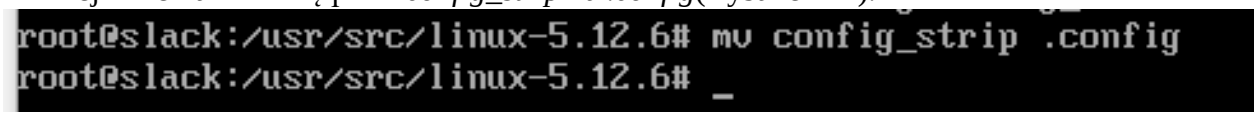
Rysunek 10. Wywołanie skryptu streamline_config.pl

Następnym krokiem jest zapisanie pliku *.conf*(konfiuracja dla jądra 4.4.261) pod inną nazwą (Rysunek 11).



Rysunek 11. Zmiana nazwy pliku .config

Później zmieniam nazwę pliku *config_strip* na *.config*(Rysunek 12).

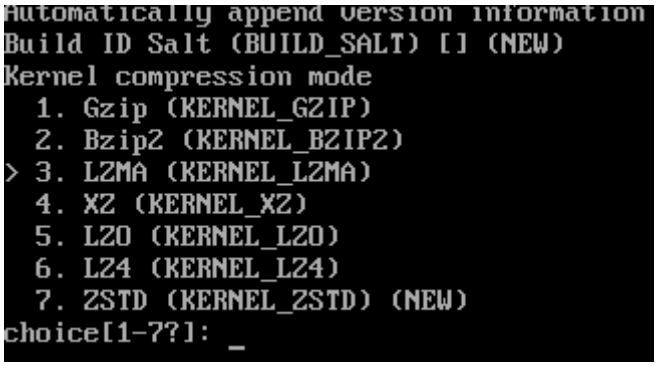


Rysunek 12. Zmiana nazwy pliku config_strip

Teraz, zgodnie z instrukcją napisaną przez autora skryptu, wywołuję *make oldconfig*(Rysunek 13).

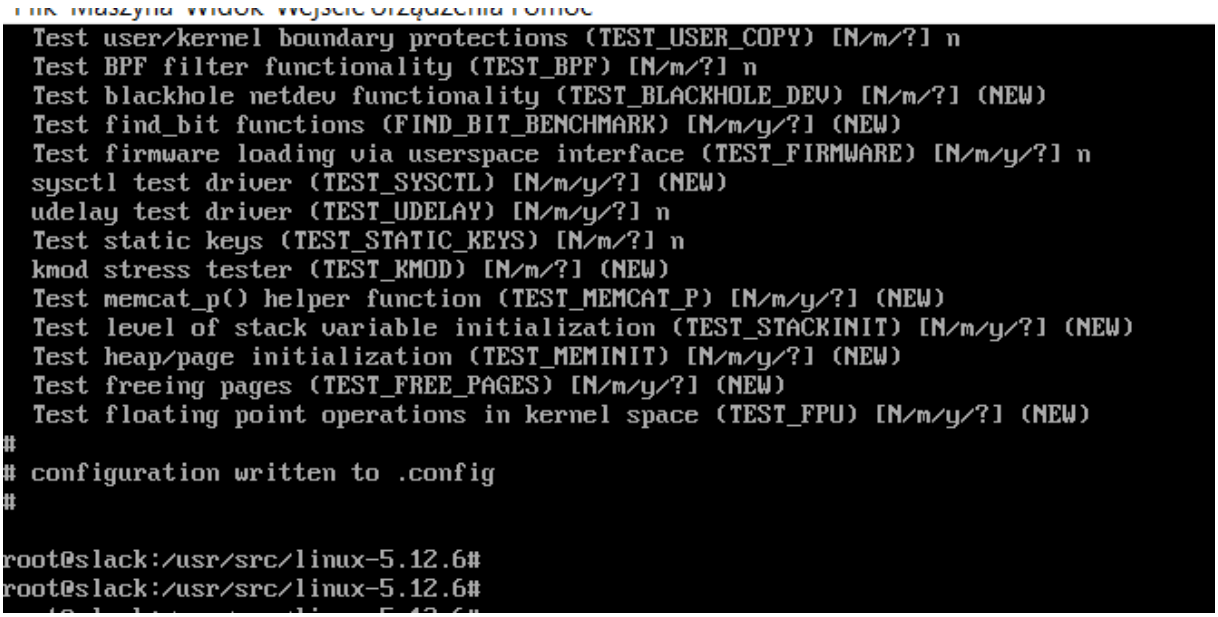
Rysunek 13. Wywołanie make oldconfig

Następnie wyświetla się seria pytań (Rysunek 14) odnośnie wyborów różnych opcji kompilacji jądra. Ja zastosowałem wszędzie opcję domyślną i po prostu wciskałem Enter.



Rysunek 14. Proces generowania pliku konfiguracyjnego

Po wybraniu wszystkich opcji konfiguracja została zapisana do pliku .config (Rysunek 15).

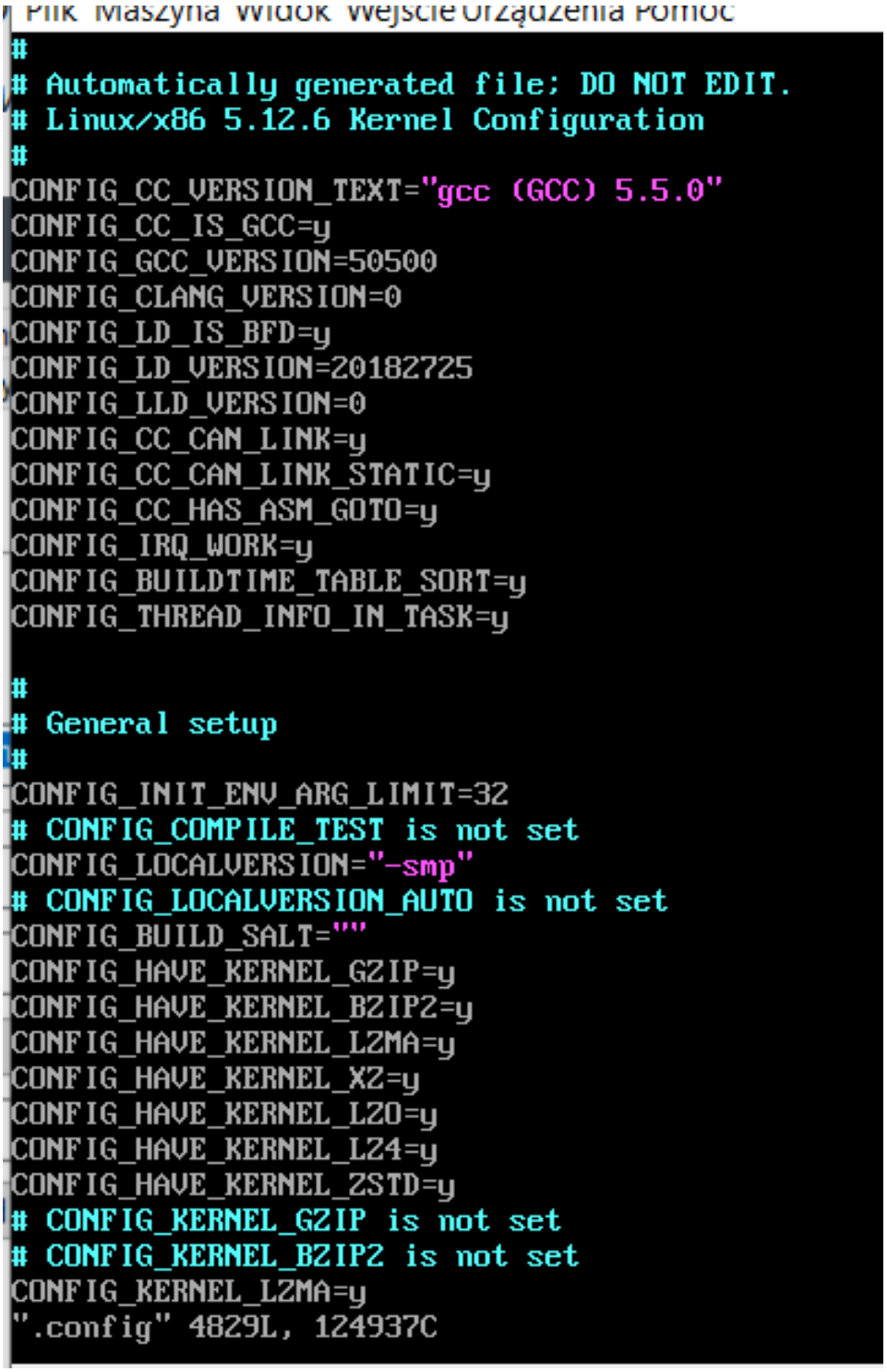


Rysunek 15. Pomyślne zakończenie generowania pliku .config

Za pomocą *vim* weryfikuję czy konfiguracja tam się znajduje (Rysunek 16).

Rysunek 16. Edycja pliku .config

I jak widać na Rysunku 17, w pliku *.config* znajduje się konfiguracja jądra 5.12.6.



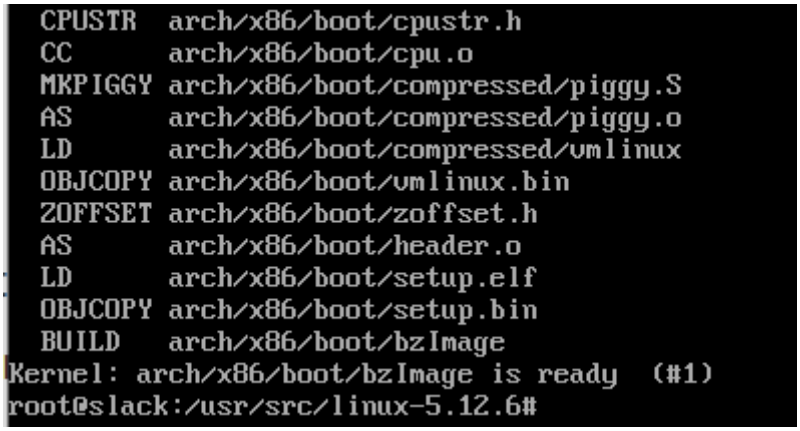
Rysunek 17. Zawarość pliku .config

Teraz, mając gotowy plik konfiguracyjny, przystępuję do kompilacji kernela przy pomocy polecenia *make*. Opcja -j2 oznacza wykorzystanie dwóch rdzeni przy procesie kompilacji (Rysunek 18).

```
root@slack:/usr/src/linux-5.12.6# make -j2 bzImage
```

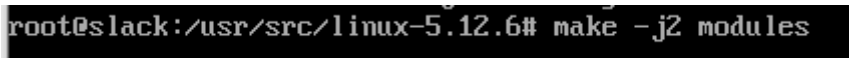
Rysunek 18. Kompilacja jądra

Jądro zostało skompilowane pomyślnie, co prezentuje Rysunek 19.



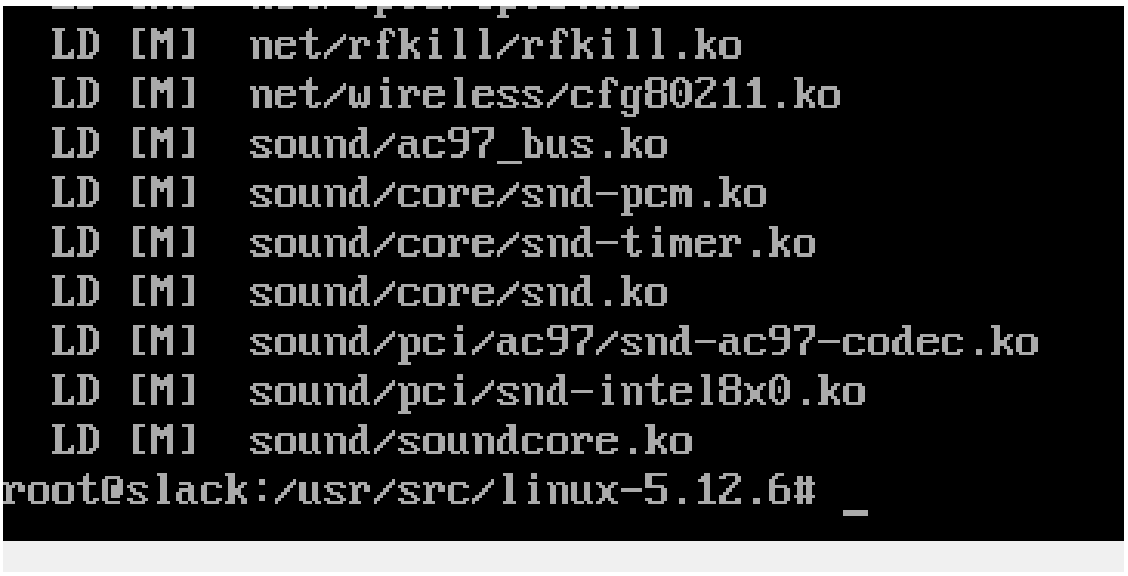
Rysunek 19. Pomyślne zakończenie kompilacji jądra

Teraz przystępuję do kompilacji modułów, również wykorzystując 2 rdzenie (Rysunek 20).



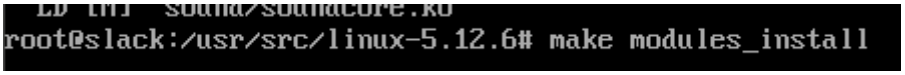
Rysunek 20. Kompilacja modułów

Po zakończeniu kompilacji okno terminala nie wyświetla żadnych błędów, czyli proces przebiegł pomyślnie (Rysunek 21)



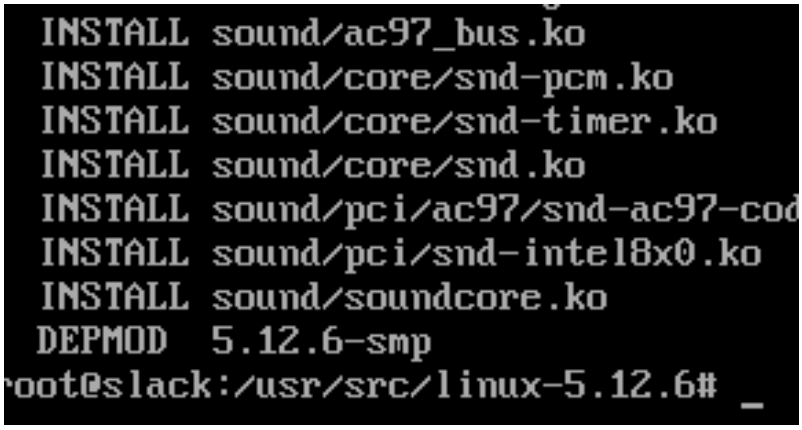
Rysunek 21. Pomyślne zakończenie kompilacji modułów

Następnie przystępuję do instalacji modułów (Rysunek 22).



Rysunek 22. Instalacja modułów

Proces ten również przebiegł pomyślnie – nie wyświetliły się żadne błędy (Rysunek 23)



Rysunek 23. Pomyślne zakończenie instalacji modułów

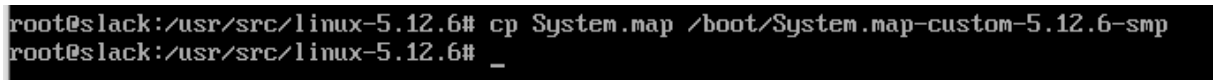
Poleceniem *ls* sprawdzam czy moduły się rzeczywiście zainstalowały (Rysunek 24). Wszystko wygląda poprawnie.

Rysunek 24. Lista modułów

Teraz przystępuję do przekopiowania plików kernela do systemu. W tym celu na początku kopiuję obraz jądra do katalogu /boot (Rysunek 25). Dodaję *custom* do nazwy obrazu by wiedzieć że jest to ręcznie kompilowane jądro.

Rysunek 25. Kopiowanie obrazu jądra

Następnie kopiuję tablicę symboli do katalogu /boot (Rysunek 26). Tablica ta zawiera informacje o funkcjach stosowanych przez kernel(m.in położenie w pamięci).

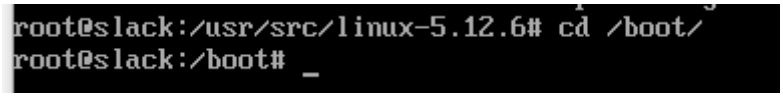


Rysunek 26. Kopiowanie tablicy symboli

W kolejnym kroku kopiuję plik konfiguracyjny kernela (Rysunek 27).

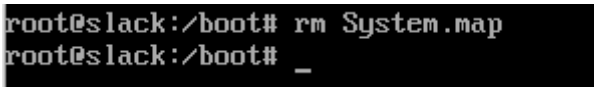
Rysunek 27. Kopiowanie pliku konfiguracyjnego

Teraz, gdy pliki kernela są przekopiowane, przechodzę do katalou /boot (Rysunek 28).



Rysunek 28. Przejście do /boot

Usuwam link symboliczny System.map za pomocą *rm* (Rysunek 29).

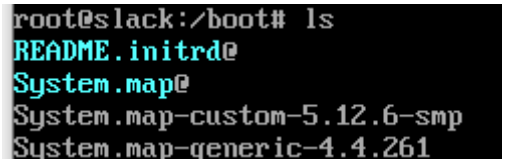


Rysunek 29. Usunięcie System.map

A następnie tworzę nowy link symboliczny wskazujący na tablicę symboli nowego kernela (Rysunek 30).

Rysunek 30. Stworzenie nowego linku symbolicznego

Poleceniem *ls* sprawdzam czy link się utworzył (Rysunek 31).



Rysunek 31. Sprawdzenie czy nowy link się stworzył

Teraz przystępuję do tworzenia dysku ram. Na początku wywołuję skrypt generujący komendę do wykonania (Rysunek 32). Parametr *-k* określa dla jakiej wersji jądra będzie generowany dysk ram.

Rysunek 32. Generowanie komendy do stworzenia dysku ram

Wynik działania skryptu widoczny jest na Rysunku 33.

Rysunek 33. Wynik działania skryptu

Kopiuję komendę otrzymaną w wyniku działania skryptu, zmieniam nazwę wynikowego pliku i ją wykonuję (Rysunek 34).

```
root@slack:/boot#  mkinitrd -c -k 5.12.6-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd-custom-
5.12.6-smp.gz
31927 bloków
/boot/initrd-custom-5.12.6-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot# _
```

Rysunek 34. Generowanie dysku ram

Komenda wykonała się poprawnie. Ostatnim krokiem jest dodanie nowego wpisu do konfiguracji bootloadera lilo. W tym celu przechodzę do pliku */etc/lilo.conf* (Rysunek 35).

```
root@slack:/boot# vim /etc/lilo.conf_
```

Rysunek 35. Edycja lilo.conf

Na końcu pliku dodaje wpis zawierający informacje o nowym kernelu (by bootloader mógł go wykrył)(Rysunel 36).

```
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 14.2"
  read-only

image = /boot/vmlinuz-custom-5.12.6-smp
  root = /dev/sda1
  initrd = /boot/initrd-custom-5.12.6-smp.gz
  label = "slack_custom"
  read-only
# Linux bootable partition config ends
"/etc/lilo.conf" 74L, 2193C zapisano
```

Rysunek 36. Dodanie wpisu do lilo.conf

Zapisuję nową konfigurację, wychodzę z edytora *vim* i wykonuję komendę *lilo* (Rysunek 37).

```
root@slack:/boot# 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_14.2  *
Added slack_custom  +
3 warnings were issued.
root@slack:/boot#
```

Rysunek 37. Wykonanie komendy lilo

Dostaję informację że pomyślnie dodano nowe wpisy. Teraz robię reboot (Rysunek 38).

```
root@slack:/boot# reboot
```

Rysunek 38. Ponowne uruchomienie systemu

Rysunek 39 przedstawia opcje wyboru systemu operacyjnego. Wybieram *slack_custom* i klikam Enter.

Rysunek 39. Ekran wyboru OS

System się ładuje poprawnie i po chwili wyświetla się opcja logowania (Rysunek 40). System został zainstalowany poprawnie.

Rysunek 40. Ekran logowania do systemu

2. Przebieg procesu kompilacji dla metody starej (localmodconfig)

W celu pokazania alternatywnej metody generowania pliku konfiguracyjnego loguję się na pierwotną wersję (Slackware_14.2) i przechodzę do katalogu */usr/src* (Rysunek 41).

Rysunek 41. Przejście do katalogu /usr/src

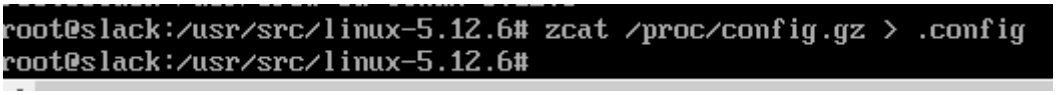
Następnie usuwam katalog z poprzednimi plikaki źródłowymi (Rysunek 42).

Rysunek 42. Usunięcie katalogu z plikami źródłowymi

W kolejnym kroku znowu wypakowuję archiwum z kodem źródłowym (Rysunek 43).

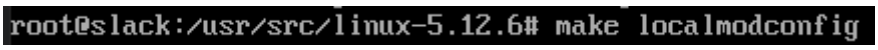
Rysunek 43. Rozpakowanie archiwum

Przechodzę do wypakowanego katalogu i znów kopiuję plik .config zawierający konfigurację aktualnego jądra (Rysunek 44).



Rysunek 44. Skopiowanie konfiguracji

W celu wygenerowania nowego pliku konfiguracyjnego wykonuję metodę *make localmodconfig* (Rysunek 45).



Rysunek 45. Wygenerowanie nowego pliku konfiguracyjnego

Pojawiają się tu również opcje do wyboru i podobnie jak w poprzedniej metodzie wszędzie wybieram opcję domyślną wciskając za każdym radem Enter. Po przejściu przez wszystkie opcje (Rysunek 46) plik konfiguracyjny jest gotowy.

Rysunek 46. Wynik działania make localmodconfig

Teraz przystępuję do kompilacji jądra (Rysunek 47). Tutaj również korzystam z dwóch rdzeni by przyspieszyć proces.

```
root@slack:/usr/src/linux-5.12.6# make -j2 bzImage
```

Rysunek 47. Komplilacja jądra

Rysunek 48 pokazuje, że kompilacja kernela przebiegła pomyślnie.

```
  OBJCOPY arch/x86/boot/vmlinux.bin
  ZOFFSET arch/x86/boot/zoffset.h
  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-5.12.6#
```

Rysunek 48. Wynik kompilacji

Kolejnym krokiem jest kompilacja modułów (Rysunek 49).

```
root@slack:/usr/src/linux-5.12.6# make -j2 modules
```

Rysunek 49. Kompilacja modułów

Tutaj również wszystko wykonuje się poprawnie (Rysunek 50).

```
  LD [M]  sound/core/snd-timer.ko
  LD [M]  sound/core/snd.ko
  LD [M]  sound/pci/ac97/snd-ac97-code
  LD [M]  sound/pci/snd-intel8x0.ko
  LD [M]  sound/soundcore.ko
root@slack:/usr/src/linux-5.12.6#
```

Rysunek 50. Wynik kompilacji modułów

Następnym etapem jest instalacja skompilowanych modułów (Rysunek 51).

```
root@slack:/usr/src/linux-5.12.6# make modules_install
```

Rysunek 51. Instalacja modułów

Podobnie jak poprzednio, proces wykonuje się bezbłędnie (Rysunek 52).

```
  INSTALL sound/pci/snd-intel8x0.ko
  INSTALL sound/soundcore.ko
  DEPMOD  5.12.6-smp
root@slack:/usr/src/linux-5.12.6#
```

Rysunek 52. Wynik instalacji modułów

Sprawdzam, czy moduły się zainstalowały (Rysunek 53).

Rysunek 53. Wylistowanie zainstalowanych modułów

Teraz przystępuję do kopiowania plików kernela do katalogu /boot. Na początku obraz jądra (Rysunek 54).

Rysunek 54. Kopiowanie obrazu jądra

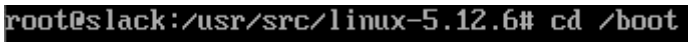
Następnie kopiuję tablicę symboli (Rysunek 55).

Rysunek 55. Kopiowanie tablicy symboli

Oraz na końcu plik konfiguracyjny (Rysunek 56).

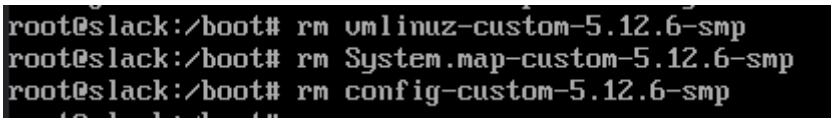
Rysunek 56. Kopiowanie pliku konfiguracyjnego

Teraz przechodzę komendą *cd* do katalogu /boot (Rysunek 57)



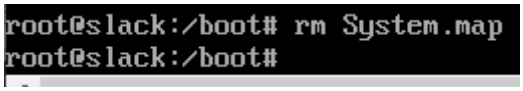
Rysunek 57. Przejście do katalogu /boot

Na początku usuwam pliki z poprzedniego jądra o tej samej wersji (Rysunek 58).



Rysunek 58. Usunięcie niepotrzebnych plików

Następnie usuwam link symboliczny (Rysunek 59).



Rysunek 59. Usunięcie linku symbolicznego

Później tworzę nowy link symboliczny wskazujący na kernel przed chwilą skompilowany (Rysunek 60).

Rysunek 60. Stworzenie nowego linku symbolicznego

Wywołuję skrypt generujący komendę do stworzenia dysku ram (Rysunek 61)

```
root@slack:/boot# /usr/share/mkinitrd/mkinitrd_command_generator.sh -k 5.12.6-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 5.12.6-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot#
```

Rysunek 61. Generowanie komendy do stworzenia dysku ram

Kopiuję i wykonuję otrzymaną w wyniku komendę zmieniając nazwę wynikowego pliku (Rysunek 62)

```
root@slack:/boot# mkinitrd -c -k 5.12.6-smp -f ext4 -r /dev/sda1/ -m ext4 -u -o /boot/initrd-custom2ndMethod-5.12.6-smp.gz
```

Rysunek 62. Wykonanie metody do stworzenia dysku ram

Wynik jej działania jest zaprezentowany na Rysunku 63.

```
mkinitrd -c -k 5.12.6-smp -f ext4 -r /dev/sda1 -m ext4 -u -o /boot/initrd.gz
root@slack:/boot# mkinitrd -c -k 5.12.6-smp -f ext4 -r /dev/sda1/ -m ext4 -u -o /boot/initrd-custom2ndMethod-5.12.6-smp.gz
31927 bloków
/boot/initrd-custom2ndMethod-5.12.6-smp.gz created.
Be sure to run lilo again if you use it.
root@slack:/boot#
```

Rysunek 63. Wynik działania metody mkinitrd

Teraz przechodzę do pliku */etc/lilo.conf* (Rysunek 64).

```
root@slack:/boot# vim /etc/lilo.conf
```

Rysunek 64. Edycja pliku lilo.conf

Następnie modyfikuję wpis tak, by bootloader wykrywał nową wersję jądra (Rysunek 65).

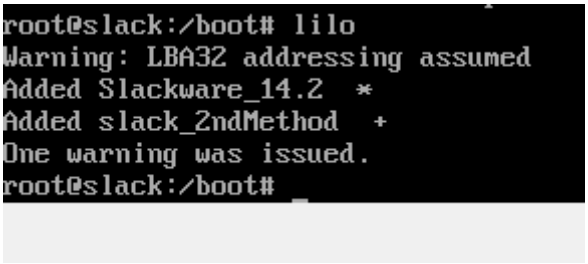
```
# End LILO global section
# Linux bootable partition config begins
image = /boot/vmlinuz
  root = /dev/sda1
  label = "Slackware 14.2"
  read-only

image = /boot/vmlinuz-custom2ndMethod-5.12.6-smp
  root = /dev/sda1
  initrd = /boot/initrd-custom2ndMethod-5.12.6-smp.gz
  label = "slack_2ndMethod"
  read-only
# Linux bootable partition config ends
"/etc/lilo.conf" 74L, 2214C zapisano
```

Rysunek 65. Dodanie wpisów do pliku lilo.conf

Plik zapisuję i wykonuję metodę *lilo* (Rysunek 66).

Rysunek 66. Wykonanie komendy lilo.

Rysunek 67 pokazuje, że pomyślnie dodano wpisy.



Rysunek 67. Wynik działania komendy lilo.

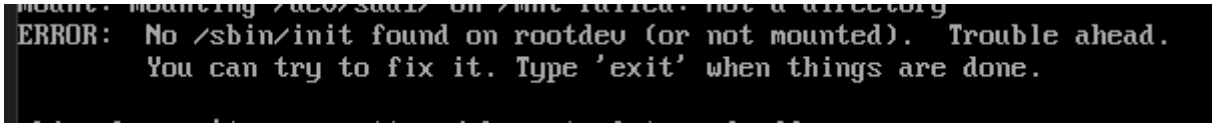
Tak więc w kolejnym kroku robię reboot (Rysunek 68).

Rysunek 68. Reboot

Bootloader widzi system, więc go wybieram i wciskam Enter (Rysunek 69).

Rysunek 69. Ekran wyboru systemu operacyjnego

Po chwili nieoczekiwanie wyświetla się błąd i kernel przechodzi w tryb Kernel Panic (Rysunek 70).

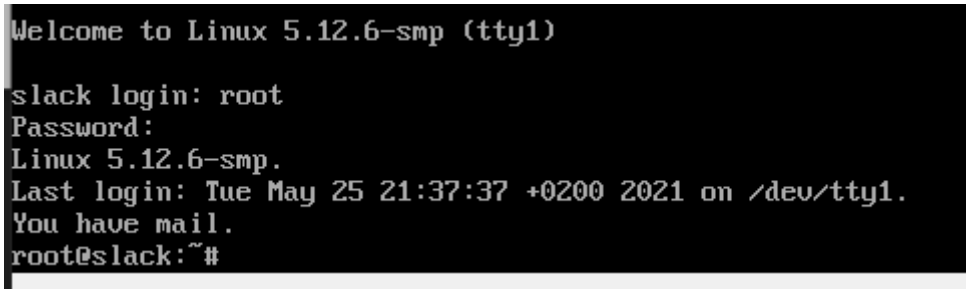


Rysunek 70. Błąd

Zalogowałem się na Slackware_14.2 i wszedłem do katalogu /boot, bo ewidentnie kernel miał problem z załadowaniem plików, po chwili namysłu okazało się że nie usunąłem pliku *initrd-custom-5.12.6-smp.gz* z poprzedniej instalacji, tak więc wykonałem polecenie *rm* (Rysunek 71).

Rysunek 71. Usunięcie initrd-custom-5.12.6.gz

Zrobiłem reboot, system znów wykrył poprawnie nowy system operacyjny (Rysunek 69). Wybrałem go i wcisnąłem Enter. Tym razem udało się poprawnie załadować system i do niego zalogować (Rysunek 72).



Rysunek 72. Logowanie do systemu

Tak więc po rozwiązaniu problemów instalacja przebiegła pomyślnie.

3. Wnioski

Sam w sobie proces kompilacji i instalacji przegiegł bez problemów, zarówno w przypadku pierwszej jak i drugiej metody. Problem się pojawił podczas próby uruchomienia systemu z drugiej metody. Zapomniałem usunąć jeden plik, który teoretycznie problemów nie powinien stwarzać, bo w lilo.conf podałem dokładną ścieżkę do odpowiedniego initrd, jednak mimo wszyskto odpalił się kernel panic, co było niemałym zaskoczeniem. Po przeczytaniu błędów jakie wyrzucał kernel i chwili namysłu postanowiłem usunąć ten plik initrd i jak się okazało to był problem. Także chwila nieuwagi i można się wpędzić w godziny debugowania i szukania przyczyn błędu(tu akurat mi to zajęło chwilę, jednak potrafię sobie wyobrazić scenariusz gdzie błąd nie byłby tak łatwy do zlokalizowania i naprawy).

Odnośnie odczuć w generowaniu plików konfiguracyjnych różnymi metodami – według mnie obie metody są ok, lecz ta z wykorzystaniem skryptu Perl jest o tyle fajniejsza, że można pominąć zbędne moduły, co przy słabszych maszynach może ratować sytuację.

Czas ładowania systemu jest w zasadzie taki sam. Różnica jest dla mnie niezauważalna.

Rysunek 73 przedstawia moduły załadowane w systemie po konfiguracji metodą starą, natomiast Rysunek 74 moduły załadowane w systemie z konfiguracją wygenerowaną za pomocą skryptu Perl. W obu przypadkach liczba modułów jest identyczna. Jest to prawdopodobnie spowodowane tym, że jądra były kompilowane na czysto zainstalowanej dystrybucji, więc nie było tam zbędnych modułów które mogłyby zostać uwzględnione w generowaniu pliku konfiguracyjnego starą metodą.

```
Linux 5.12.6-smp.
Last login: Tue May 25 21:41:20 +0200 2021 on /dev/tty1.
You have mail.
root@slack:~# lsmod
Module                  Size  Used by
ipv6                  397312  14
cfg80211              684032  0
rfkill                 24576  1 cfg80211
fuse                  102400  1
i2c_dev                16384  0
snd_intel8x0           36864  0
snd_ac97_codec        106496  1 snd_intel8x0
vmwgfx                258048  1
ac97_bus               16384  1 snd_ac97_codec
crc32_pclmul           16384  0
drm_kms_helper        208896  1 vmwgfx
snd_pcm                94208  2 snd_ac97_codec,snd_intel8x0
syscopyarea            16384  1 drm_kms_helper
snd_timer              32768  1 snd_pcm
sysfillrect            16384  1 drm_kms_helper
snd                    65536  4 snd_ac97_codec,snd_timer,snd_intel8x0,snd_pcm
sysimgblt              16384  1 drm_kms_helper
fb_sys_fops            16384  1 drm_kms_helper
ttm                    57344  1 vmwgfx
e1000                 102400  0
intel_agp              16384  0
drm                   417792  4 vmwgfx,ttm,drm_kms_helper
psmouse               122880  0
evdev                  20480  4
i2c_piix4              20480  0
intel_gtt              20480  1 intel_agp
serio_raw              16384  0
soundcore              16384  1 snd
agpgart                36864  4 intel_agp,intel_gtt,ttm,drm
i2c_core               65536  5 i2c_piix4,psmouse,i2c_dev,drm_kms_helper,drm
battery                20480  0
video                  45056  0
ac                     16384  0
button                 16384  0
loop                   28672  0
root@slack:~#
```

Rysunek 73. Moduły załadowane w systemie z konfiguracją wygenerowaną starą metodą

```
slack login: root
Password:
Linux 5.12.6-smp.
Last login: Wed May 26 01:14:25 +0200 2021 on /dev/tty1.
You have mail.
root@slack:~# lsmod
Module                  Size  Used by
ipv6                  397312  14
cfg80211              684032  0
rfkill                 24576  1 cfg80211
fuse                  102400  1
i2c_dev                16384  0
vmwgfx                258048  1
drm_kms_helper        208896  1 vmwgfx
snd_intel8x0           36864  0
snd_ac97_codec        106496  1 snd_intel8x0
crc32_pclmul           16384  0
syscopyarea            16384  1 drm_kms_helper
sysfillrect            16384  1 drm_kms_helper
ac97_bus               16384  1 snd_ac97_codec
sysimgblt              16384  1 drm_kms_helper
fb_sys_fops            16384  1 drm_kms_helper
snd_pcm                94208  2 snd_ac97_codec,snd_intel8x0
intel_agp              16384  0
ttm                    57344  1 vmwgfx
psmouse               122880  0
drm                   417792  4 vmwgfx,ttm,drm_kms_helper
evdev                  20480  4
serio_raw              16384  0
intel_gtt              20480  1 intel_agp
snd_timer              32768  1 snd_pcm
snd                    65536  4 snd_ac97_codec,snd_timer,snd_intel8x0,snd_pcm
agpgart                36864  4 intel_agp,intel_gtt,ttm,drm
i2c_piix4              20480  0
e1000                 102400  0
i2c_core               65536  5 i2c_piix4,psmouse,i2c_dev,drm_kms_helper,drm
soundcore              16384  1 snd
battery                20480  0
ac                     16384  0
video                  45056  0
button                 16384  0
loop                   28672  0
root@slack:~#
```

Rysunek 74. Moduły załadowane w systemie z konfiguracją wygenerowaną nową metodą